UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

DEPARTAMENTO DE INFORMÁTICA
DE SISTEMAS Y COMPUTADORES

Improvement of interconnection networks for clusters: direct-indirect hybrid topology and HoL-blocking reduction routing

A thesis submitted in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy
(Computer Engineering)

Author
Roberto Peñaranda Cebrian
Advisors
Prof. Pedro Juan López Rodríguez
Prof. María Engracia Gómez Requena

January 2017

Agradecimientos

Hace ya unos años que empecé este proyecto, cuando mis directores Pedro y María Engracia me dieron la oportunidad de meterme en el mundo de la investigación. Por ello y por todo los conocimientos transmitidos, y no solo sobre la temática de la tesis, les doy las gracias. También agradezco la participación de Crispín, que ha sido como un hermano mayor dentro de la investigación al principio de mi tesis. Por otro lado, me gustaría mencionar a Julio Sahuquillo quien me fue introduciendo en este mundo cuando estaba terminando la carrera universitaria. Por ese motivo me gustaría darle las gracias por encender la chispa de curiosidad sobre la investigación.

También me gustaría agradecer la paciencia, el apoyo y el cariño que me han ofrecido mis padres Patricio y Esperanza y a mi hermano Cristian. Y como no, dar las gracias al resto de la familia y a mis amigos, que en algunos momentos han ofrecido ayuda y comprensión.

También agradecer el buen ambiente de trabajo, casi siempre, en el laboratorio y de los profesores dentro del grupo de de investigación GAP. Con mención especial a Ricardo, que siempre intentaba contentar las necesidades de todos. También agradecer esas charlas sobre películas, series o incluso astrofísica a la hora del café.

¡Muchas gracias por todo!

Contents

Abstract

Nowadays, clusters of computers are used to solve computation intensive problems. These clusters take advantage of a large number of computing nodes to provide a high degree of parallelization. Interconnection networks are used to connect all these computing nodes. The interconnection network should be able to efficiently handle the traffic generated by this large number of nodes.

Interconnection networks have different design parameters that define the behavior of the network. Two of them are the topology and the routing algorithm. The topology of a interconnection network defines how the different network elements are connected, while the routing algorithm determines the path that a packet must take from the source to the destination node. The most commonly used topologies typically follow a regular structure and can be classified into direct and indirect topologies, depending on how the different network elements are interconnected. On the other hand, routing algorithms can also be classified into two categories: deterministic and adaptive algorithms.

To evaluate interconnection networks, metrics such as latency or network productivity are often used. Throughput refers to the traffic that the network is capable of accepting the network per time unit. On the other hand, latency is the time that a packet requires to reach its destination. This time can be divided into two parts. The first part is the time taken by the packet to reach its destination in the absence of network traffic. The second part is due to network congestion created by existing traffic. One of the effects of congestion is the so-called *Head-of-Line blocking*, where the packet at the head of a queue blocks, causing the remaining queued packets can not advance, although they could advance if they were at the head of the queue.

Nowadays, there are other important factors to consider when interconnection networks are designed, such as cost and fault tolerance. On the one hand, a high performance is desirable, but without a disproportionate increase in cost. On the other hand, the fact of increasing the size of the network implies an increase in the network components, thus the probability of occurrence of a failure is higher. For this reason, having some fault tolerance mechanism is vital in current interconnection networks of large machines. Putting all in a nutshell, a good performance-cost ratio is required in the network, with a high level of fault-tolerance.

This thesis focuses on two main objectives. The first objective is to combine the advantages of the direct and indirect topologies to create a new family of topologies with the best of both worlds. The main goal is the design of the new family of topologies capable of interconnecting a large number of nodes being able to get very good performance with a low cost hardware.

The family of topologies proposed, that will be referred to as k-ary n-direct s-indirect, has a n dimensional structure where the k different nodes of a given dimension are interconnected by a small indirect topology of s stages. We will also focus on designing a deterministic and an adaptive routing algorithm for the family of topologies proposed.

Finally we will focus on analyzing the fault tolerance in the proposed family of topologies. For this, the existing fault tolerance mechanism for similar topologies will be studied and a mechanism able to exploit the features of this new family will be designed.

The second objective is to develop routing algorithms specially deigned to reduce the pernicious effect of *Head-of-Line blocking*, which may shoot up in systems with a high number of computing nodes. To avoid this effect, routing algorithms able of efficiently classifying the packets in the different available virtual channels are designed, thus preventing that the occurrence of a hot node (Hot-Spot) could saturate the network and affect the remaining network traffic.

Resumen

Hoy en día, los clusters de computadores son usados para solucionar grandes problemas. Estos clusters aprovechan la gran cantidad de nodos de computación para ofrecer un alto grado de paralelización. Para conectar todos estos nodos de computación, se utilizan redes de interconexión de altas prestaciones capaces de manejar de forma eficiente el tráfico generado por esta gran cantidad de nodos.

Las redes de interconexión tienen diferentes parámetros de diseño que definen su comportamiento, de los cuales podríamos destacar dos: la topología y el algoritmo de encaminamiento. La topología de una red de interconexión define como se conectan sus componentes, mientras que el algoritmo de encaminamiento determina la ruta que un paquete debe tomar desde su nodo origen hasta su nodo destino. Las topologías más utilizadas suelen seguir una estructura regular y pueden ser clasificadas en topologías directas e indirectas, dependiendo de cómo estén interconectados entre si los diferentes elementos de la red. Por otro lado, los algoritmos de encaminamiento también pueden clasificarse en dos categorías: algoritmos deterministas y adaptativos.

Para evaluar las redes de interconexión se suelen utilizar medidas tales como la latencia o la productividad de la red. La productividad mide el tráfico que es capaz de aceptar la red por unidad de tiempo. Por otro lado, la latencia mide el tiempo que utiliza un paquete para alcanzar su destino. Este tiempo se puede dividir en dos partes. La primera corresponde al tiempo utilizado por el paquete en alcanzar a su destino en ausencia de tráfico en la red. La segunda parte sería la debida a la congestión de la red creada por el tráfico existente. Uno de los efectos de la congestión es el denominado *Head-of-Line blocking*, donde el paquete que encabeza una cola se queda bloqueado, por lo que el resto de paquetes de la cola no pueden avanzar, aunque pudieran hacerlo si ellos encabezaran dicha cola.

Hoy en día hay otros factores importantes a tomar en cuenta cuando se diseñan redes de interconexión, como son el coste y la tolerancia a fallos. Por lo tanto, las prestaciones deben mantenerse conforme aumentamos el tamaño de la red, pero sin un aumento prohibitivo en el coste. Además, el hecho de aumentar el tamaño de la red implica un aumento en el número de elementos de dicha red, con lo que la probabilidad de la aparición de un fallo es mayor. Por ese motivo, es vital contar con algún mecanismo de tolerancia a fallos en las redes de interconexión para los grandes supercomputadores actuales. En otras palabras, es de esperar una buena relación coste-prestaciones con un alto nivel de tolerancia a fallos.

Esta tesis tiene dos objetivos principales. El primer objetivo combina las ventajas de las topologías directas e indirectas para crear una nueva familia de topologías con lo mejor de ambas. En concreto, nos centramos en el diseño de una nueva familia de topologías capaz de interconectar una gran cantidad de nodos siendo capaz de obtener muy buenas prestaciones con un bajo coste hardware. La familia de topologías propuesta, que hemos llamado k–ary n–direct s–indirect, tiene una estructura n-dimensional, donde los diferentes k nodos de una dimensión se conectan entre si mediante una pequeña topología indirecta con s etapas. También diseñaremos un algoritmo de encaminamiento determinista y otro adaptativo para la familia de topologías propuesta.

Finalmente, nos centraremos en estudiar la tolerancia a fallos para la familia de topologías propuesta. Para ello se estudiarán los mecanismos de tolerancia a fallos existentes en topologías similares y se diseñará un mecanismo capaz de aprovechar al máximo las características de esta nueva familia.

El segundo objetivo consiste en el desarrollo de algoritmos de encaminamiento capaces de evitar el pernicioso efecto *Head-of-Line blocking*, lo cual puede aumentar rápidamente en sistemas con un gran número de nodos de computación. Para evitar este efecto se diseñaran algoritmos de encaminamiento capaces de clasificar de forma eficiente los paquetes en los diferentes canales virtuales disponibles, evitando así que la aparicion de un punto caliente (Hot-Spot) sature la red y perjudique a todo el tráfico de la red.

Resum

Hui en dia, els clústers de computadors són utilitzats per solucionar grans problemes computacionals. Aquests clústers aprofiten la gran quantitat de nodes de computació per a oferir un alt grau de parallelització. Per a connectar tots aquests nodes de computació, s'utilitzen xarxes d'interconnexió d'altes prestacions capaços de manejar de manera eficient el trànsit generat per aquesta gran quantitat de nodes.

Les xarxes de interconnexió tenen diferents parámetres de disseny que defineixen el seu comportament, dels quals podríem destacar dues: la topologia i l'algoritme d'encaminament. La topologia d'una xarxa de interconnexió ens defineix com es connecten els seus components, mentre que l'algoritme d'encaminament determina la ruta que un paquet ha de prendre des del seu node origen fins al seu node destí. Les topologies més utilitzades solen seguir una estructura regular i poden ser classificades en topologies directes i indirectes, depenent de com estiguen interconnectats entre si els diferents elements de la xarxa. D'altra banda, els algoritmes d'encaminament també poden classificar-se en dues categories: algoritmes deterministes i adaptatius.

Per avaluar les xarxes de interconnexió es solen utilitzar mesures com ara la latència o la productivitat de la xarxa. La productivitat mesura el trànsit que és capaç d'acceptar la xarxa per unitat de temps. D'altra banda, la latència mesura el temps que utilitza un paquet per arribar al seu destí. Aquest temps es pot dividir en dues parts. La primera correspon al temps emprat pel paquet a aconseguir al seu destí en absència de trànsit a la xarxa. La segona part seria la deguda a la congestió de la xarxa creada per el trànsit existent. Un dels efectes de la congestió és l'anomenat *Head-of-line blocking*, on el paquet que encapçala una cua es queda bloquejat, de manera que la resta de paquets de la cua no poden avançar, encara que poguessen fer-ho si ells encapçalessen la dita cua.

Hui en dia hi ha altres factors importants a tenir en compte quan se dissenyen xarxes de interconnexió, com són el cost i la tolerància a fallades. Per tant, les prestacions s'han de mantenir d'acord augmentem la mida de la xarxa, però sense un augment prohibitiu en el cost. A més, el fet d'augmentar la mida de la xarxa implica un augment en el número de elements d'aquesta xarxa, de manera que la probabilitat de l'aparició d'una fallada és més gran. Per aquest motiu, és vital comptar amb algun mecanisme de tolerància a fallades en les xarxes d'interconnexió per als gran supercomputadors actuals. En altres paraules, és d'esperar bona relació cost-prestacions amb una alta tolerància a fallades.

Aquesta tesi té dos objectius principals. El primer objectiu combina les avantatges de les topologies directes i indirectes per a crear una nova família de topologies amb el millor dels dos mons.

En concret, ens centrem en el disseny de una nova família de topologies capaç d'interconnectar una gran quantitat de nodes sent capaç d'obtenir molt bones prestacions amb un baix cost hardware. La família de topologies proposada, que hem nomenat k-ary n-direct s-indirect, té una estructura n-dimensional, on els diferents k nodes d'una dimensió se connectan entre si mitjançant una petita topologia indirecta amb s etapes. També dissenyarem un algoritme d'encaminament determinista i un altre adaptatiu per a la família de topologies proposta.

Finalment, ens centrarem en estudiar la tolerància a fallades per a la família de topologies proposada. Per a això s'estudiaràn els mecanismes de tolerància a fallades existents en topologies similars i es dissenyarà un mecanisme capaç d'aprofitar al màxim les característiques d'aquesta nova família.

El segon objectiu consisteix en la creació d'algoritmes d'encaminament capaços d'evitar el perniciós efecte *Head-of-line blocking* que pot créixer rápidament amb un gran número de nodes de computació. Per a evitar aquest efecte es dissenyaran algoritmes d'encaminament capaços de classificar de forma eficient els paquets en els diferents canals virtuals disponibles, evitant així que l'aparició d'un punt calent (Hot-Spot) sature la xarxa i perjudique tot el trànsit de la xarxa.

List of Figures

List of Tables

Abbreviations and Acronyms

Adp	**Ad**a**p**tive
HoL	**H**ead **o**f **L**ine
KNS	**k**-ary **n**-direct **s**-indirect
IODET	**I**n **O**rder **DET**erministic routing
OODET	**O**ut of **O**rder **DET**erministic routing
XORDET	**XOR DET**erministic routing
XORADAP	**XOR ADAP**tive routing
VC	**V**irtual **C**hannel

Chapter 1

Introduction

This chapter introduces some concepts and presents the motivation for this thesis. First, we describe the problem that justifies the development of this thesis. After that, some tasks are proposed in order to achieve the main purposes of this thesis.

1.1 Motivation and Justification

The computing power required by statistical and scientific problems is continuously increasing. Due to this, supercomputers are used to resolve these problems. Nowadays, most supercomputers are based on clusters of computers. The topmost machines of the top 500 supercomputer list [1] are being built up by using hundreds of thousands of processing nodes. All these processing nodes work jointly to solve a given problem in as less time as possible. For this joint work, they need an interconnection network that allows all the processing nodes to share data among them. This communication among all the processing nodes must be as efficient as possible because it strongly impacts the performance of the whole system. Transmission time of data across the interconnection network adds up to the processing time, impacting the time required to run the applications. Therefore, the performance of this interconnection network must be taken into account when designing a supercomputer.

Three of the most important design issues of interconnection networks are the topology, the routing algorithm and the switching technique [2, 3]. Topologies define how the components of the system are connected. The routing algorithm determines the path that is followed by packets from their source to their destination node. Switching techniques describe how packets are stored and forwarded to the next routing component. In this thesis, we focused on virtual cut-through, where the buffers within the routing component must have enough space to store a complete packet, although they do not need the complete packet to forward it to the next routing component.

To evaluate different interconnection network configurations, measures like latency or throughput are often used. Throughput is the amount of data that the network can deliver by time unit. On the other hand, latency is the time that a packet needs to reach its destination. This time can be divided into two parts. The first is the time taken by the packet to reach its destination in the absence of network traffic. The second part would be due to network congestion due to existing traffic. One of the effects created by the congestion is the so-called *Head-of-Line blocking*, where the packet at the head of a queue blocks, causing the remaining queued packets can not advance, although they could advance if they were at the head of the queue because the demanded resources are free.

Due to the large size of current supercomputers, these metrics need to be complemented by other. Nowadays, there are a large amount of components that increase the cost of the network. For this

reason, the design of the interconnection network must be also aware of the cost, and this cost should not disproportionately increase with the network size. Moreover, the higher the number of network components, the higher the probability of failure. Therefore, the capability of the network to tolerate failures is also extremely important, as it must affect as little as possible the network performance and cost.

1.1.1 Topologies

The topology features heavily impacts the network performance and cost. Network latency depends largely on the network diameter, which measures the maximum distance between two nodes of the topology using the shortest possible path between them. On the other hand, the maximum throughput reachable by the network is limited by the bisection bandwidth. Therefore, these aspects must be considered when a designer has to choose a network topology or when (s)he has to develop a new one because performance will highly depend on them. There are other factors to consider by the designer, as an easy implementation and the cost of the network. It is quite common to use regular structures to simplify their implementation and the possibility of expanding the network. Among the different taxonomies of regular topologies, the most commonly-used one divides them into direct and indirect topologies [2, 3].

1.1.1.1 Direct Topologies

In this case, each node is composed by its own router and its processing node in direct topologies. This router connects the node to a subset of the nodes of the network through point-to-point links. This kind of topologies usually adopt an orthogonal structure, where nodes are organized in an n–dimensional space. Each node has at least one link in each dimension, being regularly organized through all dimensions. Therefore, traversing one link produces a displacement in only one dimension. This design offers symmetry and regularity which greatly simplify its implementation and the routing algorithm. The most commonly–used direct topologies are the mesh, the torus and the hypercube. They are usually referred to as k–ary n–cubes. In what follows, to distinguish the topology parameters of both direct and indirect topologies, we will refer to this topology as k_d–ary n_d–cubes[1]. In this case, k_d represents the number of nodes in each one of the n_d dimensions. The total number of nodes in the system is given by $N = {k_d}^{n_d}$. These nodes are labeled by an identifier with as many components as topology dimensions

[1]The subscript "$_d$" stands for direct.

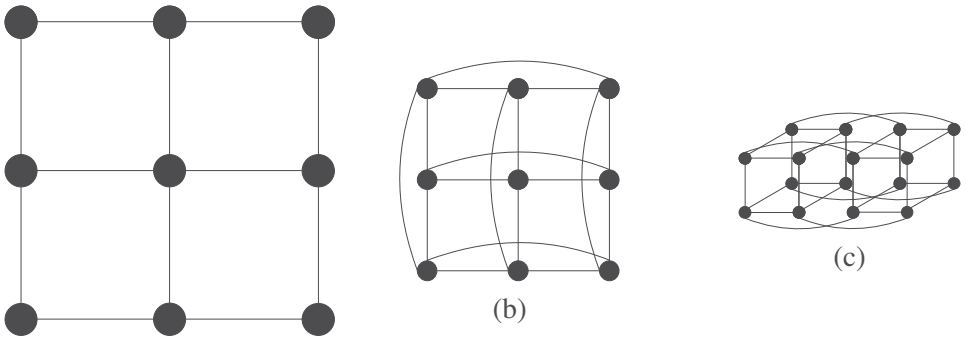


FIGURE 1.1: Some of the most commonly-used direct topologies.

TABLE 1.1: Parameters of the different direct topologies.

Topology	Diameter	Bisection width
Mesh	$n_d(k_d - 1)$	$2N/k_d$
Torus	$n_d(k_d/2)$	$4N/k_d$
Hypercube	n_d	N

$\{p_{n_d-1}, \ldots, p_0\}$. The component associated to each dimension ranges from 0 to $k_d - 1$ (i.e., nodes are numbered from $\langle 0, 0, \ldots, 0\rangle$ to $\langle k_d-1, k_d-1, \ldots, k_d-1\rangle$). The identifiers of neighbor nodes in a given dimension only differ in the component corresponding to that dimension, while the remaining components have the same value. For instance, two nodes p y p' are neighbors in the x dimension, if and only if $p_x = p'_x \pm 1$ and $p_i = p'_i$ for the rest of dimensions.

In a mesh topology, Figure 1.1.(a), all the nodes with the same coordinates except one compose a linear array. In torus, Figure 1.1.(b), all the nodes with the same coordinates except one form a ring. The hypercube is a particular case of a mesh where there are only two nodes in each linear array ($k_d = 2$), which forces the number of dimensions (n_d) to be large enough to interconnect all the nodes of the system (i.e., $n_d = log_2 N$). Direct topologies have been used in several of the most powerful supercomputers, being the 3D-torus the most widely used one. For instance, the number two (Titan) and number three (Sequoia) of the November 2015 Top500 supercomputer list [1] are based on a torus.

For a given number of processing nodes N, direct topologies provide a richer connectivity as the number of dimensions increases. As the hypercube has the largest possible number of dimensions, it provides a better connectivity than meshes and tori, but at a higher cost, since it uses more links and it has a higher router degree (number of ports of the router).

Table 1.1 shows the diameter and bisection width for these topologies. For a constant number of processing nodes ($N = k_d^{n_d}$), diameter is increased as the number of dimensions (n_d) is decreased or, in other words, the number of nodes per dimension (k_d) is increased. The increase of the distance traversed by packets also increases the probability of contention with other packets that are also crossing the network, which reduces network throughput. Moreover, the bisection bandwidth is decreased with a lower number of dimensions. From this point of view, it may seem interesting to maximize the number of dimensions of direct topologies for a given number N of processing nodes. Nevertheless, other issues have to be considered. Direct topologies with up to three dimensions can be implemented by using relatively short links in the 3-D physical space and with an acceptable wiring complexity, regardless of the size of the system. However, implementing a topology with more than three dimensions implies increasing the length of the links, since we have to map all the dimensions to the 3-D physical space. In addition, they also require using routers with a higher number of ports (two bidirectional ports per each dimension are required, one per each direction in that dimension). So, direct topologies have some limitations for the implementation of large-scale machines, since they would require using a large number of nodes per dimension (k_d), which increases the diameter of the network and also the probability of contention, which increases message latency of communications and reduces network throughput.

1.1.1.2 Indirect Topologies

In indirect topologies, the processing node and the router are two different components. Therefore, the processing nodes are not able to route packets. In this kind of topologies, the routers are called *switches*. In this case, the processing nodes are connected to the network through switches. However, unlike direct topologies, there are switches in the network that are connected only to other switches, without connecting to any processing node. In fact, most of switches are not connected to any processing node.

The most common indirect topologies are the multistage interconnection networks (MINs), where the switches are organized in stages. The processing nodes are connected to the switches of the first stage and the switches from different stages are connected using any pattern in order to offer full connection between processing nodes. MINs can be divided into unidirectional and bidirectional MINs, depending on the used links. In UMINs the last stage is connected to the processing nodes and packets must traverse all stages by an unique path. The diameter in this

TABLE 1.2: Parameters of the different direct topologies.

Topology	Diameter	Bisection width
k–ary n–tree	$2n_i$	N
RUFT	n_i	N

case is always equal to the number of stages. On the other hand, BMINs have two routing steps, packets first travel upwards the network and then downwards. In the worst case, the packets traverse all stages twice. But, if the packet is able to reach the destination in any stage when it is going upwards, it can start to go downwards. Besides that, BMINs offer several different paths to reach a given destination from a given source.

Different MINs have been proposed in the literature, depending on the connection patterns that have been used to interconnect the adjacent stages. The most widely–used MIN in commercial systems is the fat–tree topology [4], which is a BMIN. However, there are other more recent proposals, like the one proposed in [5] or [6], where the second one is focused on fault tolerance. Most of the high–performance interconnect vendors as Mellanox (manufacturer of the Infiniband technology) [7] or Myricom (manufacturer of Myrinet) [8] recommend to use a fat–tree and provide specific switches for building this topology. Moreover, it has been used in some of the most powerful supercomputers. For instance, the Tianhe 2 and the Tianhe 1A supercomputers, the number one and twenty six, respectively, in the November 2015 Top500 list [1] use this topology.

One of the most used implementation of a fat–tree topology is the k–ary n–tree. In what follows, to distinguish the topology parameters of both direct and indirect topologies, we will refer to this topology as k_i–ary n_i–tree[2]. k_i is the number of links that connect a switch to the next or the previous stage, and n_i is the number of stages of the indirect network. So, each switch has $2k_i$ bidirectional ports in fat–trees (i.e. using switches with $d \times d$ ports, k_i is $d/2$). A fat–tree requires at least $log_{k_i} N$ stages to interconnect N processing nodes. Each stage has N/k_i switches, with a total of $(N/k_i) log_{k_i} N$ switches and $N = {k_i}^{n_i}$ processing nodes.

Table 1.2 shows the diameter and bisection bandwidth for some of these topologies. In k–ary n–tree, the diameter of the network depends only on the number of stages, and it is given by $2n_i$, that is $2log_{k_i} N$, as in the worst case a packet must traverse upwards all stages and all stages downwards. Notice that, for a UMIN, the distance traversed by packets is always n_i, regardless

[2]The subscript "$_i$" stands for indirect.

of the source and destination processing nodes. In summary, for a fixed number $N = {k_i}^{n_i}$ of processing nodes of a MIN, when the number of stages n_i is increased, k_i is decreased but the distance that packets have to traverse to reach the destination is increased. Thus, it should be taken into account that, by using high-degree switches fewer switches will be required, but each of them will be more complex. On the contrary, if we use low–degree switches, we will require more switches, but much simpler.

1.1.1.3 Other Topologies

There have been attempts to design other topologies, but they have been never or seldom used in commercial products or in supercomputers. For example, in [9], WK–recursive topology was presented for interconnection networks and, later, for on–chip networks ([10]). Moreover, this topology has difficulties to guarantee deadlock freedom in the routing algorithm.

The flattened–butterfly topology was presented in [11]. This topology is obtained by combining the switches that are at the same position in each stage of a conventional butterfly MIN. As a result, we obtain an n-dimensional direct network where the nodes of the same array are not connected by a ring like in a torus. Instead of that, they are fully connected. This topology is similar to a generalized hypercube but, in this case, attaching several processing nodes to the same switch. This fact is called concentration, and is used in several topologies. Like in the generalized hypercube, the fact of having this connectivity highly impacts their cost.

Other works are based on hierarchical topologies. They usually use two different subnetworks, a local and a global one. Hierarchical designs are expected to have a higher latency and smaller throughput, since both networks must be traversed for most of the source–destination pairs. One example of this kind of topologies is proposed in [12], using as local network a simple bus, and a mesh as global network. The authors of [13] propose a tool to select the most suitable topology for a given network design. The tool explores the design space of hybrid Clos-torus networks. In particular, the explored designs are hierarchical topologies, where local networks are Clos networks and they are connected by a global torus network. Another recent hierarchical topology is the DragonFly [14, 15]. This topology groups the different routers in virtual routers to increase the effective radix of the network. To do this, it uses two different networks, one intra–group (local), and another one inter–group (global). The authors recommend to use a non-minimal global adaptive routing algorithm to balance the load across the global channels.

These global channels, which link different groups, are long links, so that a high latency can be expected.

In [16] a topology that combines several tori networks is proposed. The proposal focus on large supercomputers, but its applicability is limited to 2^{16} processing nodes and its wiring layout is complex for large machines. The proposal starts from a 2-D torus and provides bypass links in the diagonal direction as many times as needed. This proposal does not improve the number of hops in a single dimension, since no new links are added to connect nodes of the same dimension, but reduces the number of hops when traversing several dimensions. In addition, this topology has the same problem with deadlock-free routing than the WK–recursive one.

Other works propose the combination of several topologies, to reach the good properties of both of them. Most of them have been proposed for on–chip networks and therefore the target is different to ours. For example, in [17], each core is connected to two different tree networks in an on-chip environment in order to overcome the poor performance provided by trees. This proposal is not suitable for large machines due to the complexity of the cable layout and the poor performance achieved even with two trees. Another topology is the mesh of trees (MoT) introduced in [18] and later used also in NoCs [19]. It is based on an $n-$dimensional topology where the nodes of a given dimension are connected using a tree. This results in a topology with very poor expected performance due to using a simple tree for connecting the nodes of a dimension. The Bcube [20] and Hypercrossbar network [21] were also proposed. Each processing node is connected to several dimensions by using several NICs. A switch is used to connect the processing nodes of the same dimension. However, routing is performed through end-processing nodes, by ejecting packets from the network through a NIC and later reinjecting them through another one.

1.1.2 Routing Algorithms

The routing algorithm is an important design parameter in interconnection networks. It establishes how the packets cross the network in order to reach their destination. The chosen path could highly affect the performance depending on the traffic pattern that the network is handling. For this reason, choosing the appropriate routing algorithm is extremely important.

Routing algorithms can be categorized in different ways. For example, a routing algorithm can be minimal routing or non-minimal routing. With minimal routing, a packet only performs the

strictly needed hops to reach its destination node. In other words, all paths given by the routing algorithm between a source node and its destination node are minimal paths.

Orthogonally, routing algorithms can be also classified depending on the number of paths that it offers for a given pair of nodes. If there are multiple available paths, the traffic flow is able to change, adapting to unforeseen network issues. However, if only one path is returned by the routing algorithm for a source-destination pair, with a deterministic behavior, a change on the status of the network could highly affect its performance. According to this criteria, the routing algorithms can be divided into adaptive and deterministic algorithms.

When designing a routing algorithm, we have to take account that the algorithm has to be deadlock-free and livelock-free. When a deadlock appears in the network, packets are blocked and they can never advance. However, in livelocks, packets can advance but they are not able to reach their destination.

Ensuring deadlock and livelock freedom is more difficult when using adaptive routing algorithms. There are different techniques to cope with these issues. Most of them use extra resources, like virtual channels, to solve the problem. For instance, the Duato's protocol allows the use of one or more virtual channels without restrictions, provided that there are one or more escape channels that are deadlock-free. On the other hand, there are other techniques where extra resources are not needed. They usually introduce some restrictions, like Dimension Order Router (DOR) for direct topologies, that forces to cross the network in a given order, or bubble technique for rings, that requires buffer space available for more than to packets to allow new packet injections.

One of the most widely used adaptive routing algorithm for direct topologies is the fully adaptive routing algorithm that has several adaptive virtual channels, that allows packets to cross dimensions in any order, and on escape channels with DOR. Obviously, this algorithm is valid for orthogonal topologies. At the same time, DOR is widely used as a deterministic routing algorithm in orthogonal topologies. Besides this, torus topologies need to solve the deadlock problem in their rings, for instance using the bubble technique.

For k–ary n–tree topologies, the adaptive routing algorithm does not need extra virtual channels. In this case, the packet can could go through any port in the first phase, when the packet goes upwards. In the second phase, when the packet goes downwards, a deterministic algorithm must be used, since there is only one path to the destination node. DESTRO [22] is a deterministic

routing algorithm focused on k–ary n–tree topologies. In this case, the algorithm offers only one path for each source-destination pair. Moreover, this algorithm helps to avoid the Head of Line (HoL) blocking effect as it classifies different destinations to different ports.

1.1.2.1 Classifying Destinations into VCs

As mentioned above, adaptive routing provides more flexibility to forward packets in the network through the different VCs in direct or dimensional topologies. This routing freedom has two opposite effects. The positive one is that temporally congested network areas can be avoided. However, the negative effect is that packets with different destination nodes may be highly interleaved in the switch queues, which significantly increases the HoL-blocking effect, which could highly degrade network performance. For instance, if there is a hotspot node, where a lot of traffic is forwarded to this hotspot node, an adaptive routing algorithm freely spread all this hotspot traffic throughout the network, blocking output channels and generating the HoL-blocking effect. As deterministic routing does not offer such flexibility, its contribution to increase the HoL-blocking effect is lower.

Some authors have proposed algorithms that are able to reduce the HoL-blocking effect using VCs. To do this, the authors propose to classify destinations into virtual channels using some rules. These algorithms can do it dynamically or statically. In this thesis, we have focused on static algorithms. In general, they use this classification in conjunction with a deterministic routing algorithm like DOR.

For instance, in [23], VOQnet was presented. This algorithm associates each destination to a different virtual channel. Therefore, it needs as many virtual channels as destination nodes. Although this algorithm completely removes the HoL-blocking effect, it requires a high number of virtual channels that grows with the size of the network, and this is unaffordable in many cases. However, it is used for comparison purposes since it provides an upper bound that could be achieved by completely removing HoL-blocking from the network.

VOQsw was proposed in [24] as a simplification which needs less resources. In this case, the algorithm only solves the local congestion inside the router. It requires as many virtual channels as the number of output ports of the switch. In VOQsw, the packet is assigned to a virtual channel depending on the next output port that the packet will use. VOQsw does not solve

FIGURE 1.2: How DBBM assigns destinations to VCs in a 8×8 mesh with 4 VCs.

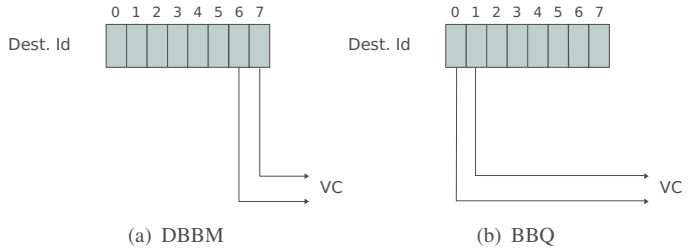
TABLE 1.3: How many destinations DBBM assigns to node 0 VCs in a 8×8 mesh with 4 VCs.

Dim	VC#0	VC#1	VC#2	VC#3
X	8	16	16	16
Y	7	No dest.	No dest.	No dest.

global congestion, so, it leads to a worse classification of packets than VOQnet and it is also not scalable, as the number of required VCs depends on the switch degree.

In order to be more scalable requiring less resources, DBBM was introduced in [25]. This algorithm selects virtual channels by using some bits from the destination identifier. By default, it uses the least significant bits required to choose a virtual channel, that is, it uses the destination identifier modulo the number of virtual channels. Although it works quite well in indirect networks, when using DBBM in a torus or a mesh, it leads to an unbalanced classification of destinations. For instance, in a 2D torus or mesh, all the nodes in a given column are assigned to the same virtual channel as shown in Figure 1.2 for an 8x8 mesh with 4 virtual channels per physical channel.

Indeed, Table 1.3 shows the number of destinations assigned to different VCs for a node of the network (node 0). For instance, virtual channel #0 of the X-dimension is used to reach 8 nodes (the 4^{th} row), while virtual channel #1 of the X-dimension is used to reach 16 nodes (the 1^{st} and 5^{th} row). As it can be seen, all the virtual channels in the Y-dimension are never used but one per port. This lack of classification in the last dimension (Y) would lead to a huge congestion



(a) DBBM (b) BBQ

FIGURE 1.3: Implementation of virtual channel selection for a 256-node 2-D network and 4 VCs.

due to the HoL-blocking effect that, at the end, could be propagated to the whole network due to upstream flow control pressure.

On the other hand, the implementation of this mechanism is very simple, provided that the number of VCs is a power of two. In that case, the modulo operation by the number of virtual channels is as easy as selecting the $\log(\#VCs)$ least significant bits of the packet destination (see Figure 1.3(a)). Besides that, the assignment to a virtual channel only depends on the packet destination. Therefore, packets use the same virtual channel while they traverse the network. This is a nice feature, as virtual channel assignment can be done once at the source node, since the rest of nodes that a packet crosses across the network merely forwards the packet through the same virtual channel from which the packet arrived to. Indeed, this fact also leads to a very low switch complexity. Because there is not need to move packets in a switch from one input virtual channel to another output virtual channel with different identifier. The internal switch of the nodes can be implemented as one independent switch per virtual channel, instead of deploying a fully-connected crossbar.

To solve the problem with the poor classification in a 2D torus or mesh, BBQ mechanism [26] was proposed. This is a specific case of DBBM, where the destination identifier is also used to choose the virtual channel for each packet. Indeed, it uses the destination $\log(\#VCs)$ most significant bits (see Figure 1.3(b)). That is, BBQ divides the network in as many horizontal bands as virtual channels, in such a way that the nodes in each column are spread as much as possible among the virtual channels. However, the problem is that all the nodes inside each horizontal band use the same virtual channel, and, therefore they may suffer from HoL-blocking in the first dimension. As in DBBM, BBQ never changes the virtual channel of a packet, and it can be assigned once at injection time, keeping the same virtual channel along its path in the network.

1.1.3 Fault Tolerance

As mentioned above, the size of the network is rapidly growing and thus the probability of failure of any component. In order to keep the connectivity of the network in presence of failures, several solutions have been proposed.

Some solutions choose to replicate all network elements, and use these additional elements as spare components. However, these solutions are extremely expensive. Other solutions are focused on modifying the routing algorithm to be able to reach the destination nodes by alternative paths, avoiding regions or components with failures. There are two different categories of fault tolerant techniques that are based on routing configuration. The first category reconfigures the routing tables when a failure occurs. In this case, the routing tables have to be updated with the new topology after the failure [27–30]. Using this technique, the network can tolerate any number of faults without requiring extra resources [31], as long as the network is still connected. However, this technique needs to use topology-agnostic routing algorithms because the resulting topology is irregular. Therefore, the performance may be affected. That is, they do not consider the specific characteristics of the topology, thus, they often provide a worse traffic balance.

On the other hand, the second category covers fault-tolerant routing algorithms. Many algorithms for interconnection networks that are able to tolerate failures have been proposed, and specially for direct network topologies like tori and meshes. Some of these routing algorithms do not require extra resources, like [32, 33] or [34]. The first two methods are able to tolerate only a few faults for low dimensional meshes, while the third method can offer more tolerance against faults, but needs extra virtual channels in a torus. However, these routing algorithms provide a poor traffic balance as a lot of traffic is directed towards a single link, which can be easily saturated. There are other algorithms that require adding some resources, like virtual channels. Sometimes, the number of virtual channels depends on the number of tolerated faults [35] or the number of dimensions of the topology [36]. Other of these algorithms are based on disabling faulty regions [37–41] or individual nodes [42–44] to route the packets around these fault regions. However, to do this, these algorithms disable healthy nodes. Some authors use the technique provided by Valiant [45] to implement a routing algorithm that uses intermediate nodes to avoid faults [46]. This technique requires a few virtual channels but it does not disable healthy nodes.

1.2 Goals and Methodology of this Thesis

The work performed in this thesis has been developed following the typical steps followed in a research work.

First, we studied the state of the art. We have reviewed the previous work devoted to the field of research in order to obtain the required knowledge to develop this thesis. In particular, we have acquired background on topologies, routing and fault tolerance.

Next, we identified some problems to be solved and/or aspects that could be improved, also developing new proposals.

In particular we proposed a new family of hybrid (direct-indirect) topologies that is able to connect a huge number of nodes, trying to combine the best features of both direct and indirect topologies. The new family of topologies is referred to as k–ary n–direct s–indirect (or KNS for short). Routing algorithms for the new family of topologies were also developed. A fault-tolerant routing algorithm for the KNS was also developed.

Although initially intended for the KNS topology, we have also developed new routing algorithms for torus and meshes that are designed to reduce the HoL-blocking effect.

Once the new proposals are done, they must be evaluated and compared with existing ones.

To do so, we have prepared the necessary framework environment to implement the different proposals which are presented in this thesis. We have extended the simulator of our research group (Parallel Architecture Group (GAP)) which was originally developed in Modula-2. We recoded this tool in the *C* programing language because of the restrictions of Modula-2. We also modeled some of the most well-known topologies and routing algorithms in this simulator, including torus, meshes, k–ary n–tree, and flattened butterfly. This framework was used to evaluate the topology and the routing algorithms proposed in this thesis, also performing an in-depth comprehension with previous existing work.

1.3 Thesis Outline

Following the current UPV rules, this thesis has been written as a compendium of articles. Therefore, the rest of the thesis is organized as follows:

The Chapter 2 enumerates the conference or journal papers where the different proposals are included.

Chapters 3 to 11 include the different publications related to the work done in this thesis. They have been adapted to the required formatting style.

In Chapter 12, a general discussion of results of the main contributions of the thesis is presented.

Finally, in Chapter 13, some conclusions and ideas for future work are presented.

Chapter 2

Summary of Publications

Most of the proposals developed in this PhD thesis have been published in different conferences and journals. This chapter enumerates these publications

2.1 KNS topology

First, we developed the KNS topology and analyzed their different configurations. We compared their results obtained with other well-known topologies in order to obtain the benefits of using KNS topologies. The analysis does not consider only performance results but it also considers the cost of the network. This work resulted in two publications:

- R. Peñaranda, C. Gómez, M.E. Gómez, P. López, and J. Duato. **A New Family of Hybrid Topologies for Large–Scale Interconnection Networks.** In *11th IEEE International Symposium on Network Computing and Applications (NCA)*. This publication can be found in Chapter 3.
- R. Peñaranda, C. Gómez, M.E. Gómez, P. López, and J. Duato. **The k-ary n-direct s-indirect Family of Topologies for Large-Scale Interconnection Networks.** In *The Journal of Supercomputing*. This journal article can be found in Chapter 4.

2.2 Fault tolerance in KNS topology

We also focused on KNS fault-tolerance. We designed a fault-tolerance mechanism for KNS topologies which is able to tolerate several faults while achieving a low network performance degradation:

- R. Peñaranda, Ernst Gunnar Gran, Tor Skeie, M.E. Gómez, and P. López. **A New Fault-Tolerant Routing Methodology for KNS Topologies.** In *The 2nd IEEE International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB).*. Please see Chapter 5.

- R. Peñaranda, M.E. Gómez, P. López, Ernst Gunnar Gran, and Tor Skeie. **A Fault-Tolerant Routing Strategy for KNS Topologies Based on Intermediate Nodes.** In *Journal of Concurrency and Computation: Practice and Experience*. This journal article can be found in Chapter 6.

2.3 HoL-blocking reduction routing

Regarding HoL-blocking reduction in direct topologies, we designed three different routing algorithms. The first algorithm was IODET, which led to two different publications in different conferences:

- R. Peñaranda, C. Gómez, M.E. Gómez, P. López, and J. Duato. **IODET: A HoL-blocking-aware Deterministic Routing Algorithm for Direct Topologies.** In *IEEE 18th International Conference on Parallel and Distributed Systems (ICPADS)*. It is presented in Chapter 7.

- R. Peñaranda, C. Gómez, M.E. Gómez, P. López, and J. Duato. **Deterministic Routing with HoL-Blocking-Awareness for Direct Topologies.** In *International Conference on Computational Science (ICCS)*. It can be found in Chapter 8.

The second routing algorithm, XORDET, was designed as an improvement of IODET and it was published in:

- R. Peñaranda, C. Gómez, M.E. Gómez, P. López, and J. Duato. **HoL-blocking Avoidance Routing Algorithms in Direct Topologies.** In *IEEE International Conference on High Performance Computing and Communications (HPCC)*. It is shown in Chapter 9.

The last designed algorithm, the adaptive one, is XORADAP. This algorithm was presented in:

- R. Peñaranda, C. Gómez, M.E. Gómez, and P. López. **XORAdap: A HoL-blocking aware adaptive routing algorithm.** In *23rd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. Please see Chapter 10.

Both contributions were merged, explaining the use of XOR operation to reduce the HoL-blocking effect and including an extended evaluation analysis. We have submitted a journal paper, which is currently under review, in:

- R. Peñaranda, C. Gómez, M.E. Gómez, and P. López. **XOR-based HoL-blocking reduction Routing Mechanisms for Direct Networks.** In *Parallel Computing*. This journal article is shown in Chapter 11.

Chapter 3

A New Family of Hybrid Topologies for Large–Scale Interconnection Networks

Authors: Roberto Peñaranda (Universidad Politécnica de Valencia), Crispín Gómez (Universidad de Castilla La Mancha), María Engracia Gómez, Pedro López, Jose Duato (Universidad Politécnica de Valencia).

Type: Conference.

Conference: 11th IEEE International Symposium on Network Computing and Applications (NCA).

Location: Cambridge, MA, USA.

Year: 2012.

DOI: $http://dx.doi.org/10.1109/NCA.2012.22$

URL: $http://ieeexplore.ieee.org/xpls/abs\_all.jsp?arnumber=6299098$

Abstract

In large supercomputers the topology of the interconnection network is a key design issue that impacts the performance and cost of the whole system. Direct topologies provide a reduced hardware cost, but, as the number of dimensions is conditioned by 3D wiring restrictions, a high number of nodes per dimension is used, which increases communication latency and reduces network throughput. On the other hand, indirect topologies can provide better performance for large network sizes, but at the cost of a high number of switches and links. In this paper, we propose a new family of topologies that combines the best features of both direct and indirect topologies to efficiently connect an extremely high number of nodes. In particular, we propose an n–dimensional topology, where the nodes of each dimension are connected through a small indirect topology. This combination results in a family of topologies that provides high performance, with latency and throughput figures of merit close to indirect topologies, but with a lower hardware cost. In particular, it is able to double the throughput obtained per switching element of indirect topologies. Moreover, the layout of the topology is much simpler than in indirect topologies. Indeed, its fault–tolerance degree is equal or higher than the one for direct and indirect topologies.

3.1 Introduction

The topmost machines of the top 500 supercomputer list [1] are composed of hundreds of thousands of processing nodes. The performance of the machine is strongly impacted by the interconnection network among the processing nodes. The main design challenges of interconnection networks are to provide low latency and a high network throughput while providing a simple implementation at a reduced hardware cost. Two of the most important design issues of interconnection networks are topology and routing algorithm [2, 3]. Topology defines how the processing nodes are connected, and the routing algorithm determines the path followed by packets from their source to their destination node. The topology of a network also impacts to a large extent its cost. Topologies usually adopt a regular structure to simplify the routing algorithm, their implementation and the possibility of expanding the network. The most commonly-used taxonomy classifies topologies in direct and indirect networks[2, 3].

Direct topologies adopt an orthogonal structure where nodes are organized in an n–dimensional space. The nodes are connected in each dimension in a ring or array fashion. 2D or 3D direct

topologies are relatively easy to build as each topology dimension is mapped to a physical dimension. Direct topologies with more than three dimensions imply not only increasing its wiring complexity but also the length of its links when they are mapped to our 3D physical space, which leads to use a higher number of nodes per dimension, increasing the communication latency and negatively impacting performance.

The alternative is to use an indirect topology. The most common indirect topologies are multi-stage indirect networks (MINs) where switches are organized as a set of stages. Indirect topologies usually provide better performance for a large number of nodes than direct topologies. However, these better results are achieved at the cost of using a high number of switches and links, and increasing the wiring complexity, which grows with its size, unlike direct topologies, where complexity grows with the number of dimensions.

In this paper, we propose a new family of hybrid topologies that combines the best features of direct and indirect topologies. We propose an n–dimensional topology where the nodes of each dimension are connected by small indirect networks, thus reducing the communication latency in each dimension. The small size of this indirect topology allows a reasonable wiring complexity opposite to large indirect topologies. This combination results in a family of topologies that provides high performance, with latency and throughput figures of merit close to the ones of indirect topologies, but with a reduced hardware cost. In particular, it is able to double their throughput per switching element.

The rest of the paper is organized as follows. Sections 3.2 and 3.3 present background and related work, respectively. Section 3.4 and 3.5 describes the proposed family of topologies and the routing algorithms, respectively. Section 3.6 evaluates the new topologies. Finally, conclusions are drawn.

3.2 Direct and Indirect Topologies

In direct topologies, each node has its own router that connects it to its neighbor nodes by means of point-to-point links. Direct topologies usually adopt an orthogonal structure, where nodes are organized in an n–dimensional space, each node having at least one link in each dimension. The symmetry and regularity of these networks greatly simplify their implementation and the routing algorithm. In what follows, to distinguish the topology parameters of both direct and indirect topologies, we will refer to this topology as k_d–ary n_d–cubes, where k_d is the number

of nodes in each of the n_d dimensions. The total number of nodes in the system is given by $N = {k_d}^{n_d}$. The most commonly–used direct topologies are meshes, tori and hypercubes. In a mesh topology, all the nodes of a dimension compose a linear array, and, in torus, a ring. The hypercube is a particular case of a mesh where there are only two processing nodes in each dimension ($k_d = 2$), which forces the number of dimensions to be $n_d = log_2 N$. The torus has been used in several of the most powerful supercomputers (i.e. current numbers 1 and 3 in the top500 list, k Computer and Jaguar). For a given number of nodes N, direct topologies provide a richer connectivity as the number of dimensions increases, but at a higher cost, since more links and a higher router degree are required. Moreover, other issues have to be considered. Implementing a topology with more than three dimensions in our 3D physical space implies increasing the length of the links, the physical layout and the switch degree. Due to this reason, the topology design could lead to use a large number of nodes per dimension (k_d), increasing the diameter of the network and the probability of contention, thus negatively impacting network performance.

The alternative is to use indirect topologies where routers become independent devices known as switches and have not necessarily an associated processing node. The most common indirect networks are the multistage interconnection networks (MINs). In MINs, switches are organized in a set of stages. Processing nodes are connected to the first stage, and the other stages are connected among them using a certain connection pattern. Unidirectional MINs (UMINs) use unidirectional switches and links, so the network is traversed by packets only in one direction. In this case, processing nodes are also attached to the last stage and a unique path between each source–destination pair is provided (using the minimum number of stages). Bidirectional MINs (BMINs) use bidirectional links and switches. So, in order to travel from a source to a destination node, packets travel upwards the network and then downwards. BMINs provide several paths between each source–destination pair. The most widely–used MIN in commercial products ([7], [8] and [48]) is the fat–tree topology [4], which is a BMIN. Moreover, it has been used in some of the most powerful supercomputers (i.e. the number 2 in the Top500 list, the Tianhe 1A supercomputer). The k–ary n–tree is the most widely–used implementation of the fat–tree topology. In what follows, we will refer to this topology as k_i–ary n_i–tree. k_i is the number of links that connect a switch to the next or the previous stage, and n_i is the number of stages of the network. So, each switch has $2k_i$ bidirectional ports. A MIN to interconnect N processing nodes requires at least $log_{k_i} N$ stages, each of them with N/k_i switches, and $N = {k_i}^{n_i}$ processing nodes. For a MIN with N processing nodes, if we use high–degree switches, we will need fewer

stages and switches connected through a simple wiring, but each of them will be more complex. However, if we use low–degree switches, we will require more switches and stages with more complex wiring, but switches will be simpler. In fat–trees, the diameter of the network depends only on the number of stages, and it is given by $2n_i$, that is $2log_{k_i}N$. So, if n_i is increased, the distance that packets have to traverse is increased. Indirect networks usually provide better scalability than direct networks, because they provide smaller diameters and shorter latencies for large network sizes. Nevertheless, this is accomplished at a higher cost, because they require a high number of switches and links, and, unlike direct topologies, the complexity of network wiring grows with its size.

3.3 Related Work

Alternative topologies to the ones presented in the previous section have been proposed in the literature, but they have been never or seldom used in commercial products or in supercomputers. This is the case of the WK–recursive topology [9] proposed for off–chip networks and more recently for NoCs [10], which has difficulties to guarantee deadlock–freedom. Another very popular topology in recent papers is the flattened–butterfly[11], which is a modification of the MIN butterfly, resulting in an n–dimensional direct network (defined with k_f and n_f parameters) where the nodes of each dimension are fully connected, similar to a generalized hypercube but attaching several nodes (parameter p_f) to the same switch. This topology, like the generalized hypercube, has a high cost, specially for large machines, which is the focus of our proposal.

Other works propose the combination of several topologies, like we do in this paper. However, most of them have been proposed for on–chip networks and therefore the target is different to ours. In [17], each core is connected to two different tree networks. This proposal is not suitable for large machines due to the complexity of the cable layout and the poor performance achieved even with two trees. A topology closer to the new family proposed in this paper is the mesh of trees (MoT) introduced in [18] and later used also in NoCs [19]. It is based on an $n-$dimensional topology where the nodes of a given dimension are connected using a tree. This results in a variant of our family with very poor expected performance due to using a simple tree for connecting the nodes of a dimension.

Many of the proposals for NoCs are based on hierarchical topologies. Subsets of cores are connected by small local networks connected by a global network. This is not the case of

the family proposed in this paper, since processing nodes are connected to both topologies. Hierarchical designs are expected to have higher latencies and smaller throughput, since both networks must be traversed for most of the source–destination pairs. In [12], the authors propose to use as local network a simple bus and a mesh as global network. However, this is not suitable for large machines due to the low performance provided by buses and meshes. The authors of [13] propose a tool to select the most suitable topology for a given design, connecting Clos networks by a global torus. Another hierarchical topology is the DragonFly[14], which is based on grouping routers to increase the effective radix of the network. Global channels, which link different groups, are long and a high latency can be expected.

Finally, in [16], a topology that combines several tori networks is proposed. The proposal focus on large supercomputers, but its applicability is limited to 2^{16} nodes and its cable layout is complex for large machines. Bypass links in the diagonal direction reduce the number of hops when traversing several dimensions. Moreover, this topology has problems to guarantee deadlock-free routing.

3.4 The New Family of Hybrid Topologies

This paper proposes a new family of topologies able to efficiently connect an extremely high number of processing nodes. We propose an hybrid topology based on combining an n–dimensional direct network with smaller indirect topologies that connect the nodes of each dimension. Therefore, we combine the advantages of direct and indirect topologies to obtain a family of topologies that is able to connect a high number of processing nodes while providing low latency, high throughput and a high level of fault–tolerance, at a lower cost than indirect networks. In particular, we propose using two different kinds of switches. First, low–degree switches are used to move packets across dimensions. We will refer to these switches as routers, and they will have at least one processing node attached to them and as many ports as dimensions. These routers could be part of the interconnection network but another option is to take advantage of network interface board with routing capabilities like the ATOLL board [49], in such a way that processing nodes, in addition to inject and receive packets into and from the network, would also be able to receive packets that are not destined to them, routing these packets towards their destination nodes. Additionally to these routers, the proposed topology also uses other switches to implement the indirect networks that interconnect the nodes of each dimension.

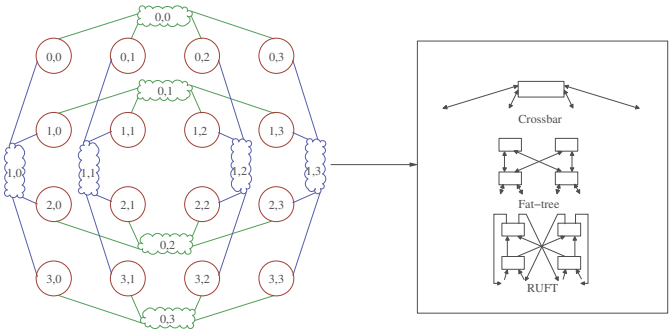


FIGURE 3.1: An example of the new topology with $n_n = 2$, $k_n = 4$ and $d_n = 4$.

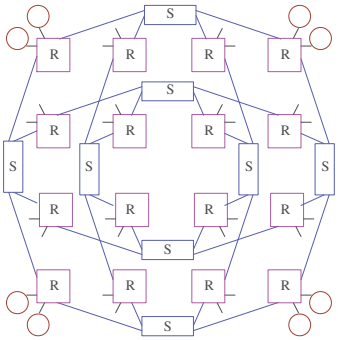


FIGURE 3.2: An example of the new topology with $n_n = 2$, $k_n = 4$, $d_n = 4$ and $p_n = 2$.

3.4.1 Description of the family

The newly proposed topology organizes nodes in n dimensions, like in a direct topology, but the nodes of a given dimension are not exclusively connected with their adjacent nodes, as in meshes and tori. Instead, all the nodes of a given dimension are directly connected by means of an indirect network. This indirect network could be even a simple switch, if the number of nodes per dimension is small. If the number of nodes per dimension exceeds the number of ports of the available switches, an indirect network (i.e. a MIN) is required. Therefore, the proposed family of topologies is defined by three parameters. Two of them are inherited from direct networks: the number of dimensions n_n and the number of nodes per dimension k_n. The number of processing nodes it can interconnect is given by $N = k_n^{n_n}$. In addition, there is an extra parameter, m_n, the number of stages of the indirect subnet. The number of stages (m_n) necessary to interconnect the k_n nodes of a given dimension depends on the number of ports of the switches of the indirect subnet (which will be referred to as d_n) and on k_n. If $k_n \leq d_n$, a simple switch is enough to interconnect all the nodes of the dimension, and m_n will be equal to 1. On the other hand, if $k_n > d_n$, a MIN with a number of stages given by $log_{d_n} k_n$ is required. We will refer to the new family of topologies proposed in this paper as k_n–ary n_n–direct m_n–indirect topology, where k_n is the number of nodes per dimension, n_n is the number of dimensions and m_n is the number of stages of the indirect subnets.

In this paper, we have considered two different MINs to connect the nodes of a given dimension. The first one is the well-known fat–tree topology. The second one is RUFT [5], a unidirectional MIN derived from a fat–tree using a load–balanced deterministic routing algorithm [22].

Topology	Parameters	
k_n–ary n_n–direct m_n–indirect	n_n	# of dimensions
	k_n	# of nodes per dimension
	m_n	# of stages of the indirect subnet
	d_n	switch degree
	p_n	# nodes per router

TABLE 3.1: Parameters of the topologies analyzed in this paper.

Figure 3.1 shows an example of the new topology with 2 dimensions ($n_n = 2$) and 4 nodes per dimension ($k_n = 4$), with a total of 16 processing nodes. In this case, as the number of ports $d_n = 4$ of the switches equals k_n, a crossbar is used in order to interconnect the 4 nodes. Therefore, the network is a 4–ary 2–direct 1–indirect. However, for a higher number of nodes per dimension, say $k_n = 8$ and the same switch size, a MIN should be used, such as a fat–tree or a RUFT with 3 stages and 12 4-port switches (bidirectional for the fat–tree and unidirectional for the RUFT), implementing in this case an 8–ary 2–direct 3–indirect.

Notice that it is also possible to attach several processing nodes to the same router (known as concentration) as shown in Figure 3.2, in which nodes and routers are shown separately. In this case, two processing nodes are connected to each router. Only the corner nodes are drawn for the sake of clarity. This extension introduces a new parameter on the topology family, p_n, the number of processing elements that are connected to each router. In this case, the number of processing nodes is given by $N = p_n k_n^{n_n}$. In Figure 3.2, two different switching elements with different sizes can be distinguished. The switches (labelled with an "S") are only connected to other switching elements and have a degree of k_n, whereas the routers (labelled with an "R") are connected on one side with processing nodes and on the other side with switches, being their degree $n_n + p_n$.

As it can be seen in Figure 3.2, if a crossbar is used as indirect subnet, switches allow packets to change its position in a given dimension by traversing only two links, whereas routers allow to change the dimension. Thus, the diameter of the new topology is $2n_n$. If a MIN is used instead of a crossbar, the diameter of the network would be the diameter of the used MIN multiplied by n_n.

Table 3.1 summarizes the parameters that define the hybrid family of topologies proposed in this paper. Notice that only the d_n or m_n parameter is needed as the other one can be easily derived from it and k_n. The proposed hybrid topologies can take advantage of current high–radix switches. Switches up to 64 ports are available [50], while recent works propose switches

with up to 144 ports [51]. With such switches, a small MIN with only 2 or 3 stages or even a single switch will be enough to connect the nodes in each dimension in most cases. As the MIN is very small, it will introduce low latency and low wiring complexity.

Hybrid topologies have several main advantages. First, they allow to reduce the diameter compared to direct topologies. If we are using crossbars, the diameter of the new topology is $2n_n$, while in a mesh it is $n_d(k_d - 1)$. This will lead to improve network performance, increasing the network throughput and decreasing the latency. On the other hand, the number of switches and links is reduced compared to an indirect topology that connect the same number of nodes (see Section 3.6), therefore reducing the cost of the network. Finally, as the number of alternative paths in the proposed topology is very high, it provides a higher level of fault–tolerance than direct topologies and the same as indirect ones (see Section 3.6).

3.5 Routing Algorithms for the new Family of Topologies

In this section, we first describe the routing algorithms proposed for k_n–ary n_n–direct 1–indirects (i.e., a crossbar is used as indirect subnet), and then the ones proposed for the general case, that is, for k_n–ary n_n–direct m_n–indirects, using a fat–tree or a RUFT as indirect subnets. In both cases, we provide deterministic and adaptive routing algorithms.

We will first explain how routers and switches are labeled in the new topology family. Each router is labeled like in meshes and tori with a set of components (as many as network dimensions) $\langle r_{n_n-1}, r_{n_n-2}, ..., r_1, r_0 \rangle$. Each coordinate represents the position of each router in each dimension. On the other hand, the switches are labeled by a 2-tuple *[d,p]*, where *d* is the dimension in which the switch is located, and *p* is the position of that switch in that dimension (see Figure 3.1). Notice that routers do not belong to any dimension, since they are connected to all of them, and packets change dimension through them. On the contrary, switches allow packets to move across a given dimension.

3.5.1 Routing in k_n–ary n_n–direct 1–indirect topologies

3.5.1.1 Deterministic routing

The deterministic routing algorithm for k_n–ary n_n–direct 1–indirects, which will be referred to as Hybrid–DOR, is adapted from the DOR deterministic routing algorithm for meshes. In DOR, packets are routed through the dimensions following an established order until the destination node is reached. At each dimension, packets traverse several routers until the movement in that dimension is exhausted. On the other hand, as each router has two links per dimension, packets must be forwarded in each dimension through the direction that guarantees the minimal path. In Hybrid–DOR, network dimensions are also crossed in an established order to guarantee deadlock freedom. However, packets do not perform several hops at each dimension. Instead, in Hybrid–DOR, routers directly forwards packets through the unique link of the dimension they have to traverse. Notice also that, in k_n–ary n_n–direct 1–indirect topologies, it is not required to choose the direction at each dimension, as there is only one link per dimension. Regarding how the crossbars route packets, they just forward packets through the link indicated by the destination component of the corresponding dimension, requiring just one hop to reach next router.

3.5.1.2 Adaptive routing

As done in meshes, adaptive routing for k_n–ary n_n–direct 1–indirect topologies is accomplished by allowing to cross the network dimensions in any order. Therefore, the routers, instead of selecting the next dimension to forward the packet, check which dimensions approach it to its destination and later, the selection function selects one of them following some criterion. The routing algorithm for the crossbars is the same as the one used in deterministic routing.

As in meshes, when using adaptive routing in k_n–ary n_n–direct 1–indirect topologies, at least two virtual channels (VCs) are required in the routers to avoid deadlocks[3]. One VC is used for adaptive routing and the other one to provide a deadlock-free escape channel (for traversing dimensions in order). In addition, two VCs are also required in the crossbar links to maintain the routing restrictions when packets are forwarded to the routers after traversing the crossbar.

3.5.2 Routing in k_n–ary n_n–direct m_n–indirect topologies

In this case, crossbars are replaced by small MINs. As stated above, we will consider fat–trees or RUFTs as MINs. In order to properly identify the switches inside a given fat–tree or RUFT, we extend the classical switch coordinates used in MINs (stage and position in that stage) to include the coordinates of the MIN in the direct topology. In this way, the switch coordinates in k_n–ary n_n–direct m_n–indirect topologies are given by a 4-tuple *[d,p,e,o]*, where d is the dimension the MIN belongs to, p is the position of the MIN in that dimension, e is the stage of the switch inside the MIN, and o is the order of that switch in that stage. Remember that d and p are the coordinates of the corresponding crossbar in a k_n–ary n_n–direct 1–indirect.

Since the routers are the same regardless of the indirect topology used, its routing algorithm is the same as the one explained for k_n–ary n_n–direct 1–indirect topologies, both for deterministic and adaptive routing. However, switch routing algorithm depends on the indirect network used.

First, we focus on the k_n–ary n_n–direct m_n–indirect topology that uses fat–trees as indirect subnets. Despite a fat–tree has several paths for each source–destination pair (i.e., it allows adaptive routing), we propose to use the deterministic routing algorithm presented in [22] since it is simple and outperforms adaptive routing for most traffic patterns [22]. Routing is composed of two subpaths. First, the packet is sent forward to the root of the fat–tree. Once the packet has finished this subpath, it is sent downwards to its destination. The link to be used in both subpaths is given by the destination coordinate corresponding to the stage where the switch is located at. Please see [22] for details. In the fat–tree subnets of the k_n–ary n_n–direct m_n–indirect topologies, the routing algorithm is the same, but we only use the part of the destination identifier corresponding to the dimension the fat-tree belongs to, instead of using the whole destination identifier. In this way, the packet is delivered to the same router that would be reached through the corresponding crossbar in a k_n–ary n_n–direct 1–indirect topology. In RUFT, there is a unique path between each source–destination pair and packets have to cross all the stages, reaching the last stage which is directly connected back to the processing nodes. The link to be used in a particular switch is given by the destination component corresponding to the switch stage. Please see [5] for details.

3.6 Evaluation

3.6.1 Network Model

We will evaluate the new family of topologies by simulation. Virtual cut–through switching with input-output queued switches and credit-based flow control was assumed. We also assumed that it takes 20 cycles to route a packet; switch and link bandwidth equal to one flit per cycle; and a fly time of 8 cycles. These values were used to model Myrinet networks in [52]. In addition, for the new topology using RUFT as indirect subnet, the fly time of the last stage long links was assumed to be 8 cycles per stage.

Several synthetic traffic patterns were used in the evaluation, including uniform, hot–spot, and complement. However, for the sake of brevity, we will show results only for the uniform traffic pattern, where message destinations are randomly chosen among all destinations. Several packet sizes were analyzed (from 16 to 512 flits). Due to space limitations, we will only show results for 256-flit packets.

3.6.2 Evaluation Results

We compare the k_n–ary n_n–direct m_n–indirect family using different indirect subnets (crossbar, fat–tree and RUFT) against other well–known topologies such as tori, meshes, fat–trees, and flattened–butterflies. Due to space limitations, we show here only a subset of the most representative simulations. In particular, we only show results for deterministic routing and uniform traffic, nevertheless we can state from obtained simulation results that adaptive routing and other traffic patterns lead to similar relative results to the ones presented in this paper. For the proposed family of topologies, the algorithms are the ones presented in Section 3.5.1.1. For tori, meshes and flattened–butterflies, DOR routing [3] is used. For fat–trees, we use the deterministic routing algorithm proposed in [22]. If not stated the contrary, only one processor is attached to each router. If several are attached, p–c notation is used, p being the number of processors attached to each node. These networks are compared with fat–trees with the same number of nodes (in some cases, several configurations are possible).

Figure 3.3.(a) shows the results for small networks (16 nodes) with uniform traffic. The flattened–butterfly is the one that achieves the lowest throughput (we have selected a flattened–butterfly

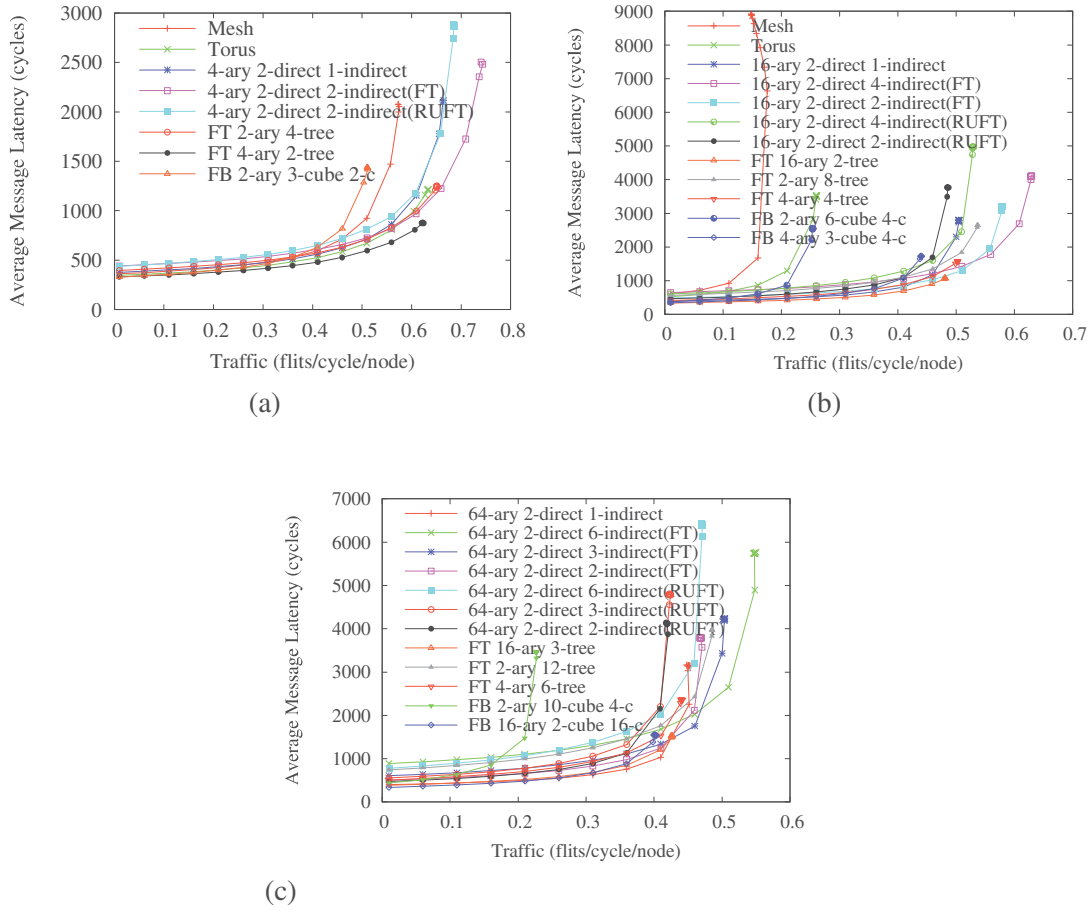


FIGURE 3.3: Network latency vs. accepted traffic with uniform traffic and deterministic routing. 2D direct topologies. (a) 16, (b) 256 and (c) 4096 nodes.

with similar hardware cost). The mesh is the second worst, and the torus and the different configurations of our family and fat–trees obtain similar throughputs.

As we increase the number of nodes per dimension in the direct topologies (Figures 3.3.(b) and 3.3.(c)), throughput goes down in all topologies. In the case of mesh and torus topologies, throughput is strongly reduced as the average distance between two nodes is markedly higher than in the other topologies. In fact, Figure 3.3.(c) does not show mesh and torus topologies because their throughput is very low (it is an 88,66% and 83,66% lower than k_n–ary n_n–direct 1–indirect, respectively). The three best topologies are the ones of our family that use a crossbar or a fat–tree to connect the different nodes of a dimension. The other topologies obtain results halfway these topologies and the direct ones. Notice that each topology has a different hardware cost, which is evaluated in following sections. In Figures 3.3.(b) and 3.3.(c) we can also see that, if we decrease the number of stages in the MINs of the k_n–ary n_n–direct m_n–indirects, the arity of the switches is increased, and a lower latency should be obtained. The plots show

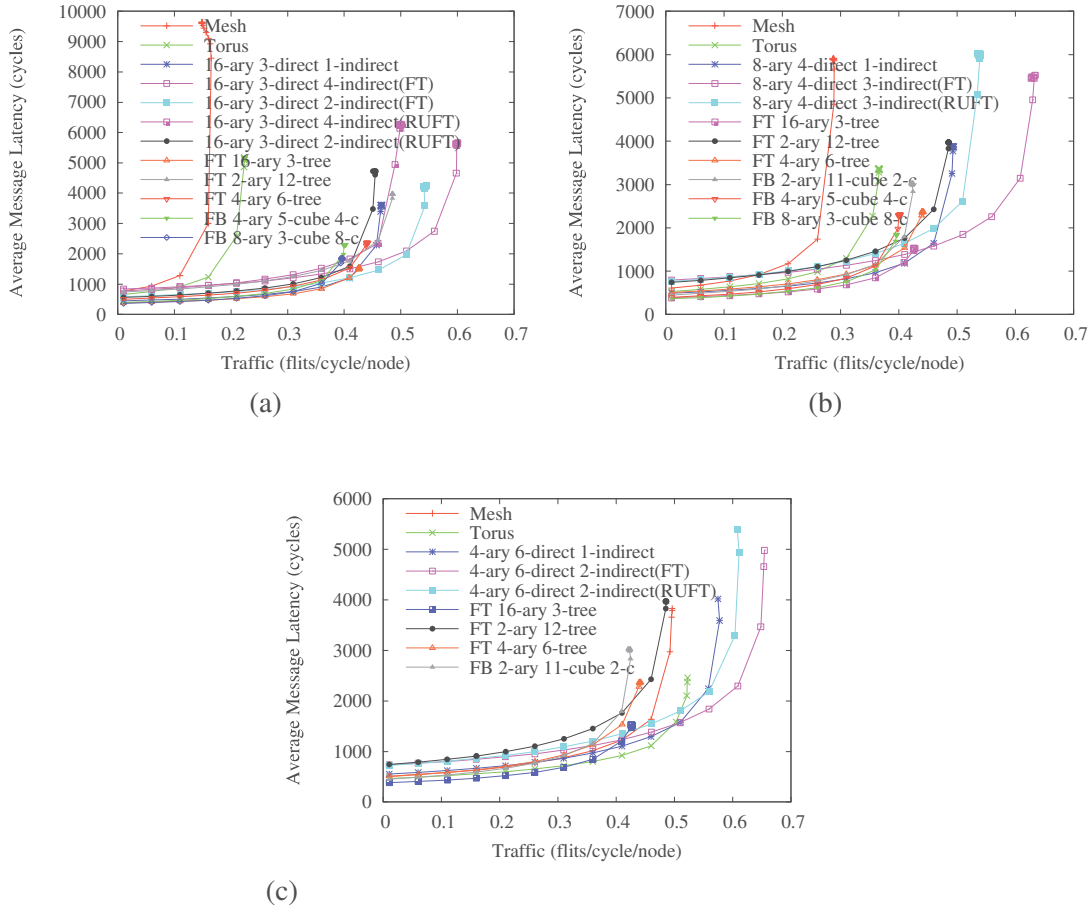


FIGURE 3.4: Network latency vs. accepted traffic with uniform traffic and deterministic routing. 4K-node topologies. (a) 3D, (b) 4D and (c) 6D.

that, the higher the number of stages, the higher the base latency as more switches have to be crossed by packets. Networks with more stages also achieve more throughput. This increase in the network throughput is explained by the HoL blocking effect. With fewer stages, each switch concentrates traffic to a higher number of destinations, leading to more HoL blocking effect and, hence, less throughput. Figures show that in a k_n–ary n_n–direct 1–indirect, base latency is stable because the number of hops between nodes does not depend on the number of nodes per dimension. However, in k_n–ary n_n–direct m_n–indirect, on fat–trees, the base latency increases with the number of nodes because the number of stages in the indirect subnets grows. In the case of tori and meshes, base latency strongly depends on the number of nodes per dimension, as the average distance is highly increased.

Figure 3.4 analyzes the impact of the number of dimensions in the different topologies. We analyze a 4K-node network implemented with different number of dimensions. We can distinguish three different behaviors. Mesh and torus topologies have a similar behavior. The higher

the number of dimensions and the fewer the number of nodes per dimension, the higher the achieved throughput. Also, base latency decreases with the number of dimensions, because the average distance between nodes is reduced. However, the behavior of k_n–ary n_n–direct 1–indirect is different. Throughput increases with the number of dimensions because the size of switches of the indirect network (a crossbar in this case) is reduced, and, hence, the pernicious effect of HoL blocking is reduced. However, the base latency does not improve with the number of network dimensions. This is due to the fact that the number of hops that packets must perform grows with the number of dimensions. Concerning the k_n–ary n_n–direct m_n–indirect, they have a similar behavior to the previous one, but with a difference. The base latency decreases when the number of dimensions increases because there are fewer nodes per dimension. Therefore, the indirect subnets have fewer stages. The selected configurations of the flattened–butterfly (the ones with a similar hardware cost to our proposal) provide an intermediate throughput. Finally, notice that, as the number of dimensions is increased, the direct topologies become more competitive compared to fat–trees. Anyway, we would like to remark that the new family of topologies always obtains the best throughput regardless of the number of dimensions.

We have analyzed the impact of packet size on the different topologies for all the traffic patterns and we found that their behavior is very similar regardless the packet size.

3.6.3 Cost–performance analysis

This section estimates and compares the hardware cost of each topology evaluated in the paper. In particular, we will analyze, for each topology the number of links, the number of switches, and the number of switching elements. In particular, we will estimate the hardware cost by using the total number of switching elements since it considers both the total number of required switches and its degree, which is related to the number of links. The number of links is not a good metric since a network with more complex switches has a lower number of links but seems to be cheaper using this metric. As known, the number of switching elements of a switch of degree d is $d \times d$. A switching element is the basic component that implements each input of a multiplexer. Moreover, we will introduce two different figures of merit that tries to combine both performance and cost of a given topology. The first one is the throughput per switching element. The higher this parameter, better performance/cost ratio the topology has. The other parameter is the product of the base latency and the number of switching elements. The higher, the worse the topology is.

	Mesh	Torus	Fat–tree	Flattened–Butterfly
Switches	N	N	Nn_i/k_i	$k_f^{n_f}$
Switching elements	$N(2n_d+1)^2$	$N(2n_d+1)^2$	$4k_in_iN$	$k_f^{n_f}(n_f(k_f-1)+N/k_f^{n_f})^2$
links	$2(k_d-1)k_d^{n_d-1}n_d$	$2k_d^{n_d}n_d$	$2N(n_i-1)$	$k_f^{n_f}(k_f-1)n_f$
	k_n–ary n_n–direct m_n–indirect FT		k_n–ary n_n–direct m_n–indirect Xbar and RUFT	
Switches	$N/p_n((m_nn_n/k_i)+1)$		$N/p_n((m_nn_n/k_i)+1)$	
Switching elements	$N/p_n((4m_nn_nk_i+(n_n+p_n)^2)$		$N/p_n(m_nn_nk_i+(n_n+p_n)^2)$	
links	$2(k_n^{n_n}n_n+(m_n-1)Nn_n/p_n)$		$2k_n^{n_n}n_n+(m_n-1)Nn_n/p_n$	

TABLE 3.2: Analytical comparisons of the Mesh, Torus, Fat–Tree, flattened–butterfly, and the k_n–ary n_n–direct m_n–indirect topologies. k_i in k_n–ary n_n–direct m_n–indirect topologies refers to the arity of indirect switches.

Table 3.2 shows the formula to compute the number of unidirectional links, switches and switching elements for each topology. Notice that, as RUFT is an unidirectional indirect network, the number of switching elements is four times smaller than the one in the fat–tree subnets.

Table 3.3 shows in the upper part results for 16–node topologies for uniform traffic and deterministic routing. The network is 2D in the case of k_n–ary n_n–direct m_n–indirect, mesh, and torus topologies. For flattened–butterflies we have considered 2D and 3D configurations that match the hardware cost of our proposal. The 2D flattened–butterfly is the topology that achieve the highest raw throughput, followed by the k_n–ary n_n–direct m_n–indirect using fat–trees as indirect subnets, but with a larger number of switching elements. The next ones are k_n–ary n_n–direct m_n–indirects with RUFT and crossbar.

Let us compare the cost of the topologies. The 3D flattened–butterfly is the one with the smallest number of switching elements. The following cheaper topologies, with the same number of switching elements, are the k_n–ary n_n–direct 1–indirect and the k_n–ary n_n–direct m_n–indirect using RUFT. Although the latter has more switches than the former, as they are simpler, the total number of switching elements is the same. On the contrary, the 2D flattened–butterfly and the k_n–ary n_n–direct m_n–indirect using fat–trees, in this order, are the topologies with the highest cost. However, when throughput and cost (i.e., the throughput per switching element) of each topology are considered, the conclusions completely differ. In that case, the best topology is the 3D flattened–butterfly, followed very near by the k_n–ary n_n–direct m_n–indirect with RUFT and the k_n–ary n_n–direct 1–indirect. On the other hand, fat-trees are less interesting due to their higher complexity. Concerning the number of stages of the indirect subnet, we can see that a lower number of stages allows the best result from the latency–cost point of view. However, if throughput is the primary concern, a MIN with more stages is a better option as it reduces the HoL blocking effect.

#Nodes	Topology	Base lat.	Throughput	Links	Switches	Switching elem.	Thro./switching elem.	Base latency* switching elem.
16	Flattened–Butterfly 2-ary 3-cube 2-c	329,00	0,51133	24	8	200	0,0025567	65800
16	4–ary 2–direct 2–indirect(RUFT)	437,01	0,69003	96	48	272	0,0025369	118866
16	4–ary 2–direct 1–indirect	375,48	0,66259	64	24	272	0,0024360	102131
16	Torus	345,30	0,63534	64	16	400	0,0015883	138120
16	Mesh	360,38	0,57466	48	16	400	0,0014367	144150
16	Fat-Tree arity 2 stages 4	415,57	0,65030	96	32	512	0,0012701	212773
16	Fat-Tree arity 4 stages 2	327,46	0,62456	32	8	512	0,0012198	167661
16	4–ary 2–direct 2–indirect(FT)	439,24	0,74223	128	48	656	0,0011314	288139
16	Flattened–Butterfly 4-ary 2-cube 1-c	328.47	0,81492	96	16	784	0,0010394	257524
4096	64–ary 2–direct 6–indirect(RUFT)	770,56	0,47438	57344	28672	135168	0,0000035	104155095
4096	64–ary 2–direct 3–indirect(RUFT)	546,71	0,42546	32768	10240	135168	0,0000031	73898072
4096	64–ary 2–direct 2–indirect(RUFT)	463,11	0,41971	24576	6144	167936	0,0000025	77772700
4096	64–ary 2–direct 6–indirect(FT)	866,40	0,55148	98304	28672	430080	0,0000013	372620624
4096	64–ary 2–direct 3–indirect(FT)	591,82	0,50569	49152	10240	430080	0,0000012	254531077
4096	Fat-Tree arity 2 stages 12	865,20	0,48493	90112	24576	393216	0,0000012	340210585
4096	Fat-Tree arity 4 stages 6	561,23	0,43872	40960	6144	393216	0,0000011	220683070
4096	Flattened–Butterfly 2-ary 10-cube 4-c	439,94	0,22688	10240	1024	200704	0,0000011	88297728
4096	64–ary 2–direct 2–indirect(FT)	499,96	0,46912	32768	6144	561152	0,0000008	280553722
4096	64–ary 2–direct 1–indirect	389,29	0,44818	16384	4224	561152	0,0000008	218448488
4096	Torus	1292,88	0,07323	16384	4096	102400	0,0000007	132390963
4096	Flattened–Butterfly 16-ary 2-cube 16-c	338,47	0,40332	7680	256	541696	0,0000007	183349068
4096	Fat-Tree arity 16 stages 3	388,45	0,41601	16384	768	786432	0,0000005	305487961
4096	Mesh	1606,12	0,05084	16128	4096	102400	0,0000005	164467087
4096	Flattened–Butterfly 64-ary 2-cube 1-c	339,65	0,60032	516096	4096	66064384	0,0000000	22435873084

TABLE 3.3: Results for different 2–D topologies with uniform traffic and deterministic routing.

Table 3.3 shows also results for larger networks. As can be seen, as the number of nodes per dimension increases, the network performance gets worse in mesh and torus topologies. Regarding the hybrid topologies proposed in this paper, the results shown confirm what we stated above. The best raw performance is obtained with the 2D flattened–butterfly and configurations of the k_n–ary n_n–direct m_n–indirect (fat–tree subnets) topology, with an advantage for the topologies proposed in this paper as we increase the number of nodes per dimension. From the cost–performance point of view, different configurations of the new family with RUFT indirect subnets are the best option. Although the k_n–ary n_n–direct m_n–indirect (RUFTs) topology does not achieve the best raw performance, it is able to take the highest advantage of the combination

#Dimensions	Topology	Base lat.	Throughput	Links	Switches	Switching elem.	Thro./switching elem.	Base latency* switching elem.
3	16–ary 3–direct 4–indirect(RUFT)	766,14	0,50088	61440	28672	163840	0,0000031	125524540
3	16–ary 3–direct 2–indirect(RUFT)	544,01	0,45590	36864	10240	163840	0,0000028	89130980
3	16–ary 3–direct 1–indirect	443,32	0,46498	24576	4864	262144	0,0000018	116213728
3	16–ary 3–direct 4–indirect(FT)	815,39	0,60257	98304	28672	458752	0,0000013	374060903
3	Fat-Tree arity 2 stages 12	865,20	0,48493	90112	24576	393216	0,0000012	340210585
3	16–ary 3–direct 2–indirect(FT)	576,95	0,50412	49152	10240	458752	0,0000011	264674815
3	Torus	654,40	0,22376	24576	4096	200704	0,0000011	131340348
3	Fat-Tree arity 4 stages 6	561,23	0,43872	40960	6144	393216	0,0000011	220683070
3	Flattened–Butterfly 4-ary 5-cube 4-c	396,56	0,40179	15360	1024	369664	0,0000011	146595250
3	Flattened–Butterfly 8-ary 3-cube 8-c	361,823	0,39680	10752	512	430592	0,0000009	155798223
3	Mesh	762,46	0,16553	23040	4096	200704	0,0000008	153029131
3	Fat-Tree arity 16 stages 3	388,45	0,41601	16384	768	78432	0,0000005	305487961
3	Flattened–Butterfly 16-ary 3-cube 1-c	365,00	0,63116	184320	4096	8667136	0,0000001	3163526160
6	4–ary 6–direct 2–indirect(RUFT)	714,24	0,61761	73728	28672	299008	0,0000021	213564147
6	4–ary 6–direct 1–indirect	546,78	0,57877	49152	10240	299008	0,0000019	163492186
6	Fat-Tree arity 2 stages 12	865,20	0,48493	90112	24576	393216	0,0000012	340210585
6	Flattened–Butterfly 2-ary 11-cube 2-c	450,03	0,42385	22528	2048	346112	0,0000012	155761542
6	4–ary 6–direct 2–indirect(FT)	715,60	0,65648	98304	28672	593920	0,0000011	425011201
6	Fat-Tree arity 4 stages 6	561,23	0,43872	40960	6144	393216	0,0000011	220683070
6	Torus	463,94	0,52038	49152	4096	692224	0,0000008	321150900
6	Mesh	502,79	0,49203	36864	4096	692224	0,0000007	348041789
6	Fat-Tree arity 16 stages 3	388,45	0,41601	16384	768	78432	0,0000005	305487961
6	Flattened–Butterfly 4-ary 6-cube 1-c	416,37	0,66666	73728	4096	1478656	0,0000005	615671623

TABLE 3.4: Results for 4096–nodes topologies with uniform traffic and deterministic routing.

of an unidirectional MIN with an optimized routing algorithm, leading to obtain by far the best throughput per switching element.

Table 3.4 shows the results for the large networks (4096 nodes) but now considering a higher number of dimensions (3 and 6). As can be seen, when the number of dimensions is increased, improvements are achieved, specially in torus and mesh topologies. However, the best raw throughput is always obtained by a flattened–butterfly, followed by the k_n–ary n_n–direct m_n–indirect (fat–trees) configuration, although at a higher cost in the case of the flattened–butterflies. The best cost–performance ratio is achieved by the k_n–ary n_n–direct m_n–indirect (RUFT). In particular, the throughput per switching elements of this topology is much better than that obtained by fat–tree and flattened–butterfly.

The previous tables also show the topology cost in links. Some of the configurations of the flattened–butterflies, fat–trees and the k_n–ary n_n–direct m_n–indirects (fat–trees) have the highest cost in links. However, the fat–trees and the k_n–ary n_n–direct m_n–indirects (fat–tree subnets) that have a lower number of links are the ones with fewer stages in the fat–tree. These configurations are also the ones that achieve the best latency–complexity figure of merit. Concerning the number of dimensions, if we increase this number, the number of links is increased, because adding more dimensions provides more connectivity. k_n–ary n_n–direct m_n–indirect (fat-trees) topology is an exception because, although we have more fat–trees by increasing the number of dimensions, they are smaller and have fewer links. This makes k_n–ary n_n–direct m_n–indirects (fat–tree subnets) to be less impacted by an increase in the number of dimensions. There is also a significant difference in the number of links between k_n–ary n_n–direct m_n–indirect using fat–trees or RUFTs subnets. This is because in RUFT, the links are unidirectional, whereas in a fat–tree they are bidirectional.

3.6.4 Fault–tolerance

The proposed topologies provide a lot of alternative paths, which is very important to avoid faults.

In the mesh topology, the worst case arises when a link connected to a corner node fails. As each corner node has a number of links equal to the number of network dimensions, to keep the network connected, the maximum number of faults it can tolerate is equal to the number of dimensions minus 1. The torus topology tolerates more faults than the mesh, twice the number of dimensions minus 1, due to the fact that packets can be moved in both directions of a dimension. The fat–tree topology tolerates as many faults as k of the switches minus one. The flattened–butterfly tolerates as many faults as $k_f(n_f-1)$, which is the switch degree. In the case of k_n–ary n_n–direct m_n–indirect topologies, the number of tolerated faults is the number of dimensions minus 1. Notice, though, that routing should be also changed to fully support fault tolerance in all topologies, but this is out of the scope of this paper.

To perform a simple topology comparison, assume we have switches of degree d. In a mesh network, the number of dimensions would be $d/2$, so the number of tolerated faults is $d/2-1$. A torus would have also $d/2$ dimensions, with a fault–tolerance degree of $d-1$ faults. In a fat–tree, switches would have $d/2$ up and $d/2$ down ports, thus tolerating $d/2-1$ faults. In the flattened–butterfly topology the number of tolerated faults is equal to $d-1$. For the new

topology, a network of d dimensions can be built, so it is able to tolerate up to $d-1$ faults, which is the best level of fault–tolerance achieved by the evaluated topologies.

We have not considered faults in injection links, but fat–trees and flattened–butterflies usually have a single link that connects the node to the network. If this link fails, the node will be isolated. Therefore, considering also the injection links, these networks will not tolerate any fault. On the contrary, in the new topology family, if the router is implemented inside the NIC, it is connected to the network through as many links as dimensions in the network, so the injection links also tolerate $d-1$ faults.

3.7 Conclusions

This paper presents a new family of hybrid topologies for large–scale interconnection networks. It uses an n–dimensional topology where the nodes of each dimension are connected through a small indirect topology. This indirect topology may be a crossbar or a MIN (a fat–tree or a RUFT). This combination results in a set of topologies that provide high–performance, with latency and throughput figures of merit close to one obtained with indirect topologies, but with a much lower hardware cost. In particular, the new topology with fat–trees as indirect subnet is the topology that achieve the highest throughput. Nevertheless, from the cost–performance point of view, the new topologies with RUFT indirect subnets are the winners, as they are able to double the throughput obtained per switching element compared to indirect topologies, and this advantage is even higher for direct topologies. Moreover, the layout of the topology is much simpler than the one for indirect topologies. Concerning fault–tolerance, for a given switch size, the new topologies are able to achieve equal or even higher levels of fault–tolerance than the ones obtained with other topologies.

Chapter 4

The k-ary n-direct s-indirect Family of Topologies for Large-Scale Interconnection Networks

Authors: Roberto Peñaranda (Universidad Politécnica de Valencia), Crispín Gómez (Universidad de Castilla La Mancha), María Engracia Gómez, Pedro López, Jose Duato (Universidad Politécnica de Valencia).

Type: Journal.

Journal: The Journal of Supercomputing.

Publisher: Springer.

Year: 2016.

DOI: $http://dx.doi.org/10.1007/s11227-016-1640-z$

URL: $http://link.springer.com/article/10.1007\%2Fs11227-016-1640-z$

ISSN: 0920-8542.

Category: Processor Architectures and Computer Science.

Impact Factor: 0.858

JRC ranking: Q2 (2014)

Abstract

In large-scale supercomputers, the interconnection network plays a key role in system performance. Network topology highly defines the performance and cost of the interconnection network. Direct topologies are sometimes used due to its reduced hardware cost, but the number of network dimensions is limited by the physical 3-D space, which leads to an increase of the communication latency and a reduction of network throughput for large machines. Indirect topologies can provide better performance for large machines, but at higher hardware cost. In this paper, we propose a new family of hybrid topologies, the k–ary n–direct s–indirect, that combines the best features from both direct and indirect topologies to efficiently connect an extremely high number of processing nodes. The proposed network is an n–dimensional topology where the k nodes of each dimension are connected through a small indirect topology of s stages. This combination results in a family of topologies that provides high performance, with latency and throughput figures of merit close to indirect topologies, but at a lower hardware cost. In particular, it doubles the throughput obtained per cost unit compared with indirect topologies in most of the cases. Moreover, their fault-tolerance degree is similar to the one achieved by direct topologies built with switches with the same number of ports.

4.1 Introduction

The size of large supercomputers has been growing year after year. The topmost machines of the top 500 supercomputer list [1] are being built up by using hundreds of thousands of processing nodes. All these processing nodes work jointly to solve a given problem in as less time as possible. This joint work is performed thanks to the interconnection network that allows all the processing nodes to share data among them. The interconnection network must enable an efficient communication among all the processing nodes because it strongly impacts the performance of the whole system. Transmission time of data across the interconnection network adds up to the processing time, impacting the time required to run the applications.

The main design challenges of interconnection networks design are to provide low latency communications, in order to reduce the execution time of applications, and a high network throughput, to allow as many simultaneous communications as needed, while providing a simple implementation at a reduced hardware cost.

In addition to performance and cost, another important feature of interconnection networks is their ability to provide fault–tolerance. The high amount of hardware that can be found in an interconnection network in high–performance machines significantly impacts the probability of having a fault in the system. Each component may independently fail, and therefore, the probability of having a single fault in the whole system drastically raises with the number of elements that compose it. Thus, tolerating faults is also a basic requirement when designing an interconnection network, specially for a large machine.

Two of the most important design issues of the interconnection networks are the topology and the routing algorithm [2, 3]. Topology defines how the components of the system are connected, and the routing algorithm determines the path that is followed by packets from their source to their destination node. The topology of a network also impacts, to a large extent, its cost. Topologies usually adopt a regular structure to simplify their implementation, the routing algorithm and the possibility of expanding the network. Among the different taxonomies of regular topologies, the most commonly-used one divides them into direct and indirect topologies [2, 3].

Direct topologies usually adopt an orthogonal structure where nodes are organized in an n–dimensional space. Each node is composed of a router and a processing node. The nodes are connected in each dimension in a ring or array fashion. 2D or 3D direct topologies are relatively easy to built as each topology dimension is mapped to a physical dimension. Implementing direct topologies with more than three dimensions implies not only increasing its wiring complexity but also the length of its links when they are mapped to the 3D physical space. Indeed, the number of ports of the routers geometrically grows with the number of dimensions (as two ports per each dimension are required). The implementation limitation in the number of dimensions leads to increase the number of nodes per dimension, which increases the communication latency, negatively impacting performance. As a consequence, direct topologies are not the most suitable ones for large machines.

The alternative is to use an indirect topology. The main difference, compared to direct topologies, is that not all the routers have an associated processing node. The most common indirect topologies are multistage indirect networks (MINs) where switches are organized in a set of n stages. Indirect topologies provide better performance for a large number of processing nodes than direct ones. However, this is achieved by using a higher amount of switches and links. Furthermore, their physical implementation is complex due to the fact that the wiring complexity grows with the number of processing nodes in the system, unlike direct topologies where

complexity grows with the number of topology dimensions.

To overcome the limitations of direct and indirect topologies, in this paper we propose a new family of topologies where we combine the best features of both types of topologies. Other hybrid topologies have been proposed previously in the literature (see Section 4.6). However, most of them are hierarchical and therefore introduce long latency for the communication of non-local processing nodes and others are intended for particular purposes [20, 21]. In this paper we propose a topology family that can be configured to meet different scenario requirements. We propose an n–dimensional topology, where the rings that connects the nodes in each dimension are replaced by small indirect networks. In this way, communication latency along each dimension no longer linearly grows with the number of nodes per dimension thanks to these indirect networks. On the other hand, the small size of this indirect topology allows a reasonable wiring complexity opposite to large indirect topologies. This combination results in a family of topologies that provides high performance, with latency and throughput figures of merit close to the ones obtained with indirect topologies, but at a reduced hardware cost. In addition, the fault–tolerance level is higher than or equal to the one provided by direct and indirect topologies.

Evaluation results show that the new proposed family of topologies can obtain similar or better performance results than the ones provided by indirect topologies, but using a smaller amount of hardware resources and with an easier implementation. In particular, they are able to double the throughput obtained per cost unit compared to the one obtained with indirect topologies in most of cases, and this difference is even higher when it is compared with direct topologies.

The rest of the paper is organized as follows. Section 4.2 presents some background. Section 4.3 describes the proposed family of topologies. The routing algorithms to be used are presented in Section 4.4. Section 4.5 evaluates the new family of topologies, comparing it against direct and indirect topologies, and other recently proposed topologies. Some related works are commented in Section 4.6. Finally, some conclusions are drawn.

4.2 Direct and Indirect Topologies

In direct topologies, each node has its own router that connects it to a subset of the nodes of the system by means of point-to-point links. Direct topologies usually adopt an orthogonal structure, where nodes are organized in an n–dimensional space, in such a way that traversing one link produces a displacement in only one dimension. That is, all the links of the network

are organized in several dimensions in a regular way, and each node has at least one link in each dimension. The symmetry and regularity of these networks greatly simplify its implementation and the routing algorithm, since the movement of a packet in a single dimension does not modify the number of hops remaining in the other dimensions to reach the packet destination.

This kind of topologies is known as k–ary n–cubes. In what follows, to distinguish the topology parameters of both direct and indirect topologies, we will refer to this topology as k_d–ary n_d–cubes[1], where k_d is the number of nodes in each of the n_d dimensions in a direct network. The total number of processing nodes in the system is given by $N = {k_d}^{n_d}$. In these topologies, nodes are labeled by an identifier with as many components as topology dimensions $\{p_{n_d-1}, \ldots, p_0\}$, and the component associated to each dimension ranges from 0 to $k_d - 1$ (i.e., nodes are numbered from $\langle 0, 0, \ldots, 0\rangle$ to $\langle k_d - 1, k_d - 1, \ldots, k_d - 1\rangle$). The identifiers of neighbor nodes in a given dimension only differ in the component corresponding to that dimension, while the remaining components have the same value. For instance, two nodes p y p' are neighbors in the x dimension, if and only if $p_x = p'_x \pm 1$ and $p_i = p'_i$ for the rest of dimensions.

The most commonly–used direct topologies are mesh, torus and hypercube. In a mesh topology, all the nodes of a dimension compose a linear array. In torus, all the nodes of each dimension form a ring. The hypercube is a particular case of a mesh where there are only two nodes in each dimension ($k_d = 2$), which forces the number of dimensions (n_d) to be large enough to interconnect all the nodes of the system (i.e., $n_d = log_2 N$). Direct topologies have been used in several of the most powerful supercomputers, being the 3D-torus the most widely used one. For instance, the number two (Titan) and number three (Sequoia) of the November 2015 Top500 supercomputer list [1] use a torus.

For a given number of processing nodes N, direct topologies provide a richer connectivity as the number of dimensions increases. As the hypercube has the largest possible number of dimensions, it provides a better connectivity than meshes and tori, but at a higher cost, since it uses more links and it has a higher router degree (number of ports of the router).

Latency is related to the average distance, measured as the number of links that packets have to cross to reach their destination. Related to this, the diameter of a topology measures the maximum distance between two nodes of the topology using the shortest possible path between them. For a constant number of processing nodes ($N = k_d^{n_d}$), diameter is increased as the number of

[1]The subscript "$_d$" stands for direct.

dimensions (n_d) is decreased and the number of nodes per dimension (k_d) is increased. The increase of the distance traversed by packets also increases the probability of contention with other packets that are also crossing the network, which reduces network throughput. From this point of view, it may seem interesting to maximize the number of dimensions of direct topologies for a given number N of processing nodes. Nevertheless, other issues have to be considered. Direct topologies up to three dimensions can be implemented by using relatively short links in the 3-D physical space and with an acceptable wiring complexity, regardless of the size of the system. However, implementing a topology with more than three dimensions implies increasing the length of the links, since we have to map all the dimensions to the 3-D physical space. In addition, they also require using routers with a higher number of ports (two bidirectional ports per each dimension are required, one per each direction in that dimension). So, direct topologies have some limitations for the implementation of large-scale machines, since they would require using a large number of nodes per dimension (k_d), which increases the diameter of the network and also the probability of contention which increases message latency of communications and reduces network throughput.

The alternative is to use indirect topologies. In these topologies, processing nodes do not have routing capabilities, since the router is separated from the processing node[2]. Routers become independent devices, known as switches. Processing nodes are connected to a switch of the network. Moreover, opposite to direct topologies, all the switches have not necessarily an associated processing node. In fact, most of the switches are usually connected to other switches but they are not connected to processing nodes.

The most common indirect networks are multistage interconnection networks (MINs). In MINs, switches are organized as a set of stages. Processing nodes are connected to the first stage, and the other stages are connected among themselves using a certain connection pattern that provides full–connectivity among all processing nodes. Two different types of MINs can be defined. Unidirectional MINs (UMINs) use unidirectional switches and links, so the network is traversed by packets only in one direction. In this case, processing nodes are also attached to the last stage and a unique path between each source–destination pair is provided (using the minimum number of stages). Bidirectional MINs (BMINs) use bidirectional links and switches. So, in order to travel from a source to a destination processing node, packets travel upwards the network and then downwards. BMINs provide several paths between each source–destination pair.

[2]Direct topologies with independent routers are also possible.

In MINs, connection patterns between stages are based on permutations of the identifiers of the processing nodes. Depending on the connection pattern used among adjacent stages, several MINs have been proposed. The most widely–used MIN in commercial systems is the fat–tree topology [4], which is a BMIN. However, there are other more recent systems, like the one proposed in [6], focusing on fault tolerance. Most of the high–performance interconnect vendors as Mellanox (manufacturer of the Infiniband technology) [7], Myricom (manufacturer of Myrinet) [8] or Quadrics (manufacturer of QsNet) [48] recommend to use a fat–tree and provide specific switches for building this topology. Moreover, it has been used in some of the most powerful supercomputers. For instance, the Tianhe 2 and the Tianhe 1A supercomputers, the number one and twenty six respectively in the November 2015 Top500 list [1] use this topology.

The k–ary n–tree is the most widely–used implementation of the fat–tree topology. In what follows, to distinguish the topology parameters of both direct and indirect topologies, we will refer to this topology as k_i–ary n_i–tree[3]. k_i is the number of links that connect a switch to the next or the previous stage, and n_i is the number of stages of the indirect network. So, each switch has $2k_i$ bidirectional ports in fat–trees (i.e. using switches with $d \times d$ ports, k_i is $d/2$). A fat–tree requires at least $log_{k_i} N$ stages to interconnect N processing nodes. Each stage has N/k_i switches, with a total of $(N/k_i)log_{k_i} N$ switches and $N = {k_i}^{n_i}$ processing nodes.

In fat–trees, the diameter of the network depends only on the number of stages, and it is given by $2n_i$, that is $2log_{k_i} N$, as in the worst case a packet must traverse upwards all stages and all stages downwards. Notice that, for a UMIN, the distance traversed by packets is always n_i, regardless of the source and destination processing nodes. In summary, for a fixed number $N = {k_i}^{n_i}$ of processing nodes of a MIN, when the number of stages n_i is increased, k_i is decreased but the distance that packets have to traverse to reach the destination is increased. Also it should be taken into account that, by using high-degree switches fewer switches will be required, but each of them will be more complex. On the contrary, if we use low–degree switches, we will require more switches, but much simpler.

Table 4.1 shows the diameter and bisection bandwidth for both types of topologies. As it can be seen, the bisection bandwidth is larger for indirect topologies –it depends only on N, without being divided– and the diameter of indirect topologies depends only on the number of stages while in direct topologies it depends on the product of n_d and k_d, which will lead in practice to

[3]The subscript "$_i$" stands for indirect.

TABLE 4.1: Parameters of the different analyzed topologies.

Topology	Diameter	Bisection bandwidth
Mesh	$n_d(k_d - 1)$	$2N/k_d$
Torus	$n_d(k_d/2)$	$4N/k_d$
Fat–tree	$2n_i$	N

larger diameters. In direct topologies, for a low number of network dimensions (remember that it is limited by the 3D physical space), the number of nodes per dimension will be high, negatively impacting diameter and bisection bandwidth. Indirect networks provide better scalability than direct networks, because they provide smaller diameters and shorter latencies for large network sizes. Nevertheless, they have a higher cost, because they require a high amount of switches and links, and their physical implementation is costly since the complexity of network wiring grows with its size, unlike direct topologies.

4.3 The KNS Family of Hybrid Topologies

This paper proposes a new family of topologies for interconnection networks that allows to efficiently connect an extremely high number of processing nodes, given the huge size of current and near future supercomputers [1]. We propose an hybrid topology based on combining an n–dimensional direct network with small indirect topologies. The aim is to combine the advantages of direct and indirect topologies to obtain a family of topologies that is able to connect a high number of processing nodes, providing low latency, high throughput and a high level of fault–tolerance at a lower hardware cost than indirect networks. In the proposed topology, the nodes of each dimension are connected through an indirect topology that allows to have a large number of nodes per dimension without negatively affecting performance.

The new family of topologies will use two different kind of switches in the network (although this is not entirely true, as we will see later). First, low–degree switches are used to connect processing nodes to each dimension and move packets across dimensions. We will refer to these switches as *routers*. They have, at least, one processing node attached to them and as many ports as dimensions. Nowadays, it is common that processing nodes are composed of a large number of cores, and the core count per processing node trend is to increase even further. So, these processing nodes need network interfaces with high bandwidth to avoid bottlenecks in the end processing node connection to the network. In fact, there are already some commercial

solutions which use network interfaces cards with dual ports [47]. Considering the core count increase trend it is expected that the bandwidth requirements of end processing nodes will be even higher. Another key point of these network interfaces with several ports is that they provide fault-tolerance. Using one port network interfaces causes having a single point of failure that will disconnect a high number of cores. Both, bandwidth and fault tolerance requirements will provoke increasing the number of ports in the near-future network interfaces.

In this paper we take advantage of these network interface cards with several ports to implement the routers used in the new family of topologies.

Additionally to these routers, the proposed topology also uses other *switches* to implement the indirect networks that interconnect the routers of each dimension.

4.3.1 Description of the family

The newly proposed topology family arranges processing nodes and their associated routers (node) in n dimensions like a direct topology. But, contrary to mesh and torus topologies, routers of a given dimension are not only connected with their adjacent routers in that dimension. Instead, all the routers of a given dimension are connected by means of a small indirect network. This indirect network could be even a single switch, considering the number of ports of current commercial high-radix switches. However, if the number of routers per dimension exceeds the number of ports of the available switches, the indirect network will be arranged as a small MIN. We will refer to this MIN as the *indirect subnet*. This will provide a low latency communication among routers in the same dimension with a small hardware extra cost compared to direct topologies. In this way, the number of routers per dimension stops being a bottleneck from the point of view of the performance, as the time to communicate two routers of the same dimension is constant or grows logarithmically with the number of routers per dimension. Notice also that each router only requires a bidirectional link per dimension to connect to the switch of each dimension. On the contrary, a mesh or torus require two bidirectional links per dimension, one per neighbor node.

The proposed family of topologies is defined by three parameters. Two of them are inherited from direct networks: the number of dimensions n_h[4] and the number of routers per dimension k_h. The number of processing nodes it can interconnect is given by $N = k_h^{n_h}$. In addition

[4]The subscript "h" stands for hybrid.

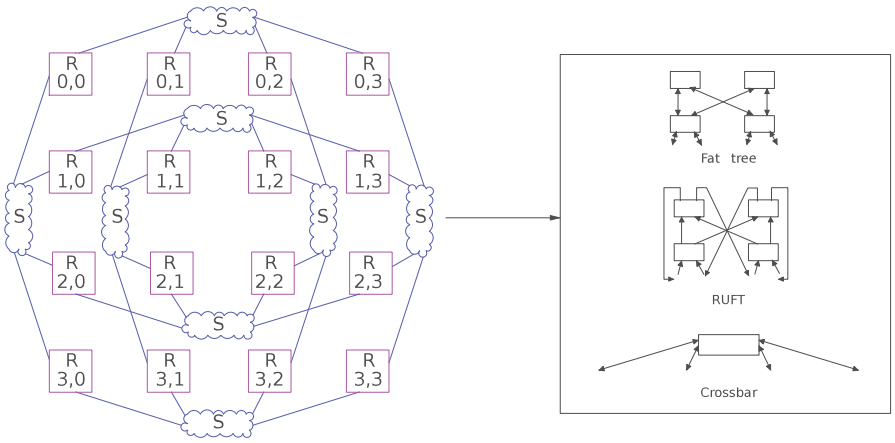


FIGURE 4.1: An example of the KNS topology with $n_h = 2$, $k_h = 4$ and $d_h = 4$.

to these two parameters, there is an additional parameter, the number of stages of the indirect subnet, referred to as s_h. This number depends on the number of routers per dimension k_h and on the number of ports of the switches used to implement the indirect subnet (which will be referred to as d_h). The ratio between k_h and d_h defines the way to interconnect the k_h routers of each dimension. If $k_h \leq d_h$, a simple switch would be able to interconnect all the routers of the dimension, and s_h will be equal to 1. On the contrary, if $k_h > d_h$, a MIN would be required to interconnect the routers of each dimension, and the number of required stages will be given by $log_{\frac{d_h}{2}} k_h$ (remember that in an indirect network built with d_h-port switches, the network radix is equal to $\frac{d_h}{2}$). We will refer to the new family of topologies proposed in this paper as k_h–ary n_h–direct s_h–indirect (KNS), where k_h is the number of routers per dimension, n_h is the number of dimensions and s_h is the number of stages of the indirect subnets.

In this paper, we have considered two different MINs to connect the routers of a given dimension. The first one is a BMIN, the fat–tree. The second one is RUFT [5], a UMIN derived from a fat–tree using a load–balanced deterministic routing algorithm [22], which requires less hardware resources.

Figure 4.1 shows an example of the new topology with 2 dimensions ($n_h = 2$), and 4 routers per dimension ($k_h = 4$), with a total number of 16 routers. In this case, the routers of the same dimension are connected by a single 4-port crossbar. However, for a higher number of routers per dimension, say $k_h = 8$ and the same switch size, a MIN should be used. In the case of using fat–tree subnets, for interconnecting the 8 routers of each dimension, it will require 3 stages and 12 bidirectional 4-port switches, implementing in this way a 8–ary 2–direct 3–indirect. In the

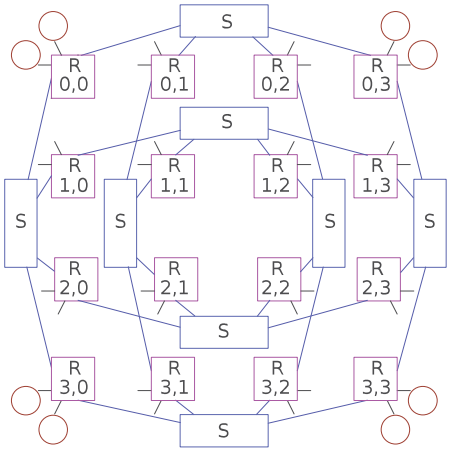


FIGURE 4.2: An example of the KNS topology with $n_h = 2$, $k_h = 4$, $s_h = 1$ and $p_h = 2$.

case of using RUFT as indirect subnets, it is also a 8–ary 2–direct 3–indirect but built with 4-port unidirectional switches. We will analyze in more depth the issues of using different indirect subnets in Section 4.5.

As it can be seen in Figure 4.2, the routers are connected to all dimensions through a different link (one per dimension). Notice that it is also possible to attach several processing nodes to the same router. Having more than one processing node attached to each router is known as concentration in the literature. In fact, the topology shown in the figure already has two processing nodes attached to each router. Only the corner processing nodes are shown in the figure for the sake of clarity. This introduces a new parameter on the topology family, p_h, the number of processing elements that are connected to each router. In this case, the number of processing nodes is given by $N = p_h k_h^{n_h}$.

In Figure 4.1, two different components with switching capabilities can be distinguished. The switches[5] (in blue and labelled with a "S" in Figure 4.1) are only connected to other switching components, whereas the routers (in pink and labelled with an "R" in Figure 4.1) are used to connect the processing nodes to the network through several dimensions. These routers are connected on one side with processing nodes and on the other side with switches, and their degree is $n_h + p_h$, that is, the sum of the number of dimensions and the number of concentrated processing nodes. As it can be seen in Figure 4.1, if a crossbar is used as indirect subnet, switches allow packets to change to any position in a given dimension by traversing only two links, whereas routers allow to travel between dimensions. Thus as, at most, only two hops are performed per dimension, the diameter of the new topology is $2n_h$, which is a very low value.

[5]It may be a single switch or a set of switches forming a MIN.

TABLE 4.2: Parameters of the different analyzed topologies.

Topology	Parameters	
Mesh or Torus	n_d	# dimensions
	k_d	# of nodes per dimension
Fat–tree	n_i	# of stages
	k_i	switch arity
k_h–ary n_h–direct s_h–indirect (KNS)	n_h	# of dimensions
	k_h	# of routers per dimension
	s_h	# of stages of the indirect subnet
	d_h	switch degree
	p_h	# processing nodes per router

If a MIN is used instead of a crossbar as indirect subnet, the diameter of the network will be the diameter of the used MIN (for a fat–tree, i.e. a BMIN, $2\times$ the number of stages, which is expected to be a small number due to using high-radix switches) multiplied by n_h, that is $2s_h * n_h$.

Table 4.2 summarizes the parameters that define the hybrid family of topologies. Notice that only one of d_h or s_h parameter is necessary, as the another one can be derived from the other parameters. In addition, the parameters that define traditional direct and indirect topologies are also shown.

The proposed hybrid topologies can take advantage of the high number of ports available in current switches. For example, edge switches of up to 36 ports are commercially available [68], while chassis switches offer up to 648 ports [68]. With such switches, in most cases, a small MIN with only 2 or 3 stages or even a single switch will be enough to connect the routers in each dimension. As the MIN is very small, it will introduce low latency and it can be easily implemented with low wiring complexity.

On the other hand, as already stated, this topology can take advantage of new network interfaces (like the new commercial HCA cards [47]), that have several ports to connect the processing nodes to the network. By using these network interfaces, the proposed topologies can be implemented by integrating the router into the processing node as part of the network interface. These network interfaces will have switching capabilities, so processing nodes equipped with these new network interfaces will be able, apart from injecting messages in the network, to route packets that are not destined to them to other processing nodes without ejecting messages from the network.

The resulting topologies could seem similar to BCube [20] or Hypercrossbar network [21], but the great difference between our proposal and BCube is that the processing nodes do not eject messages from the network; the new topology family forwards the messages in the network interface which highly reduces latency. However, these cases could be considered as particular configurations of the KNS family of topologies.

The topologies proposed in this paper have several main advantages. First, they allow to highly reduce the diameter compared to direct topologies. This will lead to network performance improvements, decreasing latency and increasing network throughput. Additionally, the number of required switches and links is reduced compared to an indirect topology that connects the same number of processing nodes, as will be shown in section 4.5. Therefore, it is expected that the proposed family of topologies reduces the cost of the network. Finally, it also provides a good fault–tolerance level (see Section 4.5).

4.4 Routing Algorithms for the KNS Family of Topologies

In this section, we describe the routing algorithms proposed for the new family of topologies. We will first describe the ones proposed for k_h–ary n_h–direct 1–indirect topologies (i.e., a crossbar is used as indirect subnet). Then, we describe the ones proposed for the general case, that is, for KNS topologies, using a fat–tree or a RUFT as indirect subnets.

First, we explain how routers and switches are labeled in the KNS topology. Each router is labeled as in meshes and tori, with a set of components or coordinates (as many as network dimensions) $\langle r_{n_h-1}, r_{n_h-2}, \ldots, r_1, r_0 \rangle$. Each coordinate represents the position of each router in each of the dimensions. On the other hand, the switches are labeled by a 2-tuple *[d,p]*, where *d* is the dimension the switch is located at, and *p* is the position of that switch in that dimension. Notice that routers do not belong to any dimension, since they are connected to all of them, and packets change dimensions through them. On the contrary, switches do not allow changing the dimension packets are traversing, they just allow packets to move through that dimension.

4.4.1 Routing in k_h–ary n_h–direct 1–indirects

Although both deterministic and adaptive routing algorithms could be used, taking into account that adaptive routing may introduce out-of-order delivery of packets and that leads to a more

complex implementation, in this paper, we will focus only on deterministic routing.

The deterministic routing algorithm for k_h–ary n_h–direct 1–indirect topologies, which will be referred to as Hybrid–DOR, is a variation of the dimension ordered deterministic routing algorithm (DOR) for meshes, adapted to the k_h–ary n_h–direct 1–indirect topology. In DOR, packets are routed through the different dimensions following an established order until the destination processing node is reached. At each dimension of the mesh, packets traverse several routers until the movement in that dimension is exhausted. On the other hand, as each mesh router has two links per dimension, packets must be forwarded in each dimension through the direction that guarantees the minimal path.

In Hybrid–DOR, network dimensions are also crossed in an established order to guarantee deadlock freedom, as in DOR. However, there is a unique link per dimension that connects the current router to a switch that allows directly reaching any of the processing nodes in that dimension. So, packets do not perform several hops at each dimension. Instead, in Hybrid–DOR, routers directly forward packets through the unique link of the dimension they have to traverse, and this link is connected to the corresponding crossbar that moves the packet to the destination component in that dimension. Notice that, contrary to meshes and tori, in k_h–ary n_h–direct 1–indirect topologies, it is not required to choose the direction at each dimension, as there is only one link per dimension. The routing in the switches is very straightforward since, they just must forward packets through the link indicated by the destination component in the current dimension, requiring just one hop to reach next router.

Next, we show the Hybrid–DOR pseudo–code for the routers and the crossbars of the network. The number of dimensions of the topology is n_h and the destination and current router coordinates are given by $\langle x_{n_h-1}, \ldots, x_{d+1}, x_d, x_{d-1}, \ldots, x_1, x_0 \rangle$, and $\langle r_{n_h-1}, \ldots, r_{d+1}, r_d, r_{d-1}, \ldots, r_1, r_0 \rangle$, respectively. In the case of crossbars, the current switch is given by *[d,p]* (the p^{th} switch of the d dimension). The chosen link to send the packet is returned in $link$.

Routers:

$i = 0;$

$Done = False$

while $(i < n_h) \wedge (!Done)$ **do**

if $x_i! = r_i$ **then**

$Done = True$

$link = i$

end if

$i = i + 1$

end while

Crossbars:

$link = x_d$

As can be seen, routers select the next dimension to forward the packet, which it is also the link of the router to be used, since there is just one link per dimension, and crossbars merely select the link given by the destination coordinate of the corresponding dimension to reach the destination component in that dimension.

4.4.2 Routing in k_h–ary n_h–direct s_h–indirects

In k_h–ary n_h–direct s_h–indirect topologies, the crossbars are replaced by small MINs. As stated above, the MINs considered in this paper are fat–trees or RUFTs. In these topologies, all the switches of a given fat–tree or RUFT are always in the same dimension and in the same position relative to the routers. In order to identify the switches inside a given fat–tree or RUFT, we extend the classical switch coordinates from MINs by including the coordinates of the MIN in the direct topology. In this way, the switch coordinates in k_h–ary n_h–direct s_h–indirect topologies will be given by a 4-tuple *[d,p,e,o]*, where d is the dimension the MIN belongs to, p is the position of the MIN in that dimension, e is the stage of the switch inside the MIN, and o is the order of that switch in that stage. Remember that d and p are the coordinates of the equivalent crossbar in k_h–ary n_h–direct 1–indirect topologies.

Since the routers are the same regardless of the indirect topology used, its routing algorithm is the same as the one shown for k_h–ary n_h–direct 1–indirect topologies. However, switch routing algorithm depends on the particular indirect network used.

First, we focus on the k_h–ary n_h–direct s_h–indirect topology that uses fat–trees in the indirect subnets. Despite the fact that a fat–tree has several paths for each source–destination pair (i.e.,

it allows adaptive routing), we propose to use the deterministic routing algorithm presented in [22] since it is simpler and is able to outperform adaptive routing. We will summarize that routing algorithm here. Routing is composed of two subpaths. First, packets are sent upwards to the common ancestor switch of the source and destination processing nodes and, then, they are sent downwards to its destination. Traffic is balanced by carefully selecting the links to be used according to the destination processing node. In particular, the link to be used in both subpaths is given by the destination coordinate corresponding to the stage where the switch is located at. For instance, if a switch located at stage e_1 routes a packet whose destination (in the fat–tree) is $\langle t_{n_i-1}, \ldots, t_{e_1+1}, t_{e_1}, t_{e_1-1}, \ldots, t_1, t_0 \rangle$, then the packet is sent through the link $k_i + t_{e_1}$ in the upwards phase and through link t_{e_1} in the downwards phase. Remember that k_i is the arity of the switches of the fat–tree topology. Please see [22] for more details.

In the fat–trees subnets of the k_h–ary n_h–direct s_h–indirect topologies, the routing algorithm is the same, but only the part of the destination identifier corresponding to the dimension the fat-tree belongs to (i.e., x_d in our notation) is used. In this way, the packet is delivered to the same router that would be reached through the corresponding crossbar in a k_h–ary n_h–direct 1–indirect topology.

This routing algorithm is shown below. Assume that destination coordinates are $\langle x_{n_h-1}, \ldots, x_{d+1}, x_d, x_{d-1}, \ldots, x_1, x_0 \rangle$, the dimension where the fat–tree is located at is d, $GetFTIdentifier$ returns from x_d the fat–tree coordinates to route locally in the fat–tree, $UpwardsPhase$ returns *true* if the packet is in its upwards subpath, or *false* otherwise, and that the stage in the fat-tree of the switch that is routing the packet is given by e:

Switches:

$t = GetFTIdentifier(x_d)$
if $UpwardsPhase()$ **then**
  $link = k_i + t_e$
else
  $link = t_e$
end if

Let us consider the case where RUFT is used in the indirect subnets of the KNS topology. In RUFT, there is a unique path between each source–destination pair and packets have to cross all the stages, reaching the last stage, which is directly connected back to the processing nodes. The link to be used by a packet at a particular switch is given by the destination component

corresponding to the stage the switch is located at. Please see [5] for details. In this case, the pseudo–code for the switch routing algorithm is the following:

Switches:

$t = GetFTIdentifier(x_d)$

$link = t_e$

If Hybrid-DOR is used in the routers jointly with the aforementioned algorithms for the switches in the indirect networks, the resulting routing algorithm for the new topology is deterministic and deadlock-free, since dimensions are crossed in order in the direct topology and the routing algorithm used in the indirect networks has not any loop in its channel dependency graph [56].

4.5 Evaluation

In this section, we evaluate the KNS topology family, comparing it with other topologies such as meshes, tori, fat-trees, and flattened-butterflies [11].

4.5.1 Network Model

To evaluate the family of topologies proposed above, a detailed event–driven simulator has been implemented. The simulator models several topologies, including the new family of topologies presented in this paper, the KNS. This simulator uses virtual cut–through switching. Each switch has a full crossbar with queues of two packets both at their input and output ports. Credits are used to implement the flow control mechanism. We assumed that it takes 20 clock cycles to apply the routing algorithm; switch and link bandwidth has been assumed to be one flit per clock cycle; and fly time has been assumed to be 8 clock cycles. These values were used to model Myrinet networks in [52]. In addition, for the new topology using RUFT as indirect subnet, the fly time of the long links that connects the output of the unidirectional MIN to the direct routers is assumed to be 8 clock cycles per stage, in order to take into account that these links are longer.

We have performed the evaluation by using several synthetic traffic patterns: uniform, hot–spot, tornado, and complement. In the uniform traffic pattern, message destination is randomly chosen among all destinations. In the hot–spot traffic pattern, a percentage of traffic is sent to a small

subset of the processing nodes (5% of nodes in this case) and the rest of the traffic is uniformly distributed. In complement, the destination processing node is obtained by complementing all the component bits of the source processing node. Therefore, in this traffic pattern, the destination processing node of all packets generated at a given source processing node is always the same. In tornado [69], the destination is chosen in such a way that each packet travels $n(\frac{k}{2}-1)$ hops. Regarding packet size, the results shown in this paper have been obtained for 256–flit packets. However, simulations with other packet sizes, such as 16, 128, and 512 flits has been performed, and the results are consistent with the ones shown here. For each simulation run, the full range of injected traffic (from low load to saturation) has been tested.

4.5.2 Performance Results

In this section, we compare the KNS using different indirect subnets (crossbar, fat-tree, and RUFT) against other well–known and frequently–used topologies such as tori, meshes, and fat–trees. Moreover, we also compare our proposal against the flattened–butterfly (FB) topology because it is becoming a popular topology in recent research papers (see Section 4.6 for further details). The FB topology is a variation of the butterfly topology obtained from using high-radix switches, that results in a direct topology. This topology can be seen as a generalized hypercube with concentration, as all the switches in the same dimension are directly connected, that is, there is a link from each switches to the others of the same dimension. Several FB configurations have been tested and compared to our proposal.

We have evaluated a wide range of network sizes, from 64 to 64K processing nodes. Larger topologies have not been simulated due to simulator memory constraints. For direct topologies, we have tested different values of the number of dimensions and number of nodes per dimension. In particular, we have evaluated networks of 2 dimensions, with 4, 8, 16, 32, 64, and 256 nodes per dimension; three dimensions, with 4, 8, and 16 nodes per dimension; four dimensions, with 4, 8, and 16 nodes per dimension; six dimensions, with 4 processing nodes per dimension; and eight dimensions, with 4 nodes per dimension. If not stated the contrary, only one processing node is attached to each router (i.e. without concentration). If several ones are attached, the x–p suffix is used, x being the number of processing nodes attached to each router. These networks are compared with fat–trees and FBs with the same number of processing nodes. Notice that, in some cases, several configurations are possible. For the sake of clarity, only a subset of the most representative simulations is shown.

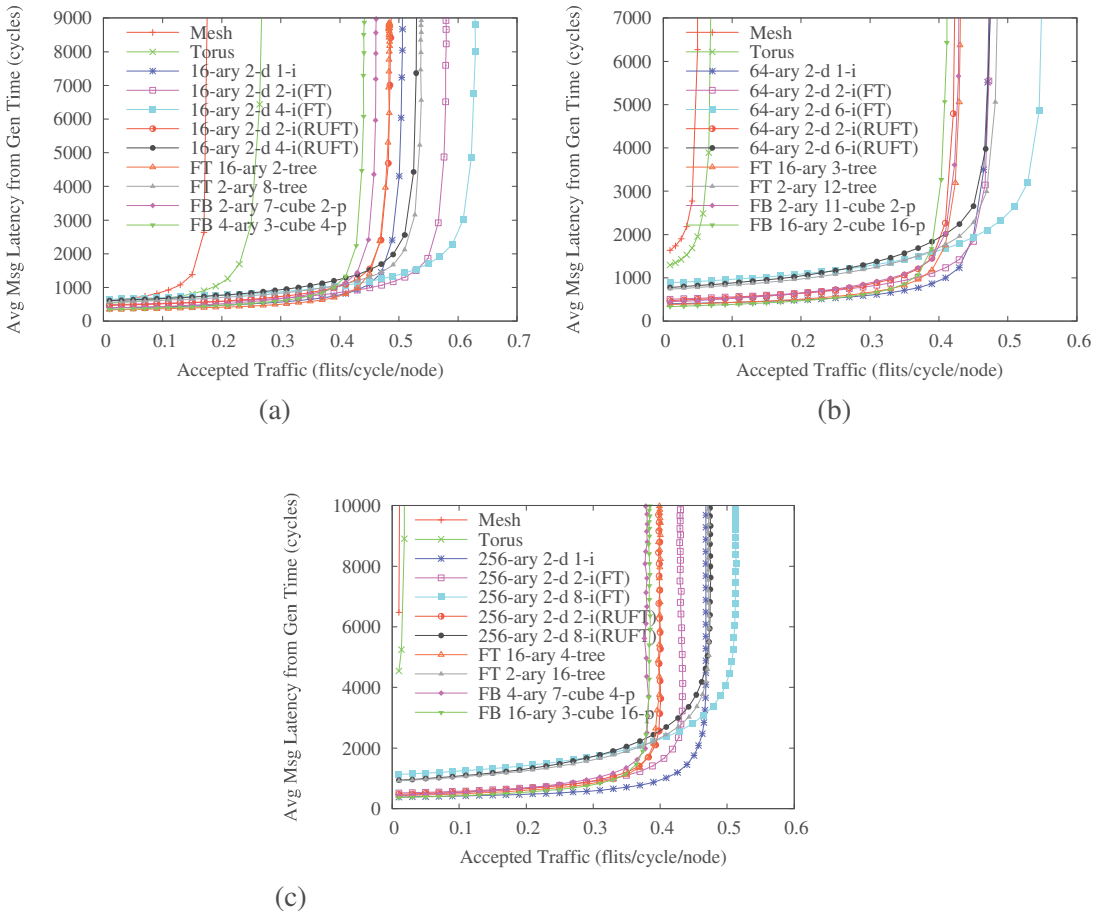


FIGURE 4.3: Average packet latency from generation vs. accepted traffic for uniform traffic and 2 dimensions for direct topologies. (a) 256 processing nodes. (b) 4K processing nodes. (c) 64K processing nodes.

Uniform traffic pattern: Figure 4.3.(a) shows results for 2-D small networks (256 processing nodes) with uniform traffic. As it can be seen, the mesh is the network that achieves the lowest throughput, followed by torus and the FBs configurations. In this latter case, we have selected a 4–ary 3–cube and a 2–ary 7–cube FB with concentration in order to compare our proposal against a topology with a hardware of similar complexity, as we will show later (in Section 4.5.3). The next topologies that achieve a better performance are the two fat–tree configurations. However, the best absolute throughput is achieved by the family of topologies proposed in this paper. In particular, the different tested configurations ordered from lower to higher throughput are the topology that uses a RUFT with two stages as indirect subnet (16–ary 2–d 2–i (RUFT)), the ones that uses crossbar (16–ary 2–d 1–i), the one that uses RUFT with 4 stages (16–ary 2–d 4–i(RUFT)), the one that uses a FT with two stages (16–ary 2–d 2–i (FT)) and the one that uses a FT with four stages (16–ary 2–d 4–i (FT)). In particular, the best configuration of the

new topology family obtains 3 times more throughput than the worst network (mesh), more than twice versus torus, more than 20% versus FT and about 40% improvement versus FB.

Figures 4.3.(b) and 4.3.(c) show how throughput is decreased in all the topologies as we increase the number of processing nodes in the network, keeping constant the number of dimensions in the direct topologies, and therefore increasing the number of routers (processing nodes) per dimension. In the case of mesh and torus topologies, throughput strongly decreases, as the average distance between two nodes is markedly higher than in the other topologies. Regarding the KNS topologies, all the tested configurations outperform both the FT and FB configurations analyzed. The best configurations are again the ones that use the tallest FT as indirect subnet. Notice that the different topologies have a different hardware cost, which is evaluated in following sections.

In Figures 4.3.(a), 4.3.(b), and 4.3.(c) we can also see the impact of using more stages in the MINs of KNS topologies. For the same number of routers per dimension, if we decrease the number of stages, the arity of the switches is increased, and a lower latency should be obtained. The plots show that, the higher the number of stages, the higher the base latency (in more detail the zero-load latency) as more switches have to be crossed by packets. Surprisingly, networks with more stages also achieve more throughput. This effect is explained by the reduction of the head-of-line (HoL) blocking effect. For a given number of routers per dimensions, a taller FT uses smaller switches (i.e. with lower number of ports). As a consequence, each switch port is potentially demanded by a lower number of input ports and, hence, the effect of HoL blocking is reduced. From another point of view, with fewer stages, each indirect topology has less switches to serve the same number of routers. Thus, each switch has to deal with more traffic, leading to more HoL blocking effect and, hence, less throughput.

Let us analyze the base latency. In a k_h–ary n_h–direct 1–indirect, base latency does not depend on the number of routers per dimension. However, in KNS that uses MINs, the base latency increases with the number of processing nodes because the number of stages in the indirect subnets also grows in order to connect a larger number of routers. This effect is more prominent in RUFT, due to the fact that packets traverse always all stages since it is a UMIN topology. In the case of torus and mesh, base latency strongly depends on the number of nodes per dimension, as average distance between nodes is increased.

Figure 4.4 analyzes the impact of the number of dimensions in the different topologies. We analyze a network with 64K processing nodes implemented with a different number of dimensions. We can distinguish three different behaviors. First, Mesh and torus topologies have a similar

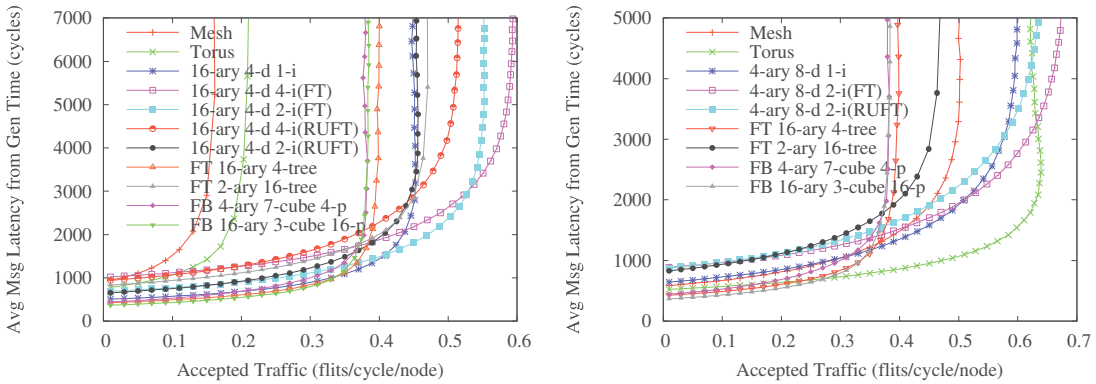


FIGURE 4.4: Average packet latency from generation vs. accepted traffic for uniform traffic with 64K processing nodes and different number of dimensions: (a) 4D and (b) 8D.

behavior. The higher the number of dimensions, the fewer the number of nodes per dimension, and the higher the achieved throughput. Also, base latency decreases with the number of dimensions, because the average distance is reduced. However, the behavior of k_h–ary n_h–direct 1–indirect is different. Throughput also increases with the number of dimensions because the size of switches of the indirect network (a crossbar in this case) is reduced, and, hence, the pernicious effect of HoL blocking is reduced. However, base latency does not improve with the number of network dimensions. This is due to the fact that the number of hops that packets must perform also grows with the number of dimensions. Concerning the k_h–ary n_h–direct s_h–indirect, they have a similar behavior to the previous one, but with a difference. The base latency, in this case, slightly decreases when the number of dimensions increases. Although network diameter increases with the number of dimensions, as started above, as there are fewer routers per dimension, indirect subnets have fewer stages, and, thus, packets have less stages to cross. Finally, the configurations of the FB shown (the ones with a hardware cost similar to the one of our proposal, see Section 4.5.3) and FT obtain an intermediate throughput value. Anyway, we would like to remark that the new family of topologies always obtains the best throughput regardless of the number of dimensions.

Complement traffic pattern: Figure 4.5 shows the obtained results for this traffic pattern for different networks (4K and 64K processing nodes). We can distinguish two different behaviors among the analyzed topologies. In torus, mesh, and FB topologies, the network is rapidly saturated. In the rest of topologies (all the KNS topologies, and fat–trees), the network is able to

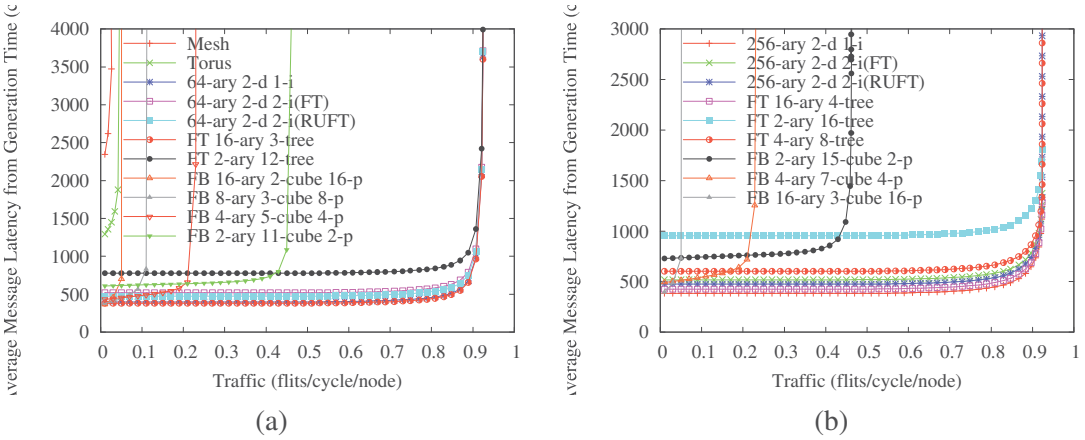


FIGURE 4.5: Average packet latency from generation vs. accepted traffic for complement traffic and (a) 4K processing nodes and (b) 64K processing nodes.

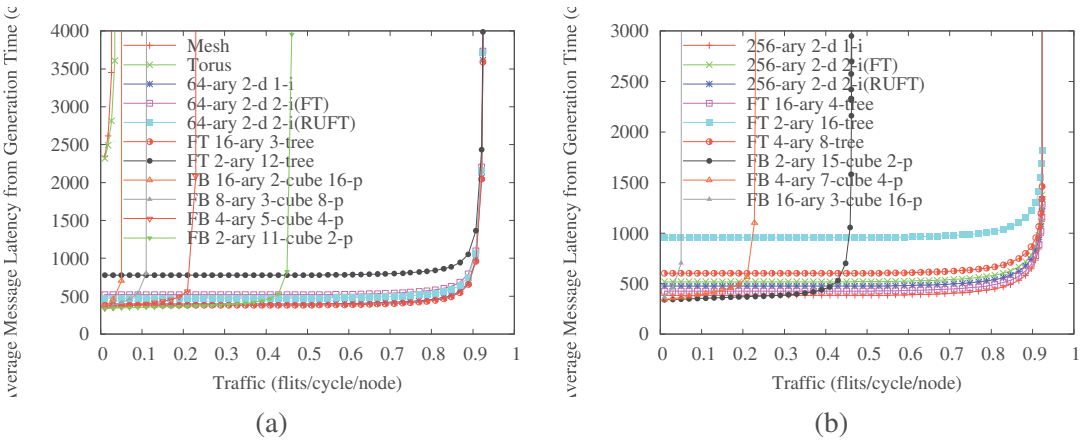


FIGURE 4.6: Average packet latency from generation vs. accepted traffic for tornado traffic and (a) 4K and (b) 64K processing nodes.

cope with all the injected traffic. The reason is that, for this traffic pattern, as an optimal load-balanced routing algorithm [22] is used in FT and RUFT, the network resources are not shared among source–destination pairs. The same happens in the indirect subnets of the proposed family of topologies as they use the same routing scheme, and, in addition, links connecting routers and switches only forward packets between a source and a destination, since each router only has one processing node attached to it and a source processing node only generates traffic to a given destination. Thus, the path used by packets from a given source-destination pair is not shared with any other packets destined to another processing node. So, hybrid topologies and fat–trees are clearly the winners for this traffic pattern, and direct topologies are not a good option.

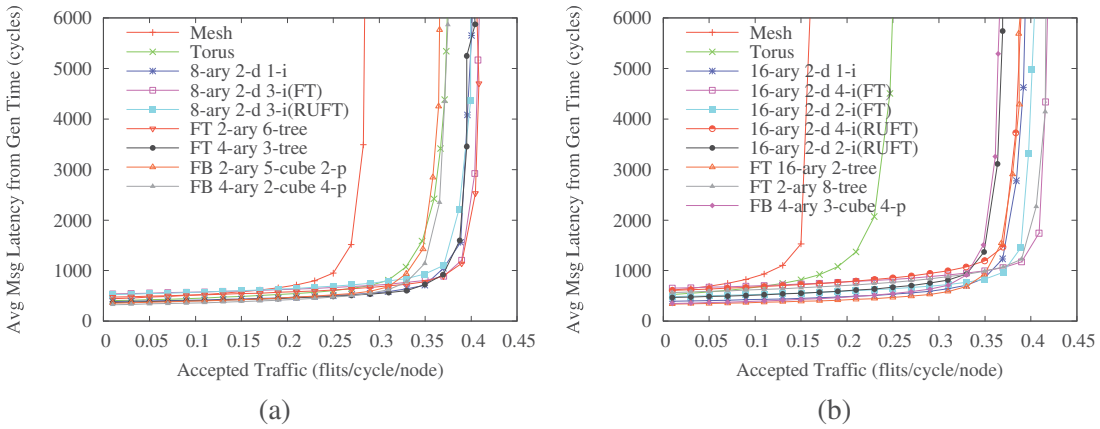


FIGURE 4.7: Average packet latency from generation vs. accepted traffic for Hot–Spot traffic at 5% in 2 dimensions. (a) 64 processing nodes. (c) 256 processing nodes.

Tornado traffic pattern: For this traffic pattern, shown in Figure 4.6, we obtain similar results than in complement traffic pattern. The KNS topologies and fat–trees are able to cope with all the injected traffic. Again, this very good behavior is due to the use of a load-balanced routing algorithm and the higher effective bisection bandwidth of these topologies.

Hot–spot traffic pattern: As expected, the concentration of packets sent to a few processing nodes makes the network saturate at a lower throughput than in other traffic patterns. As it can be seen in Figure 4.7, the worst topology is again the mesh; the torus saturates also slightly after it. The remaining topologies obtain a similar throughput, being the best one the hybrid topology that uses crossbars as subnets.

To summarize, the new family of topologies is able to obtain, in all analyzed traffic patterns and network configurations, equal or better performance results than the direct and indirect topologies evaluated for different traffic patterns.

However, each network topology has different complexity. In the next section, we estimate the complexity and cost of each network. Then, we will perform a comparison of topologies from a cost-performance point of view.

4.5.3 Cost–performance analysis

This section estimates and compares the hardware cost of each considered topology connecting the cost also with the perfomance of each of them. First, we will analyze, for each topology, the number of links and switches that it requires. However these numbers of links or switches

TABLE 4.3: Analytical comparison of the Mesh, Torus, Fat–Tree, Flattened–Butterfly and the KNS topologies. KNS topologies refers to the arity of indirect switches.

	Mesh	Torus	Fat–tree	Flattened–Butterfly
Switches	N	N	Nn_i/k_i	$k_f^{n_f}$
links	$(k_d-1)k_d^{n_d-1}n_d+N$	$k_d^{n_d}n_d+N$	$N(n_i-1)+N$	$k_f^{n_f}(k_f-1)n_f/2+N$

	KNS
Switches	$N/p_h(s_hn_h/k_i)$
Routers	N/p_h
links	$k_h^{n_h}n_h+(s_h-1)Nn_h/p_h+N$

in isolation is not a good metric, since a network with more complex switches (which are more expensive) has a lower number of links and switches and (incorrectly) seems to be cheaper. Therefore, neither the number of links nor the number of switches by themselves are accurate metrics of the actual cost. This is why we will provide another way to mesure the actual cost.

Table 4.3 shows how to compute the number of links and switches for each topology (it also shows the number of direct routers for the new family of topologies). For example, in k_h–ary n_h–direct 1–indirect (i.e. it uses crossbar as subnets), if we have N processing nodes and p_h concentrated processing nodes per each direct router, we will need N/p_h or, what is the same, $k_h^{n_h}$ direct routers and one switch (crossbar) for each group of k_h routers with the same dimension component. Then, we will need crossbars with k_h ports (in this case k_h is equal to k_i because a single switch is used in each dimension) to connect the k_h direct routers in each dimension, so we will need $N/p_h(n_h/k_h)$ crossbars.

If we use MINs as subnets (fat–tree or RUFT), we will need k_h/k_i switches per stage to implement the MIN that replaces the crossbar, yielding $N/p_h(n_h/k_h)(s_hk_h/k_i)$ switches with arity k_i, which can be simplified to $N/p_h(s_hn_h/k_i)$ switches.

Regarding links, we will need one link per dimension in each direct router (i.e. $k_h^{n_h}n_h$ links in total) and one link for each processing node to connect it to its corresponding router (N links in total). Furthermore, if we use MINs as subnets (fat–tree or RUFT), each one will have k_h links between each stage (i.e. $(s_h-1)k_h$ per MIN), yielding $(s_h-1)k_hN/p_h(n_h/k_h)$ in total, which can be simplified to $(s_h-1)Nn_h/p_h$ links. Notice that all links are bidirectional except those of the RUFT indirect subnets that use unidirectional links, and the same occurs with the switches. In the case of links, their cost is not halved, since much of the cost comes from the connectors. For this reason, and for easier comparison, we assume the same link cost for unidirectional and

TABLE 4.4: Results for different 2–D topologies with uniform traffic and 64K processing nodes.

Topology	Base Latency	Throughput	Links	Switches	Routers (KNS)
256–ary 2–direct 1–indirect	386	0.47	196,608	512	65,536
256–ary 2–direct 2–indirect(RUFT)	480	0.40	327,680	16,384	65,536
256–ary 2–direct 2–indirect(FT)	516	0.43	327,680	16,384	65,536
256–ary 2–direct 4–indirect(RUFT)	635	0.41	589,824	131,072	65,536
256–ary 2–direct 4–indirect(FT)	727	0.48	589,824	131,072	65,536
256–ary 2–direct 8–indirect(RUFT)	941	0.48	1,114,112	524,288	65,536
256–ary 2–direct 8–indirect(FT)	1,129	0.55	1,114,112	524,288	65,536
FB 16-ary 3-cube 16-p	367	0.39	157,696	4,096	0
FB 4-ary 7-cube 4-p	443	0.38	237,568	16,384	0
FB 2-ary 15-cube 2-p	512	0.41	311,296	32,768	0
FT 16-ary 4-tree	428	0.40	262,114	16,384	0
FT 4-ary 8-tree	596	0.41	524,288	131,072	0
FT 2-ary 16-tree	921	0.47	1,048,576	524,288	0
Torus	4,547	0.02	196,608	65,536	0
Mesh	6,478	0.01	196,096	65,536	0

bidirectional links. Regarding switches, a switch with p unidirectional ports can be implemented by using a switch with $p/2$ bidirectional ports.

Table 4.4 shows these metrics for different configurations of 64K–processing node topologies including also the performance results for the uniform traffic pattern. The network is 2D in the case of KNS, mesh, and torus topologies. For the FB topology, we have considered different configurations. As it can be seen, fat-trees and the KNS topologies with more stages are the topologies that achieve the highest raw throughput (256–ary 2–direct 8–indirect – FT and RUFT – and FT 2-ary 16-tree). However, if we also consider the cost, these topologies are composed of a higher number of links and switches. Specially in the case of the 256–ary 2–direct 8–indirect (FT and RUFT) and the FT 2-ary 16-tree. On the other hand, there are topologies that require a smaller number of switches, but these switches have more ports, so they may be more expensive. This is the case of the 256–ary 2–direct 1–indirect or FB 16-ary 3-cube 16-p.

TABLE 4.5: List price for switches: (a) Edge switches and (b) Chassis switches.

Edge Switches	
Ports	List price ($)
12	5,361
18	9,850
36	12,523

(a)

Chassis Switches	
Ports	List price ($)
108	32,650
216	48,778
324	65,875
648	110,177

(b)

TABLE 4.6: List price for links: (a) Copper Links and (b) Fiber Links.

Copper Links	
Length(m)	List price ($)
0.5	84
1	94
2	107
3	123
4	139
5	172

(a)

Fiber Links	
Length(m)	List price ($)
3	551
5	551
10	580
15	611
20	642
30	730
50	896
100	1,347

(b)

In order to get an actual cost figure, we have calculated the cost (in $) that some of these configurations would have when implemented with real commercial products. We have used InfiniBand products with FDR technology of Mellanox [68] (February 2015) to calculate the cost.

Tables 4.5.(a) and 4.5.(b) show the list price of switches depending on their number of ports. There are two different types of switches: edge and chassis switches. When preparing the budget, if there are no switches with the number of ports required by the configuration, we selected the next one with greater number of ports. For example, we needed to use 256-port switches for 256–ary 2–direct 1–indirect, so we selected 324-port switches.

Tables 4.6.(a) and 4.6.(b) show the list price of links depending on their length. As copper links are limited to 5 meters, if a longer link is required, fiber links must be used. We assumed an average length between cabinets (global links) of 10 meters, and 2 meters for connections in the same cabinet (local links).

The number of global and local links depends on the topology. For 256–ary 2–direct 1–indirect and 256–ary 2–direct 4–indirect with FT as subnets, the processing nodes of the same first dimension can be placed in the same cabinet or in two cabinets (one beside the other). The

TABLE 4.7: Cost for Network Interface Cards.

NIC	List price ($)
Connect IB PCIe 3.0 16x Single Port	1314
Connect IB PCIe 3.0 16x Dual Port	2378

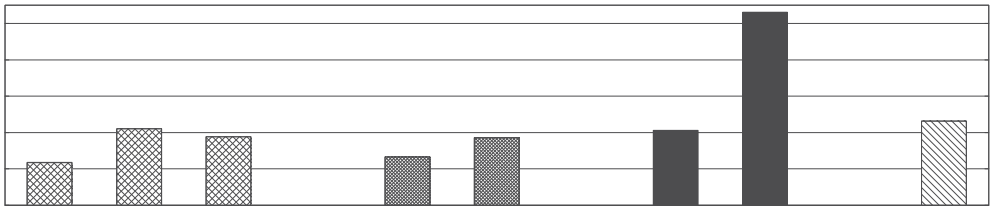


FIGURE 4.8: Total cost of different topology configurations with 64K processing nodes.

links of this dimension are local links, and the links of the second dimension are global links. Regarding the links interconnecting the stages of the MIN (a fat–tree in this case), we assume that they are local, since each subnet fits in a cabinet. In flattened–butterfly configurations, for example FB 16-ary 3-cube 16-p, we use the same approach. The links of the first dimension are local because the processing nodes of the same first dimension are placed in one or two cabinets and the links of the remaining dimensions are global. In fat–trees, the links which connect the processing nodes with the first stage are local, and the remaining links are global. In torus topology, the configuration is very similar to k_h–ary n_h–direct 1–indirect (local links for the first dimension and global links for the second dimension). However, in this case, a cabinet or a group of cabinets that contain processing nodes of the same first dimension, are connected to the neighboring cabinet or group. So, they will be very close. For this reason, in this case we have used shorter global links of 5 meters.

As previously stated, multiport NICs can be used to implement the KNS topologies. There are currently commercially available dual-port NICs that can may used to implement KNS with two dimensions. With these NICs, neither direct routers nor links between processing nodes and direct routers are longer required. The prices of dual–port and single–ports NICs are shown in Table 4.7.

Notice that all prices shown in Tables 4.5, 4.6 and 4.7 are for individual products. If purchased them massively we will surely enjoy a great discount in all cases.

TABLE 4.8: Cost–performance analysis for different topology configurations with 64K processing nodes. Throughput is measured in flits/cycle/node. Throughput/cost is measured in flits/cycle/node/$.

Topology	256-ary 2-direct 1-indirect	256-ary 2-direct 2-indirect (FT)	256-ary 2-direct 2-indirect (RUFT)	FB 16-ary 3-cube 16-p	FB 4-ary 7-cube 4-p	FT 16-ary 4-tree	FT 4-ary 8-tree	Torus 256-ary 3-cube
Throughput	0.47	0.43	0.40	0.39	0.38	0.40	0.41	0.02
Total Cost ($)	235 M	420 M	376 M	266 M	371 M	412 M	1,062 M	463 M
Throughput /Cost	2.00 x10^{-9}	1.02 x10^{-9}	1.06 x10^{-9}	1.47 x10^{-9}	1.02 x10^{-9}	0.97 x10^{-9}	0.39 x10^{-9}	0.04 x10^{-9}

Using these data, we calculated the cost of some selected configurations, which are shown in Figure 4.8. The configurations shown are the cheapest ones that their perfomance is not very far to the configuration that obtains the highest throughput.

As can be seen in Figure 4.8, the 256–ary 2–direct 1–indirect configuration obtains the lowest absolute cost. The fat–tree with 8 stages (FT 4-ary 8-tree)has a very high cost, despite having very good performance. In the case of torus, its cost is not very high, but it has a low performance. Flattened–butterfly has a competitive cost, but it is not lower than the 256–ary 2–direct 1–indirect and it does not reach a better perfomance.

To allow a better comparison of both cost and perfomance, Table 4.8 shows the ratio between cost and perfomance. If we use KNS topologies, configurations with more stages have better throughput but also a higher cost and more latency. The same applies to fat-trees. The k_h–ary n_h–direct 1–indirect combines a good perfomance with a low cost. Although the torus configuration is not very expensive, it has a very poor perfomance. Flattened–butterflies are not very expensive and obtain good perfomance. However, the k_h–ary n_h–direct 1–indirect configuration obtains the best absolute results in terms of perfomance-cost ratio. As it can be seen, the worst ratio is provided by the torus, followed by a configuration of the fat–tree (FT 4-ary 8-tree).

4.5.4 Fault–tolerance

The proposed topologies provide a lot of alternative paths for each source–destination pair, which is very important to tolerate faults. In this section, we will briefly analyze the fault-tolerance properties of the new topology.

In meshes, the worst case arises when a link connected to a corner node fails. As each corner node has a number of links equal to the number of network dimensions, the maximum number of faults that keeps the network connected is equal to the number of dimensions minus 1 ($n_d - 1$). The torus topology tolerates more faults than the mesh due to the fact that packets can move in both directions of a dimension (in particular, $2n_d - 1$ faults). The fat–tree topology tolerates as many faults as the number of up or down ports of the switches minus one ($k_i - 1$). In the FB, each router is connected to $(k_f - 1)n_f$ routers, so it tolerates as many faults as $(k_f - 1)n_f - 1$.

In the case of the KNS family of topologies and considering faults in the links connected to routers, as long as a router is still connected to one dimension, it may forward packets, providing that the indirect subnet associated to that dimension is working. Therefore, at least, the number of tolerated faults is given by the number of dimensions minus 1. If the faults occur in the links of the indirect network, as long as one subnet of every dimension is working, they will be also tolerated. Remember that there are k_h subnets per dimensions. Indeed, if fat-trees are used as the indirect subnets, several faults in each one of them are tolerated. Notice, though, that RUFT is not fault-tolerant, since there is a unique path for each source–destination pair, so it tolerates 0 faults. However, even when using RUFTs as indirect subnets, as long as other indirect subnets of the same dimension are working, the number of tolerated link faults in different indirect subnets should be higher than the one of the links connected to routers. As a consequence, we conclude that the fault tolerance degree of the new topology is upper bounded by the maximum number of faults in the routers, that is, the number of dimensions minus one, $n_h - 1$. This gives us the same fault tolerance as a mesh with the same number of dimensions.

Another analysis is also possible. Assume that we have routers with p ports available. With such a router, we could build a mesh with $\frac{p}{2}$ dimensions that tolerates $\frac{p}{2} - 1$ faults or a torus with also $\frac{p}{2}$ dimensions that tolerates $2\frac{p}{2} - 1 = p - 1$ faults. In the case of the KNS, we could built a p dimensional network that tolerates $p - 1$ faults. That is, for the same router degree, the KNS tolerates the same number of faults as a torus.

On the other hand, considering the rich connectivity of the newly proposed topology, a higher number of faults should be tolerated with a very high probability using a fault-tolerant routing algorithm or reconfiguration mechanism. Finally, it must be noticed that routing should be also changed to fully support fault tolerance in all topologies. However, an in depth analysis of both fault-tolerance probability and fault-tolerance routing issues is out of the scope of this paper.

We have not considered faults in injection links. Most topologies usually have a single link that connects the processing node to the network. If this link fails, the processing node will be isolated. Therefore, considering faults in the injection links, these networks will not tolerate any fault. However, in the KNS topology family (and also in tori and meshes), if the router is implemented inside each processing node (for instance using the HCA cards from [47]), the processing node is actually connected to the network through as many links as dimensions in the network, therefore tolerating also faults in the injection links.

4.6 Related Work

There are previous works that propose alternative topologies to the ones considered in this paper, but they have been never or seldom used in commercial products or in supercomputers. This is the case of the WK–recursive topology that was proposed in [9] for interconnection networks and more recently for on–chip networks [10], but, to the best of our knowledge, it has never been used in commercial products. Moreover, this topology has difficulties to guarantee deadlock freedom in the routing algorithm.

One of the topologies considered for comparison purposes in this paper is the flattened–butterfly[11] which is a very popular topology in recent papers. It is obtained from combining the routers in each row of a conventional butterfly MIN, thus obtaining an n–dimensional direct network where the nodes of each dimension are not connected in a ring fashion like in a torus, or through a small indirect topology like in our proposal; instead they are fully connected. In this paper, we have referred to n_f as the number of network dimensions and to k_f as the number of switches per dimension. This results in a topology very similar to a generalized hypercube but attaching several processing nodes to the same switch. Therefore, the flattened–butterfly, like the generalized hypercube, has a high cost, specially for large machines, which are the focus of our proposal. This topology is compared against our proposal in Section 4.5.

Other works propose the combination of several topologies, as we do in this paper. Most of them have been proposed for on–chip networks and therefore the target is different to ours. For example, in [17], each core is connected to two different tree networks in an on-chip environment in order to overcome the poor performance provided by trees. This proposal is not suitable for large machines due to the complexity of the cable layout and the poor performance achieved

even with two trees. Another proposal is the multi-ring topology [70], which is composed of several interconnected rings.

Many of the proposals for on–chip networks are based on hierarchical topologies. Subsets of cores are connected by small local networks connected in turn, by a global network. This is not the case of the family of topologies proposed in this paper. Hierarchical designs are expected to have a higher latency and smaller throughput, since both networks must be traversed for most of the source–destination pairs. In [12], the authors propose to use as local network a simple bus, and a mesh as global network. As can be expected, this is not an appropriate topology for a large machines due to the low performance provided by buses and meshes. The authors of [13] propose a tool to select the most suitable topology for a given network design. The tool explores the design space of hybrid Clos-torus networks. However, opposite to our proposal, the explored designs are hierarchical topologies, where local networks are Clos networks and they are connected by a global torus network. Another hierarchical topology is the DragonFly[14, 15], which provides a topology that is based on grouping routers virtual routers to increase the effective radix of the network. This topology uses two different networks, one intra–group, and another one inter–group. For this topology, it is advisable to use a non-minimal global adaptive routing to balance the load across the global channels. These global channels, which link different groups, are long links, so that a high latency can be expected.

A topology closer to the new family proposed in this paper is the mesh of trees (MoT) introduced in [18] and later used also in NoCs [19]. It is based on an $n-$dimensional topology where the nodes of a given dimension are connected using a tree. This results in a particular case of our proposed family of topologies with very poor expected performance due to using a simple tree for connecting the nodes of a dimension. The Bcube [20] and Hypercrossbar network [21] also resemble our proposal. Each processing node is connected to several dimensions by using several NICs. A switch is used to connect the processing nodes of the same dimension. However, routing is performed through end-processing nodes, by ejecting packets from the network through a NIC and later reinjecting them through another one.

Finally, in [16] a topology that combines several tori networks is proposed. The proposal focus on large supercomputers, but its applicability is limited to 2^{16} processing nodes and its wiring layout is complex for large machines. The proposal starts from a 2-D torus and provides bypass links in the diagonal direction as many times as needed. This proposal does not improve the number of hops in a single dimension, since no new links are added to connect nodes of the same

dimension, but reduces the number of hops when traversing several dimensions. In addition, this topology has the same problem with deadlock-free routing than the WK–recursive.

4.7 Conclusions

This paper proposes a new family of hybrid topologies, the KNS, for large–scale interconnection networks. It is based on an n–dimensional topology where the nodes of each dimension are connected through a crossbar or a small indirect topology (a fat–tree or a RUFT in this paper). This results in a new family of topologies that provides high–performance, with latency and throughput figures of merit close to the ones obtained with indirect topologies, but with a much lower hardware cost. In particular, from the throughput point of view, the new topologies with fat–trees as indirect subnet are the best ones. Nevertheless, from the cost–performance point of view, the new topologies with crossbars as indirect subnets are the winners, as they are able to obtain better throughput per dollar compared to indirect topologies, and the differences are even higher when comparing to direct topologies. Moreover, in the new topologies with MIN's as indirect subnets, as the indirect subnets are small, the layout of the new topologies is much simpler than the one for indirect topologies with the same number of processing nodes. Concerning fault–tolerance, the proposed family of topologies is able to tolerate, at least, the same number of faults as a mesh with the same number of dimensions, regardless the topology used in the indirect subnets.

Chapter 5

A New Fault-Tolerant Routing Methodology for KNS Topologies

Authors: Roberto Peñaranda (Universidad Politécnica de Valencia), Ernst Gunnar Gran, Tor Skeie (Simula Research Laboratory, Norway), María Engracia Gómez, Pedro López (Universidad Politécnica de Valencia).

Type: Conference.

Conference: The 2nd IEEE International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB).

Location: Barcelona, Spain.

Year: 2016.

DOI: $http://dx.doi.org/10.1109/HIPINEB.2016.9$

URL: $http://ieeexplore.ieee.org/document/7457761/$

Abstract

Exascale computing systems are being built with thousands of nodes. A key component of these systems is the interconnection network. The high number of components significantly increases the probability of failure. If failures occur in the interconnection network, they may isolate a large fraction of the machine. For this reason, an efficient fault-tolerant mechanism is needed to keep the system interconnected, even in the presence of faults. A recently proposed topology for these large systems is the hybrid KNS family that provides supreme performance and connectivity at a reduced hardware cost. This paper present a fault-tolerant routing methodology for the KNS topology that degrades performance gracefully in the presence of faults and tolerates a reasonably large number of faults without disabling any healthy node. In order to tolerate network failures, the methodology uses a simple mechanism: for some source-destination pairs, only if necessary, packets are forwarded to the destination node through a set of intermediate nodes (without being ejected from the network) which allow avoiding faults. The evaluation results shows that the methodology tolerates a large number of faults. Furthermore, the methodology offers a gracious performance degradation. For instance, performance degrades only 1% for a 2D-network with 1024 nodes and 1% faulty links.

5.1 Introduction

Interconnection networks are used for different purposes, from small devices that use networks on chip which connect several components to large supercomputers that connect a large number of nodes. The size of large supercomputers has been growing year after year. The topmost machines of the top 500 supercomputer list [1] are being built up by using hundreds of thousands of processing nodes. All these processing nodes work jointly to solve a given problem in as short a time as possible.

The high amount of hardware that can be found in an interconnection network in high–performance machines significantly impacts the probability of having a fault in the system. Each component may independently fail, and therefore, the probability of having a single fault in the whole system drastically raises with the number of elements that compose it. For this reason, it is important that systems can keep running although there are several failures on the network.

One possible solution is to replicate all network elements, and use these additional elements as spare components. But this significantly increases the cost of the network. On the other hand, there is another solution which focuses on modifying the routing algorithm to be able to reach the destination nodes using alternative paths circumventing failures.

In this paper we present a new routing algorithm for KNS networks, a recently proposed hybrid topology, that is able to tolerate multiple faults and where the performance degrades gracefully. The algorithm is based on the use of intermediate nodes [46] in tori and meshes, where an intermediate node is used for some source-destination pairs in order to avoid faults in the packet paths.

The rest of the paper is organized as follows: Section 5.2 describes briefly different fault tolerant algorithms proposed previously for other topologies. In Section 5.3, we describe KNS, the hybrid topology for which this mechanism has been implemented. In Section 5.4, we present the fault-tolerant methodology proposed in this paper that is based on using intermediate nodes. In Section 5.5, different configurations of the new routing algorithm are evaluated. Finally, in Section 5.6, some conclusions are drawn.

5.2 Related Work

There are two different categories of fault tolerant techniques that are based on routing configuration. The first category reconfigures the routing tables when a failure occurs. In this case, we have to update the routing tables with the new topology after the failure [27–30]. This technique allows the network to tolerate any number of faults without requiring extra resources [31], as long as the network is still connected, thanks to its flexibility. But this flexibility may kill performance due to the need of using topology agnostic routing algorithms as the resulting network topology is irregular. That is, they do not consider the specific characteristics of the topology, thus they often provide inferior traffic balance.

On the other hand, the second category covers fault-tolerant routing algorithms. A large number of fault-tolerant algorithms for interconnection networks have been proposed, and specially for direct network topologies like tori and meshes. Some of them often require many resources (virtual channels), sometimes depending on the number of tolerated faults [35] or the number of dimensions of the topology [36]. Other algorithms are based on disabling fault regions [37–41] or individual nodes [42–44] to route the packets around these fault regions. However, to do this,

these algorithms disable healthy nodes. In [46], the authors use the technique of Valiant routing [45] to implement an algorithm that uses intermediate nodes to avoid faults. This methodology requires a few virtual channels and does not disable healthy nodes. There are other algorithms that do not require extra virtual channels like [32, 33] or [34]. The first two methods are able to tolerate only a few faults for low dimension meshes, while the third method can offer more tolerance against faults, but needs extra virtual channels in a torus. However, these last algorithms were designed specifically for meshes and tori, and they provide a bad traffic balance as a lot of traffic is directed towards a single link which can be easily saturated.

We have focused on direct topologies because of the similarity with the KNS topology.

5.3 Preliminaries

Topologies usually adopt a regular structure to simplify their implementation and the routing algorithm. Among the different taxonomies of regular topologies, the most commonly-used one divides them into direct and indirect topologies [2, 3].

Direct topologies usually adopt an orthogonal structure where nodes are organized in an n–dimensional space, and each processing node has an associated router. The nodes are connected in each dimension in a ring or array fashion. 2D or 3D direct topologies are relatively easy to built as each topology dimension is mapped to a physical dimension. Implementing direct topologies with more than three dimensions implies not only increasing its wiring complexity but also the length of its links when they are mapped to the 3D physical space. Indeed, the number of ports of the routers geometrically grows with the number of dimensions (as two ports per dimension are required). The implementation limitation in the number of dimensions leads to an increase in the number of nodes per dimension, which increases the communication latency, negatively impacting performance.

The alternative is to use an indirect topology. The main difference, compared to direct topologies, is that not all the routers have an associated processing node. The most common indirect topologies are multistage indirect networks (MINs) where switches are organized in a set of n stages. Indirect topologies provide better performance for a large number of nodes than direct ones. However, this is achieved by using a higher amount of switches and links. Furthermore, their physical implementation is complex due to the fact that the wiring complexity grows with

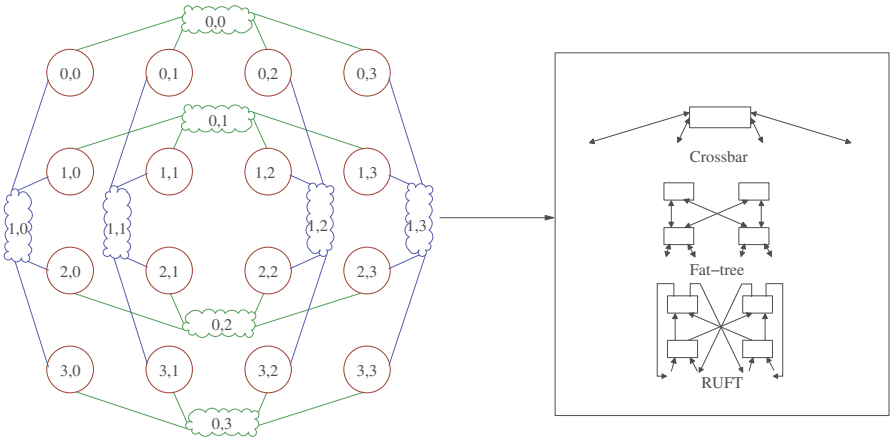


FIGURE 5.1: An example of the KNS topology with $n = 2$ and $k = 4$.

the number of nodes in the system, unlike direct topologies where complexity grows with the number of topology dimensions.

To overcome the limitations of direct and indirect topologies, hierarchical and hybrid topologies have been proposed. In [11], the authors propose a variation of the butterfly topology obtained from using high-radix switches, that results in a direct topology. This topology can be seen as a generalized hypercube with concentration, as all the nodes in the same dimension are directly connected. That is, there is a link from each node to the others of the same dimension. As an extension of this topology, the Dragonfly topology was proposed in [14], which provides a hierarchical topology that is based on grouping routers in virtual routers to increase the effective radix of the network. This topology uses two different networks, one intra–group, and one inter–group. For this topology, it is advisable to use a non-minimal global adaptive routing to balance the load across the global channels. These global channels, which link different groups, are long links, so that a high latency can be expected.

To solve the limitations of previous topologies, the k-ary n-direct s-indirect (KNS) was proposed in [67], an n–dimensional topology, where the rings that connect the nodes in each dimension are replaced by small indirect networks. In this way, communication latency along each dimension no longer linearly grows with the number of nodes per dimension. On the other hand, the small size of this indirect topology allows a reasonable wiring complexity opposite to large indirect topologies. This combination results in a family of topologies that provides high performance, with latency and throughput figures of merit close to the ones obtained with indirect topologies, but at a reduced hardware cost. This topology is defined by three parameters: the number of

dimensions n, the number of nodes per dimension k, and the number of stages of the indirect subnetworks s.

In Figure 5.1 we can see an example of this topology. The topology has 2 dimensions and 4 nodes per dimension. We can use different solutions to connect the nodes of the same dimension. For example, we can use a crossbar or a single switch to connect them or different multistage topologies like a k-ary n-tree or a RUFT topology [5] if we have a higher number of nodes per dimension. The number of processing nodes is given by $N = k^n$. In this paper, we focused in k-ary n-direct 1-indirect topologies, although the proposed algorithm can be implemented in any KNS topology configuration.

5.4 Fault-Tolerant Routing Methodology

In this paper, we will assume a KNS topology using crossbars as indirect subnetworks with minimal deterministic routing, Hybrid-DOR [67]. If we have no failure, packets are routed using this minimal routing algorithm that does not need virtual channels.

We will consider link faults, since a switch fault can be modeled like a switch with failures in all its links. We assume that if a channel fails then the link fails in both directions. In this paper, we do not focus on how the failure information propagates to the other nodes. We assume a static fault model where the whole network has in advance the information about the failures of the network.

For each source-destination pair without failures in their path, we route packets with Hybrid-DOR using minimal paths. But, if there is any fault in the path of any source-destination pair, the methodology uses intermediate nodes, like in [46]. The use of intermediate nodes was proposed in [45] for other purposes, such as traffic balancing. So, the routing algorithm avoids these faults by sending the packet to one or several intermediate nodes and from the last intermediate node to the destination node. The Hybrid-DOR algorithm is used in all subpaths. Notice that the packets are not ejected from the network when they reach intermediate nodes. The idea is to avoid the faults by deviating the packet to an intermediate node. The intermediate node for each source-destination pair is selected with this purpose. If there is not any faulty link in the path between the source and the destination, then no intermediate node is used.

The address of intermediate nodes are stored in the packets, in addition to the address of the destination node. When the intermediate node is reached, its address is removed from the packet. This procedure is repeated for each intermediate node, if more than one is used. Using more than one intermediate node allows to tolerated more faults.

For each source–destination pair, it is checked whether it exists a fault-free path. If routing through intermediate nodes is required, they are computed and stored in a table at every source node. There is a table entry for each destination node that requires routing through intermediate nodes.

Like in [46], we will denote the source node as S and the destination node as D. For each intermediate node, we will use the nomenclature I_x, where x represents the index of the intermediate node (I_1 for the first one, I_2 for the second one and so on).

To ensure deadlock freedom we need at least as many additional virtual channels as intermediate nodes. For example, if we use up to two intermediate nodes, we need at least three virtual channels. When a packet reaches an intermediate node, the packet changes the virtual channel to avoid deadlocks. So, following the example with two intermediate nodes, one virtual channel is used from S to I_1, another one from I_1 to I_2, and the last one from I_2 to D. In this way, we avoid the occurrence of deadlocks, because we are dividing the network into three virtual networks, and we use a deadlock-free routing algorithm within each virtual network. We could also use an adaptive routing algorithm, using several adaptive channels and an escape channel [56], but in such a case, we would need one escape channel for each virtual network to ensure deadlock freedom, where the escape channels use Hybrid-DOR. In this paper, we focus on the deterministic routing algorithm.

In this section, we do not consider the cases where a set of faults physically disconnect one or more nodes of the network, since in this case the packet would be unable to reach its destination. Because this methodology does not add new resources. Therefore, these cases, where the whole network is not connected, are not considered in this section.

Next, we will present how to select the intermediate nodes. First, we will focus on using only one intermediate node. After that, we show how to extend the methodology to use multiple intermediate nodes.

5.4.1 One Intermediate Node

In this case, we will use only one intermediate node when one or more faults affect the minimal path provided by Hybrid-DOR between a pair of nodes. This intermediate node, I_1, has to satisfy two rules:

1. I_1 is reachable from S.
2. D is reachable from I_1.

A node Y is reachable from X when there is a minimal path provided by Hybrid-DOR between this pair of nodes, and there is no fault in it.

For direct topologies like tori or meshes, the choice of this intermediate node is very important, because the number of hops can greatly increase, depending on the selected intermediate node. In KNS, the number of hops depends on the number of dimensions that the packet must cross. The methodology prioritizes the use of those intermediate nodes which allow the packet to reach the destination node without additional hops compared with minimal path between the source and the destination.

For instance, in Figure 5.2.(A) there is an example of a 4-ary 2-direct 1-indirect network where there is a fault at the source node (S) in the X dimension link. So, it cannot reach the destination node (D) using Hybrid-DOR. In this case, the possible intermediate nodes are all nodes of the same column (surrounded by the dashed line in the figure), but the best choice is the node that is in the same row as the destination node, because it allows to reach the destination node by a minimal path.

Lemma 5.1. *Given a KNS network with n dimensions, using one intermediate node, the routing algorithm can tolerate $n - 1$ failures.*

Proof. For a network with n dimensions, each node has n links. Each link allows connecting to the other nodes of the same dimension by a indirect subnetwork. Let T_{RS} be the set of nodes reachable from S using Hybrid-DOR, and T_D the set of nodes from which D is reachable using Hybrid-DOR. As long as $T_{RS} \cap T_D$ is not empty for any source-destination pair, the network is able to handle the fault combination. Therefore, the worst fault combinations are the ones that reduce the number of nodes in T_{RS} or/and T_D.

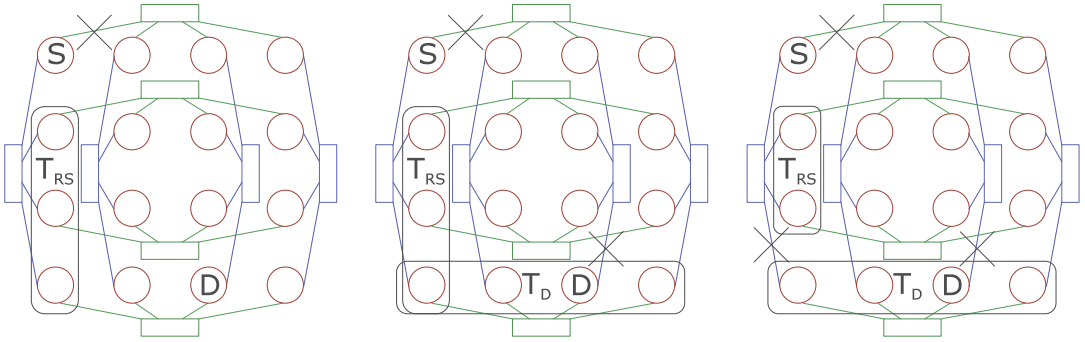


FIGURE 5.2: An example of a set of faults that disconnect the network with a KNS topology with $n = 2$ and $k = 4$.

For instance, in Figure 5.2.(A), while T_D set comprises all network nodes (except the source node), the fault reduces T_{RS} to only 3 nodes. $T_{RS} \cap T_D$ cardinal is 3 and the fault is supported. However, assume that there is another link failure that reduces T_D. The worst scenario occurs when the fault is located on the Y dimension link of D (Figure 5.2.(B)). Thus, $T_{RS} \cap T_D$ is reduced to one node, and the fault is supported. However, if a new failure appears in the links of the intermediate node (thus, there will be 3 faults), the source–destination pair S–D will become disconnected. This can be seen in Figure 5.2.(C).

However, the worst scenario happens when the source and the destination nodes are located in the same row. In this case, with only two faults (the first two faults of the example, a link failure in X link of the source node and another one in the Y link of destination node) the destination node is not longer reachable from the source node and $T_{RS} \cap T_D$ is empty (see Figure 5.3). Thus, the network supports one fault. Notice, though, that there are some cases were 2 faults physically disconnects nodes (i.e. a failure in all links of a node) and it is impossible to reach them. These cases are obviously unsupported.

In general, for any n-dimensional KNS network, the minimal non tolerated scenario happens when the source and the destination nodes have the same coordinates but the coordinate of the first dimension. If T_D is reduced due to faults on all links of destination node but the one of the first dimension (which physically disconnect the node) and there is a fault on the link of the first dimension on the source node, both nodes become disconnected.

Therefore, the minimal number of faults needed to disconnect the network with only one intermediate node is n faults ($n - 1$ failures at the destination node links plus one failure at the

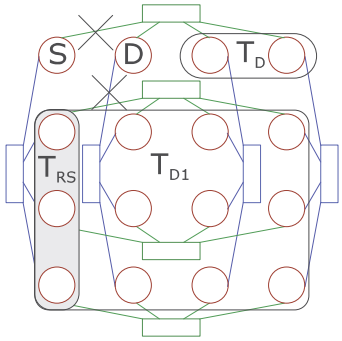


FIGURE 5.3: An example of the KNS topology with $n = 2$ and $k = 4$ and 2 faults.

source). Thus, the routing algorithm is able to tolerate $n-1$ failures. This means that the routing algorithm is able to tolerate all the fault combinations with less or equal number of faults. □

5.4.2 Multiple Intermediate Nodes

There are cases where only one intermediate node is not enough to handle the faults. In these cases, the routing algorithm can use more than one intermediate node. Assume that a number of x intermediate nodes $I_1, I_2, \ldots I_x$ are required. Intermediate nodes are selected according to the following rules:

1. I_1 is reachable from S.
1. I_{i+1} is reachable from I_i, for $0 < i < x$.
2. D is reachable from I_x.

Therefore, we can guarantee that the packet is able to reach its destination node following the path S-I_1-...-I_i-I_{i+1}-...-I_x-D.

In order to reduce the number of hops we try to use a set of intermediate nodes that does not increase the number of hops beyond a minimal path. In particular, if there are available non-minimal paths using i intermediate nodes and also a minimal path using j intermediate nodes, where $i < j$, the later will be finally selected.

Lemma 5.2. *Given a KNS network with $n > 2$ dimensions, using two intermediate nodes, the routing algorithm can tolerate $2 * (n-1) + k - 3$ failures. If $n = 2$, the routing algorithm can tolerate $2 * k - 1$ faults.*

Proof. Let T_{D1} be the set of nodes from which the destination node is reachable using Hybrid-DOR using one intermediate node. That is, the set of nodes that can reach the destination node using an intermediate node from T_D. In Figure 5.3 we can see an example of a 4-ary 2-direct 1-indirect network with 2 faults. The first one is located at the source node X-link and the second one at the destination node Y-link. The Figure shows the set of reachable nodes from the source node (T_{RS} with gray background), the set of nodes which can reach the destination node (T_D), and the set of nodes that can reach the destination node using any node within T_D as an intermediate node (T_{D1}). Thus, in this case, we have to use one node of $T_{RS} \cap T_{D1}$ as the first intermediate node, and another one of T_D as the second intermediate node. As long as $T_{RS} \cap T_{D1}$ is not empty for any source-destination pair, the network is able to handle the fault combination. Therefore, the worst fault combinations are these that reduce the number of nodes in T_{RS}, T_D or/and T_{D1}.

As in Lemma 1, if there are faults on all the destination node links except the one of the first dimension to avoid disconnecting the network, T_D will be reduced in $k-1$ nodes, the nodes which share all coordinates with the destination node, except the coordinate of the first dimension. On the other hand, if some faults appear on all links of the source node, except the one of the last dimension, T_{RS} will be reduced in $k-1$ nodes, the nodes which share all coordinates with the source node except the coordinate of the last dimension. In this way, the possible second intermediate nodes, T_D, are only reached by the link of the last dimension. But if we assume that these links are also faulty, the destination node will not be not longer reachable. So, we only need $n-1$ faults in the source node links, $n-1$ faults in the destination node links and $k-1$ faults in the set of possible second intermediate nodes. However, a still worst scenario is when the source node shares all coordinates with the destination node, except the coordinate of the first dimension (see Figure 5.3). In this case, the destination node will not be reachable with only $k-2$ faulty links in T_D set. Hence, the source–destination pair becomes disconnected with $2*(n-1)+k-2$ faults, ergo, the network can tolerate $2*(n-1)+k-3$ faults. This means that the routing algorithm is able to tolerate all the fault combinations with less or equal number of faults.

Notice, though, that for 2-dimensional networks, adding faults for every possible second intermediate node (which comprise T_D) physically disconnects the row where the destination node is located, as we can see in Figure 5.3. These cases are unsupported by the methodology.

For these kind of topologies, another combination of faults is needed. For instance, keeping

the faults of destination node ($n - 1 = 1$) and source node ($n - 1 = 1$), and setting faults on the X-links of nodes of T_{RS} except one of them ($k - 2$, see Figure 5.3), to avoid physically disconnecting the column, we have only one node in $T_{RS} \cap T_{D1}$ which will be the first intermediate node. From this first intermediate node there are $k - 2$ possible paths to the destination, i.e., one path for each possible second intermediate node which comprises T_D. If there are faults in these paths, the destination node will not be reachable by the source node, i.e., with $1 + 1 + (k - 2) + (k - 2) = 2 * k - 2$ faults. Thus, the routing algorithm is able to tolerate $2 * k - 1$ faults in a 2-D KNS network. □

5.4.3 Extension To Other Indirect Subnetworks

In this paper, we have focused on KNS topologies that use crossbars as indirect subnetworks. However, we can extend the methodology to KNS topologies that use other indirect subnetworks like fat-trees or RUFT. To do this, a specifically designed methodology should be also used to tolerate faults in each indirect subnetwork. Intermediate nodes are used globally and the specific methodology locally for each subnetwork.

While the subnetworks can avoid faults, the direct routers will work normally. However, if a node becomes unreachable due to a fault at a given subnetwork, it will be modeled like a fault in this node, in the link of the corresponding dimension of this subnetwork.

5.5 Experimental Evaluation

To evaluate this methodology, we have divided this section in two parts. First, we analyze the number of failures that can be tolerated. A fault-tolerant routing algorithm is able to tolerate n failures if it can provide a valid path between every source-destination pair with any combination of n failures. There are situations where the failures physically disconnect the network. We consider these situations as combinations where there is no path for all source-destination pairs.

Second, we evaluate the network performance using this methodology with different number of faults. To do this, we have simulated different network configurations with different number of faults under uniform traffic. For each number of faults, we have tested 50 random fault combinations to obtain the average throughput and latency. In this experiments, the situations where some nodes are physically disconnected were not taken into account either.

We have analyzed this methodology for two network configurations: a 32-ary 2-direct 1-indirect and a 10-ary 3-direct 1-indirect topologies. They have similar number of nodes (1024 and and 1000, respectively), so we can analyze the impact of having a different number of dimensions.

5.5.1 Simulation Model

To perform the simulations, we have used an event-driven simulator which models KNS topologies with bidirectional links. This simulator uses virtual cut–through switching. Each switch has a full crossbar with queues of 4 packets both at their input and output ports. Credits are used to implement the flow control mechanism. Packet length is 16-flit. We assume a pipelined router with a latency of 4 clock cycles, and switch and link bandwidth is assumed to be one flit per clock cycle.

5.5.2 Fault Analysis

The number of possible fault combinations increases exponentially with the number of faults. For this reason, it is not possible to explore all possible fault combinations in a reasonable amount of time. Therefore, we have used as a tool the statistical analysis.

Specifically, we have analyzed a subset of the fault combinations, where the faults are randomly chosen. This subset is large enough to obtain results with a confidence level of 99% and an error lower than 1%.

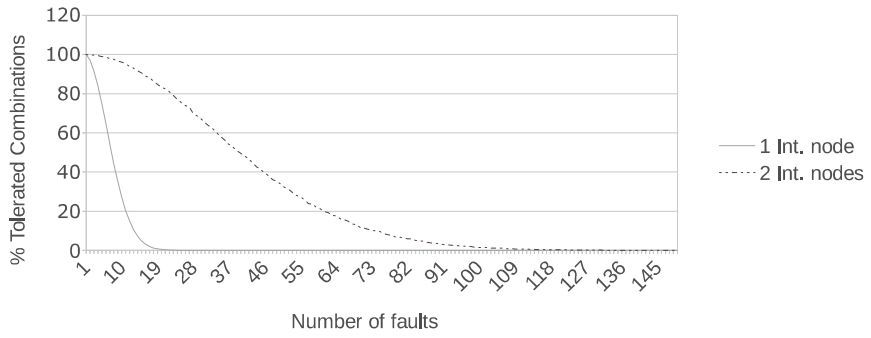


FIGURE 5.4: Supported fault combinations by the methodology when using one or two intermediate nodes in a 2D-network with 1024 nodes.

Figures 5.4 and 5.5 show the percentage of supported combinations using one or two intermediate nodes for a 2D-network with 1024 nodes and a 3D-network with 1000 nodes, respectively.

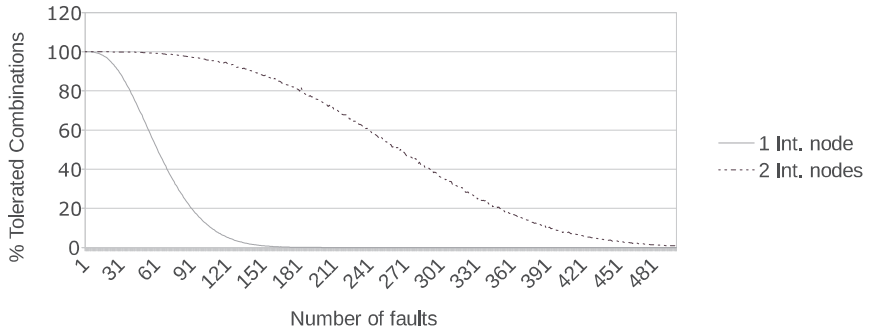


FIGURE 5.5: Supported fault combinations by the methodology when using one or two intermediate nodes in a 3D-network with 1000 nodes.

These figures show the results from 1 fault to 150 for 2D-networks and 500 for 3D-networks. First, as expected, the percentage of supported combinations of faults strongly increases when using two intermediate nodes versus only one. On the other hand, in the 2D-network, the percentage of tolerated combinations of faults decreases considerably with a relatively number of faults if we compare to the 3D-network. However, the 3D-network has more resources. There are 3000 links in the 3D-network versus 2048 links in the 2D-network. In addition, the fact of having more dimensions gives more possibilities to route the packet through different paths that do not share resources, being able to avoid more faults. Therefore, we can see that, with 23 faults in the 2D-network even with two intermediate nodes, the percentage of supported combinations is less than 80%. However, in the 3D-network, more than 100 faults are needed to reach a percentage less than 80%. Remember that the fault combinations that physically disconnect the network are included in this analysis. For instance, in 2D-networks, there are unsupported fault combinations with only 2 faults (or 3 in 3D-networks).

5.5.3 Performance Analysis

In this Section we will analyze the behavior of the fault-tolerant routing methodology in presence of faults. To do this we have simulated several network scenarios, with 1% faulty links, 3% faulty links and 5% faulty links. For each fault scenario, we have generated 50 random fault sets, all of them supported by the methodology. We have used the methodology using 2 intermediate nodes, since this allows to test fault combinations with more faults. However, the fact of using two rather than one intermediate node does not strongly impact network performance. This is because the methodology is able to avoid the fault using only one intermediate node in many

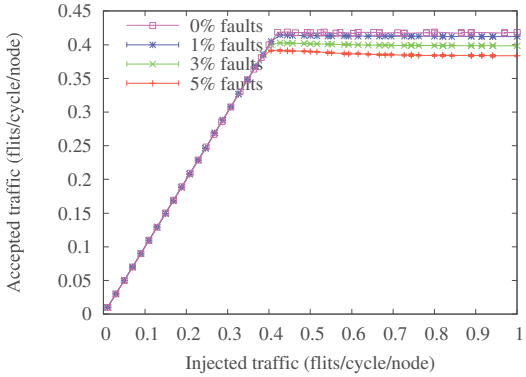


FIGURE 5.6: Accepted traffic versus injected traffic for a 32-ary 2-direct 1-indirect under uniform traffic.

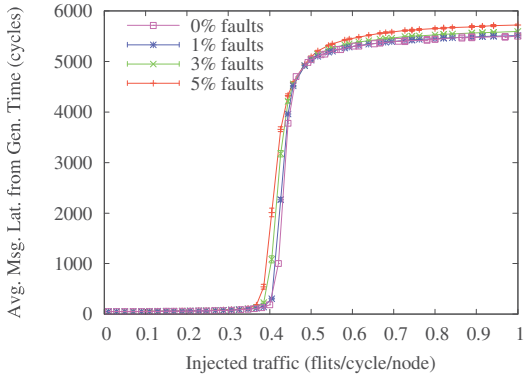


FIGURE 5.7: Average latency versus injected traffic for a 32-ary 2-direct 1-indirect under uniform traffic.

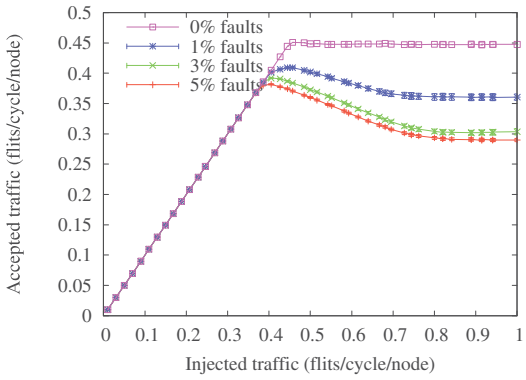


FIGURE 5.8: Accepted traffic versus injected traffic for a 10-ary 3-direct 1-indirect under uniform traffic.

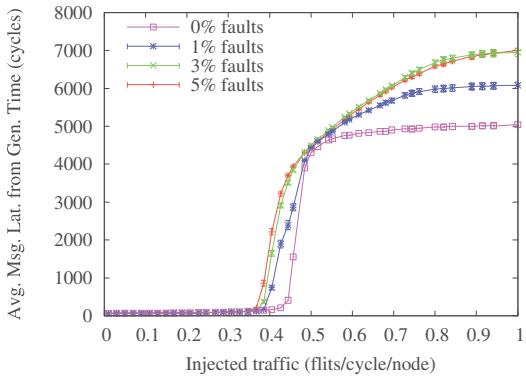


FIGURE 5.9: Average latency versus injected traffic for a 10-ary 3-direct 1-indirect under uniform traffic.

cases, although it is able to use two. Therefore, the methodology only uses 2 intermediate nodes in a few cases, which does not impact the performance to a large extent.

In order to measure the performance, we obtained the accepted traffic and average message latency versus injected traffic. Accepted traffic is measured as the amount of data per node and per time that the network can accept (flits/cycle/node). Network throughput is the peak value of accepted traffic. Average message latency is measured as the mean of the elapsed from message injection into the network at the source node until its ejection at the destination node.

We can see the results for a 32-ary 2-direct 1-indirect network under uniform traffic in Figures 5.6 and 5.7. In this case, we have a degradation of about 1% in throughput with 1% faulty links (21 links) compared to the same configuration without faults. For 3% and 5% faulty links

performance degradation increases to 3,8% and 6,5%, respectively. Latency is affected as well, increasing the average value with respect to the fault-free. In particular, at saturation, latency is increased by 67% with 1% faulty links.

In the case of using a 10-ary 3-direct 1-indirect network (Figures 5.8 and 5.9) we can see that the degradation, with the same percentage of faulty links, is higher In particular, network throughput degrades by about 9% with 1% links with faults (30 faults) compared to the fault-free network, and by 13% and 15% with 3% and 5% faulty links, respectively. The increase in latency, for the 1% link faults, is 1.8 times at the saturation point. In part, this is because this network has more or less the same number of nodes, but more dimensions, so more links than the 2-dimensional configuration and, therefore, a higher number of absolute link faults for the same fault percentage. Therefore, mode more paths are affected. On the other hand, as shown in Figures 5.4 and 5.5, the fact of having a higher number of dimensions with the same number of nodes improves the probability of avoiding a given fault combination. Therefore, although throughput is degraded 3.8% with 3% faulty links in the 2D-network against a degradation of 13% with 3% faulty links in the 3D-network, the of supporting a combination with this number of faults in the 3D-network is about 97%, against only 16% in the 2D-network.

To summarize, the proposed methodology offers a better avoidance of faults in networks with more dimensions, but this does not mean a better performance, because it depends on the fault combination and the selected intermediate nodes. With more dimensions, the selected intermediate nodes can increase the number of hops to a greater extent, because the distance between two nodes in a KNS topology increases with the number of dimensions.

5.6 Conclusions

We have proposed a fault-tolerant routing algorithm for k-ary n-direct 1-indirect topologies. This algorithm is based on using intermediate nodes and assumes a static fault model. It can tolerate a large number of faults without suffering a great fall in performance. This algorithm does not disable any healthy node, unlike other algorithms, and does not require too many resources. The algorithm only requires one extra virtual channel per intermediate node. This algorithm has been evaluated by simulation under uniform traffic, and the results show that the performance only suffers a small degradation. For instance, using only two intermediate nodes (2 extra virtual channels), the results show a degradation in performance of 1% for a 2D-network

with 1024 nodes, 1% faulty links (21 faults). The proposed methodology can be easily extend to other configurations of the KNS network topology.

Chapter 6

A Fault-Tolerant Routing Strategy for KNS Topologies Based on Intermediate Nodes

Authors: Roberto Peñaranda, María Engracia Gómez, Pedro López (Universidad Politécnica de Valencia), Ernst Gunnar Gran, Tor Skeie (Simula Research Laboratory, Norway).

Type: Journal.

Journal: Journal of Concurrency and Computation: Practice and Experience.

Publisher: Wiley.

ISSN: 1532-0634.

State: Accepted.

Impact Factor: 0.942

JRC ranking: Q2

Abstract

Exascale computing systems are being built with thousands of nodes. The high number of components of these systems significantly increases the probability of failure. A key component for them is the interconnection network. If failures occur in the interconnection network, they may isolate a large fraction of the machine. For this reason, an efficient fault-tolerant mechanism is needed to keep the system interconnected, even in the presence of faults. A recently proposed topology for these large systems is the hybrid k-ary n-direct s-indirect (KNS) family that provides optimal performance and connectivity at a reduced hardware cost. This paper presents a fault-tolerant routing methodology for the KNS topology that degrades performance gracefully in presence of faults and tolerates a large number of faults without disabling any healthy computing node. In order to tolerate network failures, the methodology uses a simple mechanism. For any source-destination pair, if necessary, packets are forwarded to the destination node through a set of intermediate nodes (without being ejected from the network) with the aim of circumventing faults. The evaluation results shows that the proposed methodology tolerates a large number of faults. For instance, it is able to tolerate more than 99.5% of fault combinations when there are ten faults in a 3-D network with 1,000 nodes using only one intermediate node and more than 99.98% if two intermediate nodes are used. Furthermore, the methodology offers a gracious performance degradation. As an example, performance degrades only by 1% for a 2-D network with 1,024 nodes and 1% faulty links.

6.1 Introduction

The size of large supercomputers has been growing year after year. The topmost machines of the top 500 supercomputer list [1] are being built up by hundreds of thousands of processing nodes. For instance, the current (June 2016) number one has 10,649,600 computing cores comprising 40,960 nodes. All these processing nodes work jointly to solve a given problem as fast as possible. A high-speed interconnect among nodes allows running processes to communicate.

The large amount of hardware that can be found in the interconnection network of large high–performance machines significantly impacts the probability of having a fault in the system. Each component may independently fail, and therefore, the probability of having a single fault in the whole system drastically raises with the number of elements that compose it. Therefore, it is

extremely important that systems can keep running despite the presence of several failures in the network.

A trivial alternative that has been followed in some proposals is to replicate network components, and use these additional elements as spare parts. The main problem of this alternative is that it significantly increases the cost of the network. Taking into account that interconnection network topologies usually provide different alternative paths between source–destination node pairs, it is possible to modify the routing algorithm to circumvent failures. Hence, destination nodes can be reached using alternative paths not affected by failures.

In [71] we presented a routing algorithm for the recently proposed KNS topology [67] that is able to tolerate multiple faults with minimal performance degradation in presence of failures. In this paper, we extend this previous work, providing a more detailed description of the mechanism, including implementation details and presenting new evaluation results.

The rest of the paper is organized as follows. Section 6.2 briefly describes different fault tolerant algorithms previously proposed for other topologies. In Section 6.3, we describe KNS, the hybrid (direct-indirect) topology the proposed mechanism has been implemented for. In Section 6.4, we present the fault-tolerant methodology proposed in this paper that is based on using intermediate nodes. Section 6.5 evaluates different configurations of the new routing algorithm. Finally, in Section 6.6 some conclusions are drawn. Finally, some conclusions are drawn.

6.2 Related Work

In this section, we will give some background on fault-tolerant routing algorithms. We will focus on direct topologies because of the similarity with the KNS topology.

There are two different categories of fault tolerant techniques that rely on the routing algorithm. The first category reconfigures the routing tables when a failure occurs. In this case, routing tables should be updated according to the new topology that results after the failure [27–30]. This technique allows the network to tolerate any number of faults without requiring extra resources [31] as long as the network is still connected, thanks to its flexibility. However, this flexibility may kill performance due to the need of using topology agnostic routing algorithms [72], as the resulting network topology may become irregular. Irregular topologies do not consider

and, hence, do not take advantage of the specific characteristics of the topology, thus they often provide inferior performance.

On the other hand, the second category covers fault-tolerant routing algorithms. A large number of fault-tolerant routing algorithms for interconnection networks have been proposed in the literature, and specially for direct network topologies like tori and meshes. Some of them require adding resources (virtual channels), usually depending on the number of tolerated faults [35] or the number of dimensions of the topology [36]. Other routing algorithms are based on disabling faulty regions [37–41] or individual nodes [42–44] to route the packets around these faulty regions. However, to do this, these algorithms usually disable healthy nodes. In [46], the authors use the technique of Valiant routing [45] to implement an algorithm that uses intermediate nodes to avoid faults. This methodology requires a few virtual channels and does not disable healthy nodes. There are other routing algorithms that do not require extra virtual channels, like [32, 33] or [34]. The first two methods are able to tolerate only a few faults for low dimension meshes, while the third method can offer more tolerance against faults, but needs extra virtual channels in tori. These routing algorithms were specifically designed for meshes and tori and they provide bad traffic balance as a lot of traffic is directed towards a single link, which can be easily saturated, thus degrading interconnection network performance.

6.3 The k-ary n-direct s-indirect (KNS) topology

The topology of the interconnection network defines the connection pattern among its nodes. Topologies usually adopt a regular structure to simplify their implementation and the routing algorithm. Among the different taxonomies of regular topologies, the most commonly-used one divides them into direct and indirect topologies [2, 3].

Direct topologies usually adopt an orthogonal structure where nodes are organized in an n–dimensional space, and each processing node has an associated router. The nodes are connected in each dimension in a ring (torus) or array (meshes) fashion. 2-D or 3-D direct topologies are relatively easy to built as each topology dimension is mapped to a physical dimension. However, implementing direct topologies with more than three dimensions implies not only increasing its wiring complexity, but also the length of its links when they are mapped to the 3-D physical space. Indeed, the number of ports of the routers geometrically grows with the number of dimensions of the topology (as two ports per dimension are required). Therefore, for large

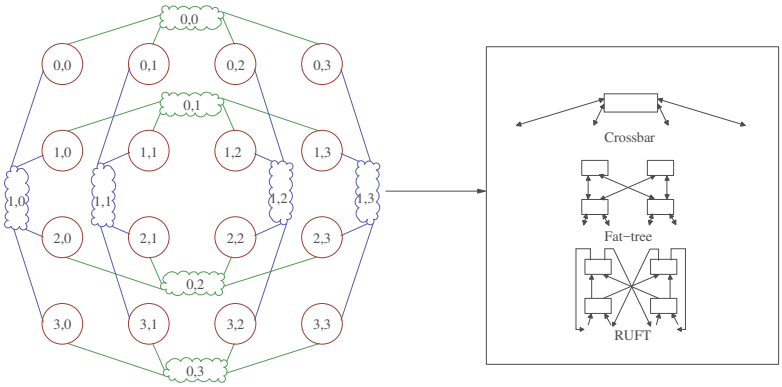


FIGURE 6.1: An example of the KNS topology with $n = 2$ and $k = 4$.

topologies, the implementation limitation in the number of dimensions leads to an increase in the number of nodes per dimension, which increases the communication latency, negatively impacting performance.

The alternative is to use an indirect topology. The main difference, compared to direct topologies, is that not all the routers have an associated processing node. The most common indirect topologies are multistage indirect networks (MINs), where switches are organized as a set of stages. For a large number of nodes, indirect topologies provide better performance than direct ones. However, this is achieved by using a higher amount of switches and links. Furthermore, their physical implementation may become very complex due to the fact that the wiring complexity grows with the number of nodes in the system, unlike direct topologies where complexity grows with the number of topology dimensions.

To overcome the limitations of direct and indirect topologies, hierarchical and hybrid topologies have been proposed. In [11], the authors propose the Flattened Butterfly, a variation of the butterfly topology obtained by using high-radix switches, that results in a direct topology. This topology can be seen as a generalized hypercube with concentration (i.e., several processing nodes are attached to every router), as all the nodes in the same dimension are fully connected (i.e., there is a link from each node to all the others of the same dimension). As an extension of this topology, the Dragonfly topology was proposed in [14], which provides a hierarchical topology that is based on grouping routers in virtual ones to increase the effective radix of the network. This topology uses two different networks, one intra–group and one inter–group. There are global channels that link different groups. It is advisable to use a non-minimal global adaptive routing to balance the load among the global channels. However, global channels are long links so that a high latency can be expected.

To solve the limitations of previous topologies, the k-ary n-direct s-indirect (KNS) was recently proposed [67]. Like meshes and tori, it is organized as an n–dimensional topology, but the rings or arrays that connect the nodes in each dimension are replaced by small indirect networks. In this way, communication latency along each dimension no longer linearly grows with the number of nodes per dimension. On the other hand, the small size of this indirect topology allows a reasonable wiring complexity opposite to large indirect topologies. This combination results in a family of topologies that provides high performance, with latency and throughput figures of merit close to the ones obtained with indirect topologies, but at a reduced hardware cost. The KNS topology is defined by three parameters: the number of network dimensions n, the number of nodes per dimension k, and the number of stages of the indirect subnetworks s that interconnect the nodes of each dimension. The total number of processing nodes is given by $N = k^n$.

Figure 6.1 shows an example of the KNS topology, with 2 dimensions and 4 nodes per dimension. Several solutions are feasible to connect the nodes of the same dimension. The simplest option is using a crossbar (i.e. a single switch, leading to a k-ary n-direct 1-indirect) to connect them, but a multistage topology like a fat-tree or a RUFT topology [5] can be used for a high number of nodes per dimension. Finally, any number of processing nodes can be attached to every network node (i.e., concentration). In KNS, we also refer to network nodes as routers, as opposed to the switches of the indirect subnets. In this paper, we focused only on k-ary n-direct 1-indirect topologies, although the proposed routing algorithm can be implemented in any KNS topology configuration.

6.4 Description of the Fault-Tolerant Routing Methodology

As stated above, in what follows, we will assume a KNS topology using crossbars as indirect subnetworks. Concerning routing, Hybrid-DOR [67] is used. It is a deterministic routing algorithm that crosses network dimensions in increasing order.

Hybrid-DOR works much like DOR (Dimension Order Routing) [3] for direct topologies. When a packet is injected, the router connected to the source node computes and compares the coordinates of the source (i.e., $s_{n-1} \dots s_1 s_0$) and destination (i.e., $d_{n-1} \dots d_1 d_0$) nodes in every dimension, and it sends the packet through the link that corresponds to the first dimension (i.e, f) the packet has to cross. Once the packet reaches the indirect subnetwork that corresponds

to the crossed dimension, the packet is routed to reach the node of the same dimension that has as coordinate in this dimension the one corresponding to the destination node (i.e., d_f). How to route the packet in the subnetwork depends on the subnetwork topology. If a single switch is used as an indirect subnetwork, the routing algorithm merely selects the output link which reaches the next router. If other topologies are used to implement the indirect subnetworks, any deadlock-free routing algorithm suitable for these topologies could be used. Notice that this is a local routing algorithm of the indirect subnetwork. This process is repeated for each dimension where the coordinates of the source and destination nodes are different. An increasing order is followed to ensure deadlock freedom. Notice that this routing algorithm is minimal and does not require virtual channels.

Let us analyze how to deal with network faults. We will only consider link faults, since a switch fault can be easily modeled as a switch with failures in all of its links. However, notice that in this case, link failures of a switch would be correlated. Moreover, as network links are bidirectional, we assume that if there is a link fault, then it fails in both directions. In this paper, we do not focus on how the failure information propagates to network nodes. In this way, a static fault model is assumed. This means that when a fault is discovered all the processes are stopped, the network is emptied, and a management application is run in order to deal with the fault. Checkpointing techniques must also be used so that applications can be brought back to a consistent state prior to the fault occurred. Detection of faults, checkpointing, and distribution of routing info is assumed to be performed as part of the static fault model, and are therefore not further discussed in this paper.

For each source-destination pair without failures in their path, packets are routed using Hybrid-DOR following minimal paths. But, if there is any fault in the path of a given source-destination pair, the methodology routes packets through intermediate nodes, like in [46]. The use of intermediate nodes was proposed in [45] for other purposes, such as traffic balancing. In our case, the idea is to avoid the faults by deviating the packet to an intermediate node. Therefore, a suitable intermediate node for each source-destination pair with faults in their path needs to be selected. The routing algorithm avoids faults by first sending the packet to an intermediate node and then, from this intermediate node, to the destination node. Several intermediate nodes could be also used for each pair of nodes, where packets are forwarded through these nodes until the destination node is reached. Notice that the Hybrid-DOR algorithm is used in all sub-paths. Using more than one intermediate node allows the mechanism to tolerate more faults by having more control over the global path followed by the packet. Notice also that the packets are not

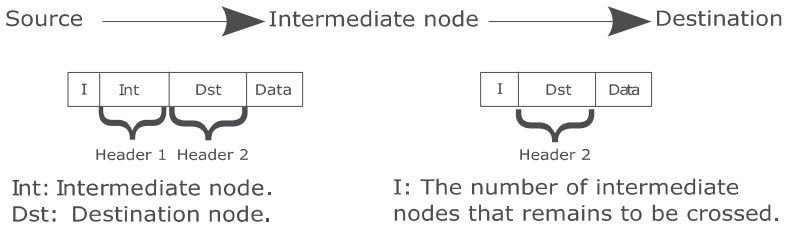


FIGURE 6.2: Header for packets using the intermediate node methodology.

ejected from the network when they reach intermediate nodes. In Sections 6.4.1 and 6.4.2 we will describe in detail how intermediate nodes are selected for the case of requiring only one, or several of them, respectively.

Regarding the structure of packets, several fields should be added to the packet header to support routing through intermediate nodes. Figure 6.2 shows the packet header. First, the number of intermediate nodes that the packet has to cross is stored in a new field (I in the figure). In addition to the destination node, the addresses of intermediate nodes are also stored in the packet header. Every time an intermediate node is reached, its address is removed from the packet header and the I field is decreased. Other implementations are possible, such as storing in the packet header a pointer to the field that should be considered for routing. As soon as the packet reaches an intermediate node, this field will point to the next intermediate one or, finally, to the destination node.

For each source–destination pair, the mechanism checks whether the deterministic Hybrid-DOR path is fault-free or not. If not, routing through intermediate nodes is required. A list of intermediate nodes should be computed for paths with faults. This list will be stored in a table at every source node. There is a table entry for each destination node that requires routing through intermediate nodes. This table can be implemented as linear (i.e., with as many entries as the network size) or as random (i.e., as content-addressable memories) tables. The size of the latter table depends on the number of faults the network has to tolerate. The higher this number, the higher the number of affected source–destination pairs. In practice, though, it is expected that it should be enough to tolerate a relatively low number of faults, as if a network suffers a high number of faults, the problem should be solved in another way.

However, the size of these tables also depends on how we codify the info stored in them. For example, considering a 2-D network, if the fault is located at the first dimension link of a node (i.e. the one that connects to the nodes of the same row), each source node in the same row (except the one connected to the faulty link) requires routing through intermediate nodes to reach

every other node located in the same column of the fault. However, in the same example, the source node that is connected with the faulty link needs an intermediate node for each destination node in every other columns. This means that $k*(k-1)$ destination nodes need an intermediate node to reach them from this source node. Generally speaking, for more network dimensions, if a fault occurs at a link of the i dimension, the source node requires using intermediate nodes to reach $(k^{(n-i-1)})*(k-1)$ destinations. Moreover, if there are more faulty links in this source node, each fault will involve a number of destinations according to this equation. Therefore, a table design where there is an entry for each destination that needs an intermediate node could require a big size in large networks. In such cases, an alternative design based on the use of masks, similar to IP routing tables, could be used. The table would have two fields (id to compare and mask to select the bits) and an option flag. The option flag indicates whether the corresponding entry is considered when the comparison is satisfied or not. Of course, there is also a field to store the possible intermediate nodes. In general, each entry can be used for a set of destinations, thus reducing the size of the table. However, some destination nodes may need specific intermediate nodes. Additional entries for them will be included in the table, which will lead to several hits for the same destination node. The solution is to insert the entries in the table following a given priority order and then selecting the entry with higher priority.

As in [46], we will refer to the source node as S and the destination node as D. For each intermediate node, we will use the notation I_x, where x represents the index of the intermediate node (I_1 for the first one, I_2 for the second one and so on). To ensure deadlock freedom, the routing algorithm needs at least as many additional virtual channels as intermediate nodes for fault-tolerant routing. For example, if we use up to two intermediate nodes, we need at least three virtual channels (i.e., the original plus two additional ones). When a packet reaches an intermediate node, the packet is re-injected into the network using a new virtual channel to avoid deadlocks. For instance, assume that two intermediate nodes are used. One virtual channel (v_1) is used from S to I_1, another one (v_2) from I_1 to I_2, and the last one (v_3) from I_2 to D. In this way, deadlocks are avoided because the network is split into three virtual networks, deadlock-free routing algorithm is used within each virtual network, and each virtual network transition is performed following a strict order ($v_1 \rightarrow v_2 \rightarrow v_3$ in the example). Adaptive routing could also be used. In this case, several virtual channels can be used for adaptive routing provided that there is an escape channel to break cyclic dependencies for each subpath [56]. Routing in the escape channels uses Hybrid-DOR. Adaptive channels can be used in any of the sub-paths. However, in such a case, a different escape channel is required in each sub-path to ensure deadlock freedom.

In this paper, though, we only focus on deterministic routing.

There are some combinations of failures that physically disconnect one or more nodes of the network. As the proposed methodology does not add new resources to the network, these sets of faults can not be supported. The aim of the proposed methodology is to provide a path for every source–destination pair, provided that they are physically connected by the interconnection network.

Next, we will present how the intermediate nodes are selected. First, we will focus on using only one intermediate node. After that, we show how to extend the methodology to use multiple intermediate nodes.

6.4.1 One Intermediate Node

In this case, we will use only one intermediate node when one or more faults affect the minimal path provided by Hybrid-DOR between a pair of nodes. This intermediate node, I_1, has to satisfy two rules:

R1. I_1 is reachable from S.

R2. D is reachable from I_1.

We say that a node Y is reachable from X if there is a minimal path provided by Hybrid-DOR between this pair of nodes, and there is no fault in it.

For direct topologies like tori or meshes, the choice of this intermediate node is very important, because the number of hops can greatly increase, depending on the selected intermediate node. However, in the KNS topology, the number of hops only depends on the number of dimensions that the packet must cross. The proposed methodology prioritizes the use of those intermediate nodes which allow the packet to reach the destination node without additional hops compared to the original minimal path between the source and the destination nodes.

For instance, Figure 6.3.(a) shows a 4-ary 2-direct 1-indirect network with a fault at the source node S in the x dimension link. So, it cannot reach the destination node D using Hybrid-DOR. In this case, the possible intermediate nodes are all the nodes of the same column (surrounded by the dashed line in the figure). The best choice, however, is the node that is located in the same

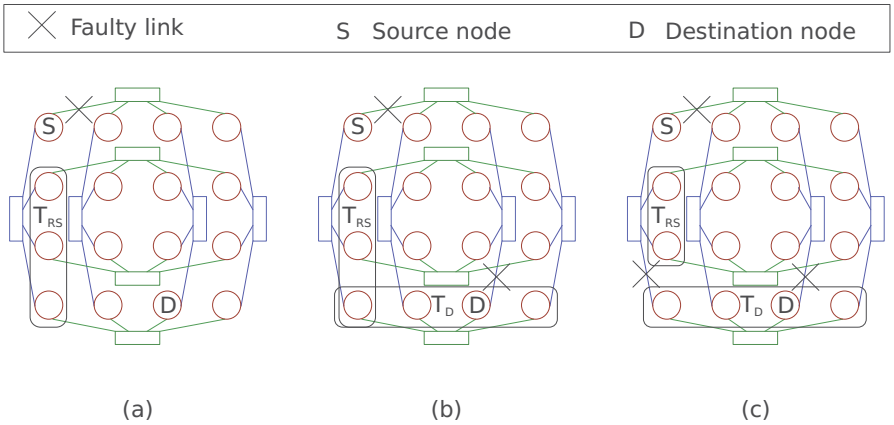


FIGURE 6.3: Examples of fault combinations in a KNS topology with $n = 2$ and $k = 4$.

row as the destination node, because it allows to reach the destination node by using a minimal path.

We say that a fault-tolerant routing algorithm is able to tolerate f failures if it can provide a valid path between every source-destination pair for any combination of up to f failures.

Lemma 6.1. *Given a KNS network with n dimensions, using one intermediate node, the routing algorithm can tolerate up to $n - 1$ failures.*

Proof. For a KNS network with n dimensions, each node has n links. Each link allows connecting to the other nodes of the same dimension by an indirect subnetwork. Let T_{RS} be the set of nodes reachable from S using Hybrid-DOR, and T_D the set of nodes from which D is reachable using Hybrid-DOR. As long as $T_{RS} \cap T_D$ is not empty for any source-destination pair, the network is able to handle the fault combination by using one intermediate node located in this set. Therefore, the worst fault combinations are the ones that reduce the number of nodes in T_{RS} and/or T_D set.

For instance, in Figure 6.3.(a), while T_D set comprises all network nodes (except S), the fault reduces T_{RS} to only 3 nodes. $T_{RS} \cap T_D$ cardinal is 3 and therefore the fault is tolerated. However, assume that, then, there is another link failure that reduces T_D. The worst scenario occurs when the fault is located at the Y dimension link of D (Figure 6.3.(b)). In this case, $T_{RS} \cap T_D$ is reduced to one node, and the fault is still tolerated. However, if a new failure appears in the links of this unique intermediate node (thus, there will be 3 faults in the network), the source–destination pair S–D will become disconnected. Remember that this example corresponds to one intermediate node. This situation can be seen in Figure 6.3.(c).

However, the worst scenario occurs when the source and the destination nodes are located in the same row, i.e., when all coordinates but the one of the first dimension are the same for both the source and destination nodes. In this case, with only two faults (the first two faults of the previous example: a link failure at the x link of the source node and another one at the y link of the destination node), the destination node is no longer reachable from the source node, and $T_{RS} \cap T_D$ is empty (see Figure 6.4). This means that the network is able to tolerate all the fault combinations of 1 fault, but neither of 2 faults, and therefore, the network supports one fault. Additionally, there are also 2-fault combinations that physically disconnect nodes (i.e. a failure in all links of a node) and therefore they are not reachable, and not all the source-destination pairs are able to communicate. These cases are not tolerated.

In general, for any n-dimensional KNS network, there is a non-tolerated scenario that happens when the source and the destination nodes have the same coordinates but the coordinate of the first dimension. If T_D is reduced due to faults on all links of the destination node but the one of the first dimension (which physically disconnects the node) and there is a fault on the link of the first dimension of the source node, both nodes become disconnected as there is not any suitable intermediate node.

Therefore, the minimal number of faults needed to disconnect the network with only one intermediate node is n faults ($n - 1$ failures at the destination node links plus one failure at the source). Thus, the routing algorithm is able to tolerate $n - 1$ failures. □

6.4.2 Multiple Intermediate Nodes

There are cases where only one intermediate node is not enough to handle the network fault combination. In these cases, the routing algorithm can use more than one intermediate node to have more chances of finding a set of fault-free deterministic Hybrid-DOR paths between the source and destination nodes. Assume that a number of x intermediate nodes $I_1, I_2, \ldots I_x$ are required. Intermediate nodes are selected according to the following rules:

R1. I_1 is reachable from S.

R2. I_{i+1} is reachable from I_i, for $0 < i < x, x > 1$.

R3. D is reachable from I_x.

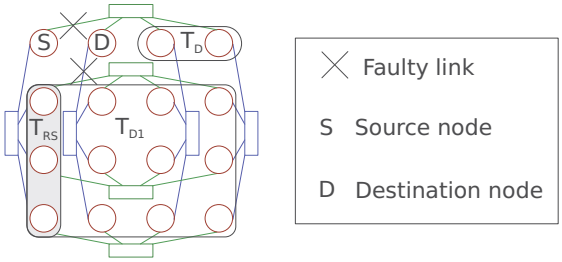


FIGURE 6.4: An example of the KNS topology with $n = 2$ and $k = 4$ and 2 faults.

Therefore, we can guarantee that the packet is able to reach its destination node following the path S-I_1-...-I_i-I_{i+1}-...-I_x-D.

In order to avoid non-minimal routing, the methodology tries to use a set of intermediate nodes that does not increase the number of hops beyond a minimal path. In particular, if there are non-minimal available paths using i intermediate nodes and also a minimal path using j intermediate nodes, where $i < j$, the latter will be finally selected.

Lemma 6.2. *Given a KNS network with $n > 2$ dimensions, using two intermediate nodes, the routing algorithm can tolerate $2 * (n - 1) + k - 3$ failures. If $n = 2$, the routing algorithm can tolerate $2 * k - 1$ faults.*

Proof. Let T_{D1} be the set of nodes from which the destination node is reachable using Hybrid-DOR using one intermediate node. That is, the set of nodes that can reach the destination node using an intermediate node from T_D. In Figure 6.4 we can see an example for a 4-ary 2-direct 1-indirect network with 2 faults. The first failure is located at the source node x-link and the second one at the destination node y-link. The Figure shows the set of reachable nodes from the source node (T_{RS} with gray background), the set of nodes which can reach the destination node (T_D), and the set of nodes that can reach the destination node using any node within T_D as an intermediate node (T_{D1}). Thus, in this case, we have to use one node of $T_{RS} \cap T_{D1}$ as the first intermediate node, and another one of T_D as the second intermediate node. As long as $T_{RS} \cap T_{D1}$ is not empty for any source-destination pair, the network is able to handle the fault combination. Therefore, the worst fault combinations are those that reduce the number of nodes in T_{RS}, T_D and/or T_{D1}.

As in Lemma 1, if all the destination node links fail except the link of the first dimension (to avoid disconnecting the network), T_D will be reduced to $k - 1$ nodes. T_D includes the nodes

which share all coordinates with the destination node, except the coordinate of the first dimension. On the other hand, if all links of the source node fail, except the one of the last dimension, T_{RS} will be reduced to $k-1$ nodes. T_{RS} includes the nodes which share all coordinates with the source node except the coordinate of the last dimension. In this way, the possible second intermediate nodes, T_D, are only reached by the link of the last dimension. But if we assume that these links are also faulty, the destination node will be no longer reachable. So, with $n-1$ faults at the source node links, $n-1$ faults at the destination node links and $k-1$ faults in the set of possible second intermediate nodes, the fault combination is not tolerated.

However, the worst scenario occurs when the source node shares all coordinates with the destination node, except the coordinate of the first dimension (see Figure 6.4). In this case, the destination node will not be reachable with only $k-2$ faulty links in the T_D set. Hence, the source–destination pair becomes disconnected with $2*(n-1)+k-2$ faults, ergo, the network can tolerate $2*(n-1)+k-3$ faults.

Notice, though, that for 2-D networks, adding faults for every possible second intermediate node (which comprises T_D) physically disconnects the row where the destination node is located at, as we can see in Figure 6.4. These corner cases are not tolerated by the methodology.

For 2-D topologies, the scenario is different. For instance, keeping the same number of faults at the destination node ($n-1=1$ fault) and at the source node ($n-1=1$ fault), and setting faults on the x-links of nodes in T_{RS} except one of them ($k-2$, see Figure 6.4), to avoid physically disconnecting the column, we have only one node in $T_{RS} \cap T_{D1}$, which will be the first intermediate node. From this first intermediate node, there are $k-2$ possible paths to the destination, i.e., one path for each possible second intermediate node which comprises T_D. If there are faults in all these paths, the destination node will not be reachable by the source node, i.e., $1+1+(k-2)+(k-2) = 2*k-2$ faults are necessary to not tolerating the fault combination. Thus, the routing algorithm is able to tolerate $2*k-1$ faults in a 2-D KNS network. □

6.4.3 Extension To Any Indirect Subnetwork

In this paper, we have focused on KNS topologies that use crossbars as indirect subnetworks. However, we can extend the methodology to KNS topologies that use other indirect subnetworks like fat-trees or RUFT. To do this, a specifically designed methodology should also be used to

tolerate faults on each indirect subnetwork. Therefore, intermediate nodes are used globally, while a specific methodology to tolerate faults will be used locally on each subnetwork.

While the subnetworks can avoid faults, the direct routers will work normally. However, if a node becomes unreachable due to a fault located at a given subnetwork, it will be modeled like a link fault at this node, in the link of the corresponding dimension of this faulty subnetwork.

6.5 Evaluation Results

To evaluate the proposed methodology, we have performed two kind of analysis. First, we analyze the number of network failures that can be tolerated. Remember that a fault-tolerant routing algorithm is able to tolerate n failures if it can provide a valid path between every source-destination pair for any combination of n failures. Notice that there are situations where the failures physically disconnect the network. We consider these situations as combinations where there is no path for all source-destination pairs and therefore the combination is not tolerated.

Second, we evaluate the network performance degradation in presence of faults when using the proposed methodology. To do this, we have simulated different tolerated network configurations with a varying number of faulty links under uniform traffic. For each number of faults, we have tested 50 random fault combinations to obtain the average network throughput and latency. In those experiments, the combinations where some nodes are physically disconnected are discarded and not simulated since only tolerated combinations are simulated.

We have analyzed the proposed methodology for two network configurations: a 32-ary 2-direct 1-indirect topology and a 10-ary 3-direct 1-indirect topology. Both configurations have a similar number of nodes (1,024 and 1,000 nodes, respectively), so we can analyze the impact of the number of dimensions on the behavior of the fault-tolerant routing algorithm proposed in this paper.

6.5.1 Simulation Model

To perform the simulations, we have used a simulation environment developed at our research group. A prior version of this tool was used to provide evaluation results in [3]. This tool is an event-driven simulator which models KNS topologies with bidirectional links and uses virtual cut–through switching. Each switch has a full crossbar with queues of 4 packets both at their

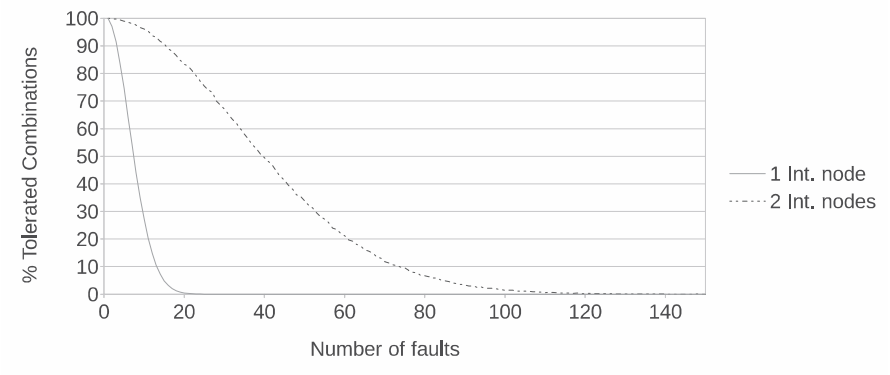


FIGURE 6.5: Fault combinations tolerated by the methodology when using one or two intermediate nodes in a 2-D network with 1,024 nodes.

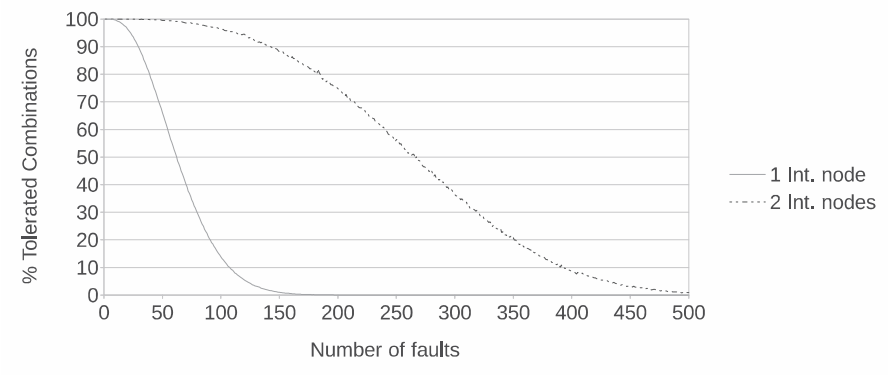


FIGURE 6.6: Fault combinations tolerated by the methodology when using one or two intermediate nodes in a 3-D network with 1,000 nodes.

input and output ports. Credits are used to implement the flow control mechanism. Packet length is 16-flit. We assume a pipelined router with a latency of 4 clock cycles, while the switch and link bandwidth is assumed to be one flit per clock cycle.

6.5.2 Fault Analysis

The number of possible fault combinations exponentially increases with the number of faults. For this reason, it is not possible to explore all possible fault combinations for a given number of faults in a reasonable amount of time. Therefore, we have used statistical analysis as a tool. Specifically, we have analyzed a subset of the fault combinations, where the faults are randomly chosen. This subset is large enough to obtain results with a confidence level of 99% and an error lower than 1%.

TABLE 6.1: Percentage of paths that use one or two intermediate nodes when at most two intermediate nodes are used: (a) 1,024-node 2-D network and (b) 1,000-node 3-D network.

Link faults	% of paths	
	1 inter.	2 inter.
1	0.19%	0%
2	0.38%	0.000003%
3	0.57%	0.000009%
4	0.75%	0.00018%
5	0.94%	0.000031%
6	1.13%	0.000046%
7	1.32%	0.000067%
8	1.50%	0.000087%
9	1.69%	0.000115%
10	1.88%	0.000145%
11	2.06%	0.000179%
12	2.25%	0.000216%
13	2.43%	0.000257%
14	2.61%	0.000307%
15	2.80%	0.00036%

(a)

Link faults	% of paths	
	1 inter.	2 inter.
1	0.18%	0%
2	0.36%	0%
3	0.54%	<0.000001%
4	0.72%	<0.000001%
5	0.90%	<0.000001%
6	1.08%	<0.000001%
7	1.25%	<0.000001%
8	1.43%	<0.000001%
9	1.61%	<0.000001%
10	1.79%	<0.000001%
11	1.96%	<0.000001%
12	2.14%	0.000001%
13	2.32%	0.000002%
14	2.50%	0.000002%
15	2.67%	0.000002%

(b)

Figures 6.5 and 6.6 show the percentage of tolerated fault combinations for different number of faults using one or two intermediate nodes for a 2-D network with 1,024 nodes and a 3-D network with 1,000 nodes, respectively. The results are shown for a number of link faults up to 150 for 2-D networks and 500 for 3-D networks. First, as expected, the percentage of tolerated combinations of faults strongly increases when using two intermediate nodes instead of only one because there is more control over the path followed by the packet. That is, using two intermediate nodes instead of only one provides more alternative paths or, what is the same, we have more options to configure the final path to the destination node, avoiding the faults. On the other hand, in the 2-D network, the percentage of tolerated combinations of faults is considerably lower than in the 3-D one because more alternative paths are available in the latter for each source-destination pair. In the case of the 3-D network, the methodology is able to tolerate more than 99.5% of the 10-fault combinations with only one intermediate node and more than 99.98% for the configurations of 15 faults with 2 intermediate nodes. The fact of having more dimensions gives more probablity to route the packet through different paths that do not share resources, being able to avoid more faults. However, the 3-D network has more resources. There are 3,000 links in the 3-D network versus 2,048 links in the 2-D network. This is why, for 23 faults in the 2-D network even with two intermediate nodes, the percentage of tolerated combinations is lower than 80%. However, in the 3-D network, more than 100 faults

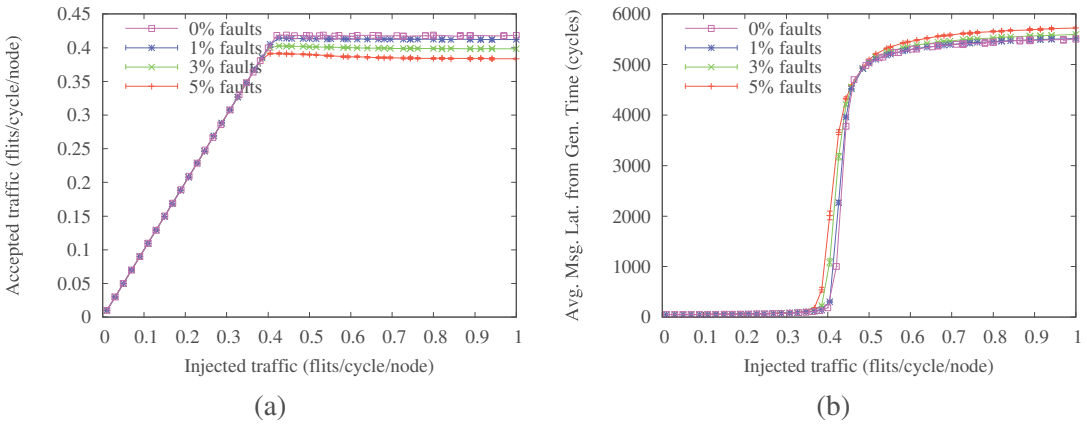


FIGURE 6.7: 32-ary 2-direct 1-indirect under uniform traffic: (a) Accepted traffic and (b) average latency versus injected traffic.

are needed to reach a percentage of tolerated combinations lower than 80%.

Tables 6.1.(a) and 6.1.(b) show the percentage of paths that have to use intermediate nodes to reach the destination node, and how many intermediate nodes are needed when up to two intermediate nodes can be used for a 2-D network with 1,024 nodes and a 3-D network with 1,000 nodes, respectively. These tables only show the information for up to 15 faulty links. Regarding the percentage of paths using intermediate nodes, we can see that it is quite low for one intermediate node, and much lower using a second intermediate node. Even for 15 faults, less than 3% of the paths use intermediate nodes. Therefore, it is expected that the extra latency due to the intermediate nodes does not significantly impact the final average latency, as the number of affected paths is extremely low. In the case of 3-D networks, the percentage of paths that use intermediate nodes is even lower.

6.5.3 Performance Analysis

In this Section we analyze the performance degradation suffered by the network when applying the fault-tolerant routing methodology in presence of faults. To do this, we have simulated several network scenarios with 1% faulty links, 3% faulty links and 5% faulty links. For each fault scenario, we have generated 50 random fault combinations, all of them tolerated by the methodology. That is, all source-destination pairs are able to communicate. Up to 2 intermediate nodes can be used, since this allows to test fault combinations with a higher number of faults. However, the fact of using two rather than only one intermediate node does not strongly impact

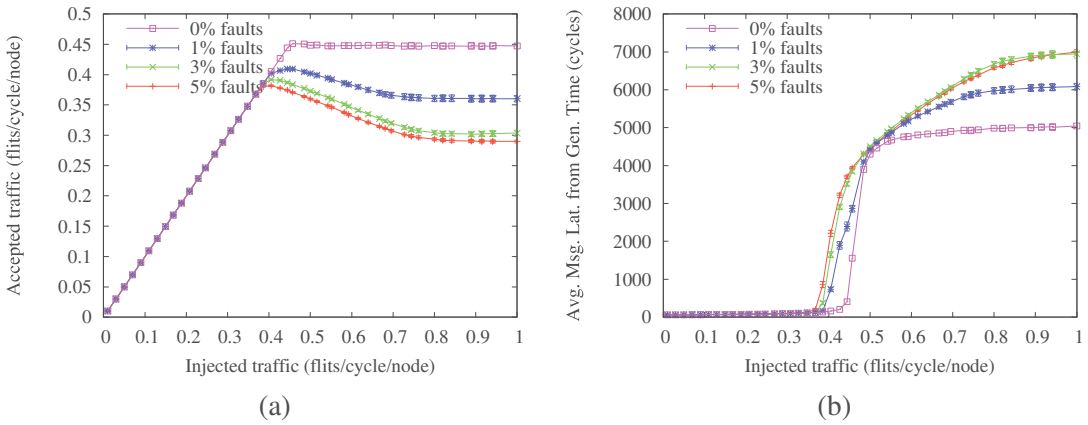


FIGURE 6.8: 10-ary 3-direct 1-indirect under uniform traffic: (a) Accepted traffic and (b) average latency versus injected traffic.

network performance. This is because the methodology is able to avoid the fault using only one intermediate node in most of the cases, although it is able to use two (as shown in Table 6.1.(a) and 6.1.(b)).

In order to measure the performance degradation, we obtained the accepted traffic and average latency form generation time versus injected traffic. We consider average latency form generation time to consider the time spent by the packet at the injection queue. Accepted traffic is measured as the amount of data per node and per time that the network can accept (flits/cycle/node). Network throughput is the peak value of accepted traffic. Average latency form generation time is measured as the mean of the elapsed time from message generation at the source node until its ejection at the destination node. For network load, source nodes inject traffic following a uniform traffic pattern (i.e., randomly selecting the destination node).

Figures 6.7.(a) and 6.7.(b) show results for a 32-ary 2-direct 1-indirect network under uniform traffic. In this case, the network suffers a performance degradation of about 1% in throughput with 1% faulty links (21 links) compared to the same network without faults. For 3% and 5% faulty links, performance degradation increases to 3,8% and 6,5%, respectively. Latency is affected as well, increasing the average value with respect to the fault-free case. In particular, at saturation, latency is increased by 67% with 1% faulty links.

For the 10-ary 3-direct 1-indirect network (Figures 6.8.(a) and 6.8.(b)) the performance degradation, for the same percentage of faulty links, is higher when compared to the 32-ary 2-direct 1-indirect network. In particular, network throughput degrades by about 9% with 1% faulty links (30 faults) compared to the fault-free network, and by 13% and 15% with 3% and 5%

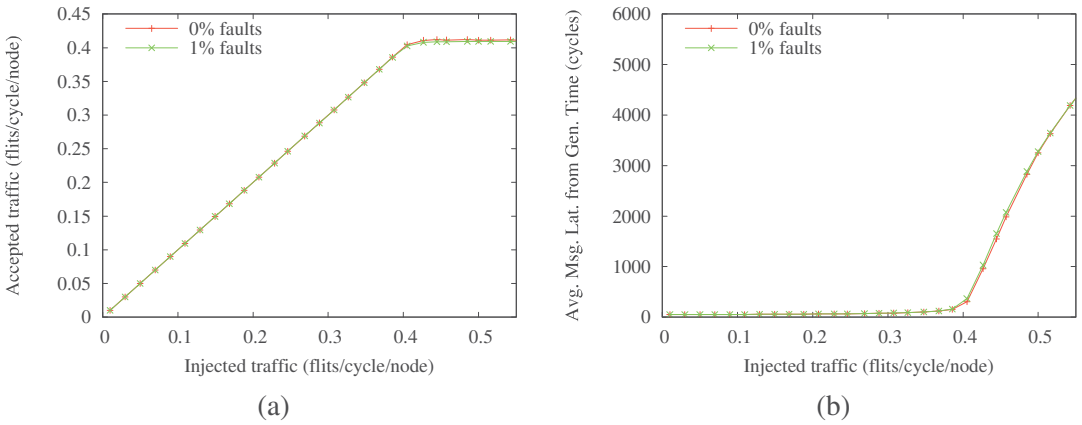


FIGURE 6.9: 64-ary 2-direct 1-indirect under uniform traffic: (a) Accepted traffic and (b) average latency versus injected traffic.

faulty links, respectively. The increase in latency for the 1% link faults case, is 1.8 times at the saturation point. Notice that the increase in latencies is because the network with faults saturates before than the one without faults. If we focus on the base latency or the latency before saturation, we can see that there is almost no impact. This is because the use of intermediate nodes is only required for a low percentage of source–destination pairs (Tables 6.1.(a) and 6.1.(b)). Although both networks have roughly the same number of nodes, they have a different number of links and, for the same number of relative link faults, the 3-D network involves more faulty links. Therefore, more paths are affected. On the other hand, as shown in Figures 6.5 and 6.6, the fact of having a higher number of dimensions with the same number of nodes improves the probability of avoiding a given fault combination. Therefore, although throughput is degraded by 3.8% with 3% faulty links in the 2-D network versus a performance degradation of 13% with 3% faulty links in the 3-D network, the probability of supporting a combination with this number of faults in the 3-D network is about 97%, against only 16% in the 2-D network.

We can see the same behavior in Figures 6.9.(a), 6.9.(b), 6.10.(a), 6.10.(b), 6.11.(a) and 6.11.(b), where results for larger networks are shown. In this case, we consider 4,096-node networks and we only checked a number of faults equal to 1% of the links. Again, for 2-D networks, the performance is not affected to any large extent. On the other hand, for 3-D networks, the performance degradation is more intense, but remember that this network has a higher absolute number of faulty links for the same percentage of faults. Remember also that the higher the number of network dimensions, the higher the number of tolerated faults. Therefore, there is a trade-off. If we have a network with a very low probability of faults, or the faults can be

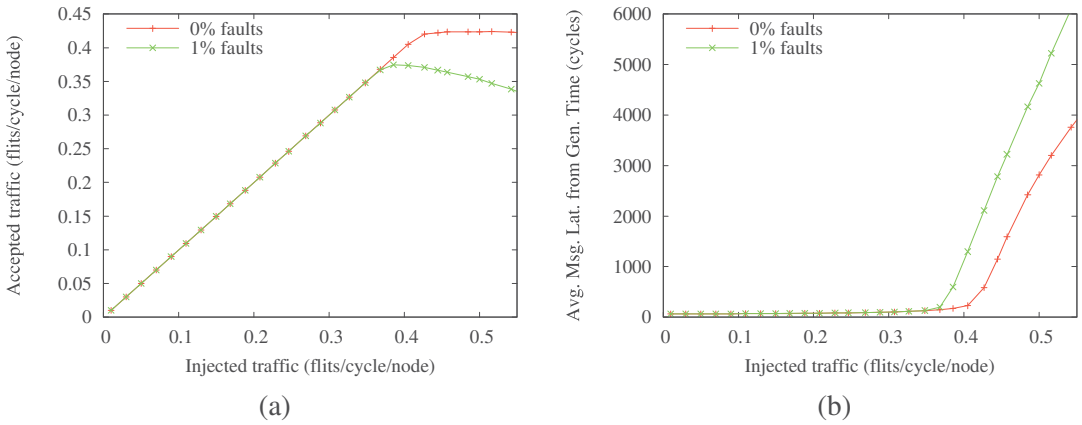


FIGURE 6.10: 16-ary 3-direct 1-indirect under uniform traffic: (a) Accepted traffic and (b) average latency versus injected traffic.

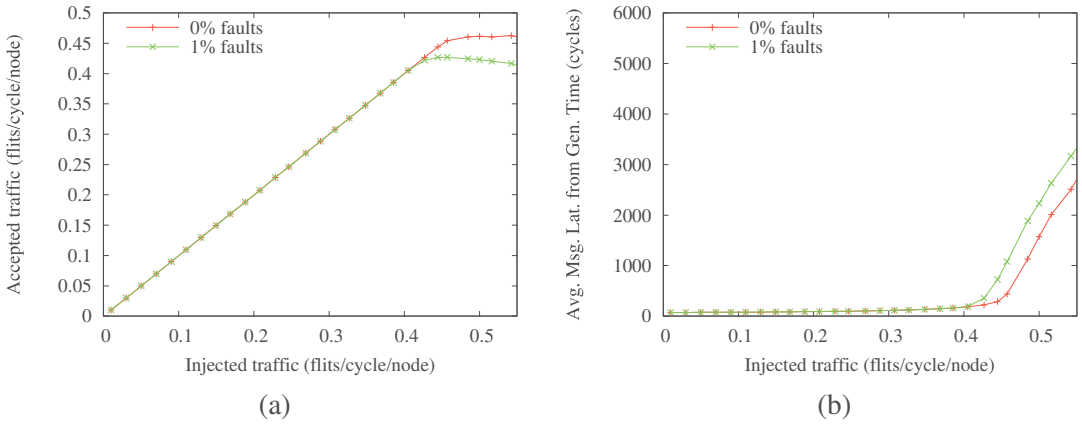


FIGURE 6.11: 8-ary 4-direct 1-indirect under uniform traffic: (a) Accepted traffic and (b) average latency versus injected traffic.

repaired in a short period of time, we should choose a network configuration with a low number of dimensions. Otherwise, if the probability of faults is higher, network configurations with more dimensions would be a better choice.

To summarize, the proposed methodology offers a better tolerance of faults for networks with more dimensions, but this does not necessarily mean a better performance, because it depends on the fault combination and the selected intermediate nodes. With more dimensions, the selected intermediate nodes can increase the number of hops to a greater extent, as the distance between two nodes in a KNS topology increases with the number of dimensions.

6.6 Conclusions

This paper proposes and evaluates a fault-tolerant routing algorithm for k-ary n-direct 1-indirect topologies. This new routing algorithm is based on the use of intermediate nodes assuming a static fault model. It is able to tolerate a large number of faults without suffering a great fall in performance. This routing algorithm does not disable any healthy node, unlike other algorithms, and does not require any additional resources except one extra virtual channel per each intermediate node used. The mechanism is able to tolerate 99.5% of 10-fault combinations with only one intermediate node and more than 99.98% for the configurations of 15 faults with 2 intermediate nodes in a 3-D network with 1,000 nodes. The proposed fault-tolerant routing algorithm has been evaluated by simulation under uniform traffic and the results show that the network performance only suffers a small degradation. For instance, using only two intermediate nodes (2 extra virtual channels), the evaluation results show a performance degradation of 1% for a 2-D network with 1024 nodes and 1% faulty links (21 faults). The proposed methodology can be easily extended to other configurations of the KNS network topology.

Chapter 7

IODET: A HoL-blocking-aware Deterministic Routing Algorithm for Direct Topologies

Authors: Roberto Peñaranda (Universidad Politécnica de Valencia), Crispín Gómez (Universidad de Castilla La Mancha), María Engracia Gómez, Pedro López, Jose Duato (Universidad Politécnica de Valencia).

Type: Conference.

Conference: IEEE 18th International Conference on Parallel and Distributed Systems (ICPADS).

Location: Singapore.

Year: 2012.

DOI: $http://dx.doi.org/10.1109/ICPADS.2012.103$

URL: $http://ieeexplore.ieee.org/xpls/abs\_all.jsp?arnumber=6413629$

Abstract

In large parallel computers routing is a key design point to obtain the maximum possible performance out of the interconnection network. Routing can be classified into two categories depending on the number of routing options that a packet can use to go from its source to its destination. If the packet can only use a single predetermined path then the routing is deterministic, whereas if several paths are possible it is adaptive. It is a well-known fact that adaptive routing usually outperforms deterministic routing; but in this paper we take the challenge of developing a HOL-blocking-aware deterministic routing algorithm that can obtain a similar or even better performance than adaptive routing, while decreasing its implementation complexity and providing some inherent advantages to deterministic routing such as in-order delivery of packets. In this large computers regular direct topologies are widely-used, so in this paper we focus on meshes and tori.

7.1 Introduction

The interconnection network performance strongly impacts the performance of the large parallel computers. Latency and throughput are the key performance metrics of interconnection networks [2, 3]. To achieve the required performance level, the designer manipulates three main parameters [2, 3]: topology, routing and switching. Topology usually adopts a regular structure that simplifies routing, implementation and expansion capability. Among the different taxonomies of regular topologies, the most commonly one divides them into direct topologies and indirect topologies. Taking into account that many of the very large machines of the top500 list [1] adopt a direct network topology, in this paper, we will focus on direct networks, although its conclusions could be extrapolated to indirect networks. Routing is a critical design issue of interconnection networks [3]. Routing algorithms can be deterministic or adaptive. In deterministic routing schemes, an injected packet traverses a unique, fixed, predetermined path between source and destination, while in adaptive routing schemes several paths are available. However, as several routes are possible, a choice of the path that will finally used is required, which makes routing operation more complex. In addition, several concerns about deadlock-freedom must be taking into account. Moreover, adaptive routing introduces the problem of out-of-order delivery of packets, which occurs when a packet sent from a given source arrives to a given destination before another one sent previously from the same source. In-order packet delivery is important

for cache coherence protocols and communication libraries. While there are solutions to this problem (for instance by using a reordering buffer at destinations [53]), they are not simple, as they require the use of storage resources and control packets. Instead, deterministic routing guarantees in-order packet delivery.

In this paper, we explore the behavior of both deterministic and adaptive routing on direct topologies using VCT switching. We also propose a new deterministic routing algorithm designed to reduce the HoL blocking effect.

The rest of the paper is organized as follows. Section 7.2 introduces background knowledge about direct topologies and routing in them. In Section 7.3, we present the proposed techniques to utilize the virtual channels in the deterministic routing. Section 7.4 evaluates the different topologies (torus and mesh) with different routing algorithms. Finally, some conclusions are drawn.

7.2 Direct topologies

This kind of topologies is known as k–ary n–cubes, being k the number of nodes in each of the n dimensions. In these topologies, nodes are labeled by an identifier with as many components as dimensions in the topology $\langle p_{n-1}, ..., p_0 \rangle$, and the value of the component associated to each dimension ranges from 0 to $k-1$ (i.e., nodes are numbered from $\langle 0, 0, ..., 0 \rangle$ to $\langle k-1, k-1, ..., k-1 \rangle$).

Deterministic routing in meshes and tori can be implemented with the DOR (dimension-order routing) routing algorithm [3]. This algorithm routes packets by crossing dimensions in strictly increasing (or decreasing) order. However, in tori, it is not enough to obtain a deadlock-free routing algorithm. Cycles are broken by splitting each physical channel into two virtual channels (VCs) [54]. Another technique used to avoid deadlocks in tori with deterministic routing is the bubble flow control mechanism [55].

Adaptive routing can be achieved in meshes and tori by allowing packets to cross dimensions in any order. According to Duato's theory [56], we add a new VC (or more) that may be used to cross network dimensions in any order. Deadlock freedom is achieved by providing a *escape path* to packets.

7.3 A HoL-blocking-aware Deterministic Routing Algorithm

Adaptive routing requires more resources than deterministic routing. Indeed, more VCs imply not only increasing the number of buffers but also increasing the crossbar size and the routing algorithm complexity since it has to deal with a higher number of output ports[1]. In this paper, we design a new deterministic routing algorithm which tries to reduce the HoL-blocking effect.

A first approach is using deterministic routing with several VCs which offer several routing options (the number of VCs) and requires the use of a selection function as adaptive routing does. This routing algorithm improves network performance over the baseline deterministic routing. However, switch complexity and routing time will be increased. Moreover, deterministic routing with several VCs may also introduce out-of-order delivery of packets. For this reason, we will refer to this mechanism as OODET (Out-of-Order DETerministic routing).

However, with a different method used to assign packet destinations to VCs, the resulting routing algorithm will preserve all the advantages associated to deterministic routing (simpler, faster and guaranteeing in-order delivery). The idea is to take into account packet destination to select the VC that will finally be used. The final output VC is selected using the component of the destination corresponding to the dimension in which the packet is being routed modulo the number of VCs. In [25], a similar mechanism was used but considering the whole destination identifier. In our proposal, we assign packets to VCs according to the component of the dimension. A nice property of this mechanism is that, it preserves, by design, in-order delivery of packets. On the other hand, VC selection is easily done, as only the LSBs of a component from destination identifier are required. In contrast to OODET, we will refer to this mechanism as IODET (In-Order DETerministic routing).

7.4 Experimental Evaluation

In this section, we compare by simulation adaptive routing versus deterministic routing in meshes and tori. Each node has a switch based on a full crossbar with queues of two packets both at their input and output ports. We assume that switch and link bandwidth is one flit per clock cycle, and fly time is 1 clock cycle. Each node has 20 clock cycles to implement the routing algorithm in the baseline deterministic routing with one virtual channel. However, when

[1] Even if the crossbar is multiplexed, VCs will add the multiplexers and arbiters to choose which VC will access the crossbar each time.

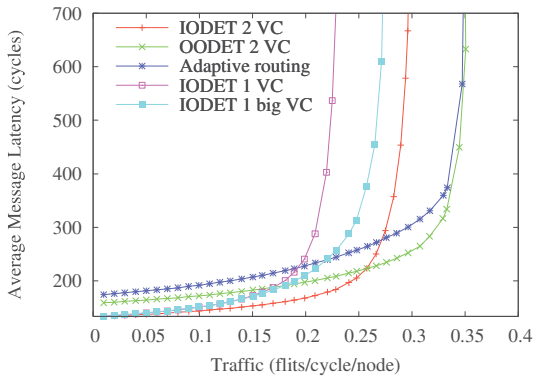


FIGURE 7.1: Average packet network latency vs. accepted traffic with uniform traffic pattern for 2D torus with 64 nodes..

several routing options are available and one has to be selected, we have increased the routing delay using Chien's model ([57, 58]). This model states that routing time depends on the degree of freedom (i.e., the number of routing options) of the routing algorithm. In particular, it is the sum of a constant time (i.e. the one required to apply the routing function) plus a component that grows logarithmically with the number of routing options. If this fact is taken into account, OODET and adaptive routing would take more clock cycles to complete the routing of a packet than IODET.

7.4.1 Performance analysis

Regarding the behavior of the mesh, adaptive routing outperforms the baseline deterministic routing, with one VC. But adaptive routing uses two VCs of size 2 packets. For this reason, it has been compared to a deterministic routing with one VC with double size, 4 packets (big VC). But, although it improves the baseline deterministic routing, it does not reach the performance of adaptive routing. If we compare it with the same number of VC, both IODET as OODET improves the adaptive routing.

Figure 7.1 shows the behavior of a 2D torus with 64 nodes. In this case, adaptive routing works better than in mesh. This is because the mesh is not a regular topology and the adaptive routing concentrates the traffic in the center of the network, lowering performance. We can see that OODET and the adaptive routing outperform IODET. But IODET has a lower cost, because the number of switching elements required is reduced by the restrictions presented by this routing algorithm. So we take this cost difference to add more virtual channels and achieve the performance of OODET and adaptive routing, also offering in-order delivery.

7.5 Conclusions

This paper focuses on developing a HOL-blocking-aware deterministic routing for direct topologies that uses a similar amount of resources to adaptive routing, i.e., uses the same number of VCs. Moreover, this routing algorithm keeps most of the good features of deterministic routing over adaptive one: simplicity, low routing times and in-order delivery of packets. The differences between adaptive and deterministic routing are shifted towards deterministic, becoming the best routing option in most of the cases. Furthermore, if we consider the cost of each routing algorithm, the proposed deterministic routing algorithm becomes the best routing option, since although the same of VCs is used in deterministic routing, the amount of switching elements required is much smaller, being even half the amount required by adaptive routing.

Chapter 8

Deterministic Routing with HoL-Blocking-Awareness for Direct Topologies

Authors: Roberto Peñaranda (Universidad Politécnica de Valencia), Crispín Gómez (Universidad de Castilla La Mancha), María Engracia Gómez, Pedro López, Jose Duato (Universidad Politécnica de Valencia).

Type: Conference.

Conference: International Conference on Computational Science (ICCS).

Location: Barcelona, Spain.

Year: 2013.

DOI: $http://dx.doi.org/10.1016/j.procs.2013.05.432$

URL: $http://www.sciencedirect.com/science/article/pii/S1877050913005759$

Abstract

Routing is a key design factor to obtain the maximum performance out of interconnection networks. Depending on the number of routing options that packets may use, routing algorithms are classified into two categories. If the packet can only use a single predetermined path, routing is deterministic, whereas if several paths are available, it is adaptive. It is well-known that adaptive routing usually outperforms deterministic routing. However, adaptive routers are more complex and introduces out-of-order delivery of packets. In this paper, we take up the challenge of developing a deterministic routing algorithm for direct topologies that can obtain a similar performance than adaptive routing, while providing the inherent advantages of deterministic routing such as in-order delivery of packets and implementation simplicity. The proposed deterministic routing algorithm is aware of the HoL-blocking effect, and it is designed to reduce it, which, as known, it is a key contributor to degrade interconnection network performance.

8.1 Introduction

The interconnection network performance strongly impacts on the performance of large parallel computers. Latency and throughput are the key performance metrics of interconnection networks [2, 3]. To achieve the required performance level, the designer manipulates three main parameters [2, 3]: topology, routing and switching. The switching mechanism decides how resources are allocated to the messages while they advance through the network. Topology usually adopts a regular structure that simplifies routing, implementation and expansion capability. Among the different taxonomies of regular topologies, the most commonly one divides them into direct and indirect topologies. Taking into account that many of the very large machines of the top500 list [1] adopt a direct network topology, in this paper, we will focus on direct networks, although its conclusions could be extrapolated to indirect networks. Routing is a critical design issue of interconnection networks [3]. Routing algorithms can be deterministic or adaptive. In deterministic routing schemes, an injected packet traverses a unique, fixed, predetermined path between source and destination, while in adaptive routing schemes, several paths are available. However, as several routes are possible, a choice or selection of the path that will be finally used is required, which makes routing operation more complex. In addition, several concerns about deadlock-freedom must be taking into account. Moreover, adaptive routing introduces the problem of out-of-order delivery of packets, which occurs when a packet sent from

a given source arrives to a given destination before another one sent previously from the same source to the same destination. In-order packet delivery is important for cache coherence protocols and communication libraries. While there are solutions to this problem (for instance by using a reordering buffer at destinations [53]), they are not simple, as they require the use of storage resources and control packets. Instead, deterministic routing guarantees in-order packet delivery by design. Another issue to consider when designing routing algorithms is to avoid interference among packets destined to different nodes [22, 59], since the Head-Of-Line (HoL) blocking effect may limit the throughput of the switch up to 58% of its peak value [60–62].

In this paper, we explore the behavior of both deterministic and adaptive routing on direct topologies, also proposing a new deterministic routing algorithm that takes advantage of virtual channels to reduce the HoL blocking effect.

The rest of the paper is organized as follows. Section 8.2 introduces some background. In Section 8.3, we present the new HoL-blocking-aware deterministic routing algorithm with virtual channels. Section 8.4 evaluates different topologies (torus and mesh) with different routing algorithms. Finally, some conclusions are drawn.

8.2 Direct topologies

The most important regular direct topology is the k–ary n–cube, which has k nodes in each of its n dimensions, connected in a ring fashion. In this topology, nodes are labeled by an identifier with as many components as dimensions in the topology $\langle p_{n-1}, ..., p_0 \rangle$, and the value of the component associated to each dimension ranges from 0 to $k-1$ (i.e., nodes are numbered from $\langle 0, 0, ..., 0 \rangle$ to $\langle k-1, k-1, ..., k-1 \rangle$). This topology is popularly known as torus. A mesh is a particular case where the k nodes of each dimension are connected by a linear array (i.e. without wraparound links).

Deterministic routing in meshes and tori can be implemented with the DOR (dimension-order routing) routing algorithm [3]. This algorithm routes packets by crossing dimensions in strictly increasing (or decreasing) order. Despite that DOR is deadlock-free in meshes, it is not in tori due to the wraparound links. Channel dependence graph cycles has to be broken by splitting each physical channel into two VCs [54]. Another technique used to avoid deadlocks in tori with deterministic routing is the bubble flow control mechanism [55].

Adaptive routing can be achieved in meshes and tori by allowing packets to cross dimensions in any order. According to Duato's theory [56], we add a set of VCs that may be used to cross network dimensions in any order. Deadlock freedom is achieved by providing a *escape path* to packets, by means of using a deadlock-free routing algorithm (for instance, DOR) in other virtual channel.

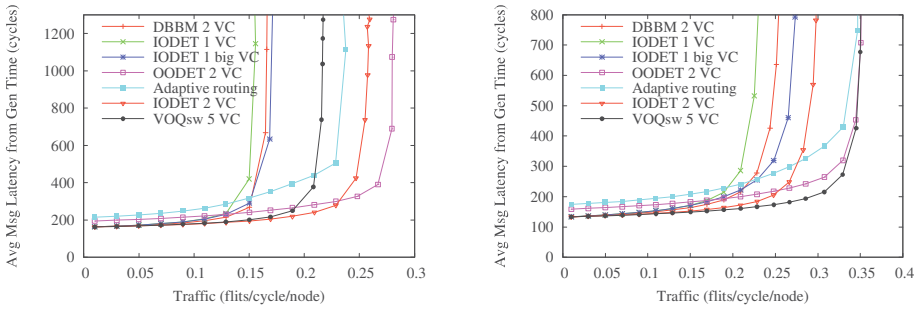
8.3 A HoL-blocking-aware Deterministic Routing Algorithm

Adaptive routing requires more resources (i.e. VCs) than deterministic routing. Indeed, more VCs imply not only more buffers but also increasing the crossbar size and the routing algorithm complexity. We will analyze alternatives to improve network performance also based on the use on VCs but trying to reduce this complexity.

A first approach is to use deterministic routing with several VCs, which offer as many routing options as VCs, also requiring a selection function as adaptive routing does. Although this routing algorithm improves network performance over the baseline deterministic routing (see Section 8.4), switch complexity and therefore routing time are still increased. Moreover, deterministic routing with several VCs may also introduce out-of-order delivery of packets. For this reason, we will refer to this mechanism as OODET (Out-of-Order DETerministic routing).

What is actually needed is a method to assign packets to VCs. We propose to classify them depending on their destinations. If a packet destined to a given node is only assigned to one VC, always the same, the result is a deterministic routing algorithm, with all its advantages (simpler, faster and in-order delivery of packets). In particular, VCs are assigned to packets according to the component of its destination node in the dimension in which the packet is being routed, modulo the number of VCs. As a consequence, the proposed mechanism classifies packets among the VCs, thus contributing to reduce the interference among packets, and, therefore, to reduce the HoL-blocking effect. A nice property of this mechanism is, that, as only one routing option is provided for each destination, it preserves, by design, in-order delivery of packets. On the other hand, VC selection is easily done, as only the LSBs of a component from destination identifier are required. In contrast to OODET, we will refer to this mechanism as IODET (In-Order DETerministic routing).

In [25], a similar mechanism (DBBM) was proposed, but considering the whole destination identifier modulo the number of VCs to assign packets to VCs. The fact of only considering



(a) 8×8 Mesh (b) 8×8 Torus

FIGURE 8.1: Average packet latency vs. accepted traffic for uniform traffic.

the LSBs of the destination provides lower opportunities of classifying packets among VCs (see Section 8.4). DBBM is a simplification of VOQnet [23], which needs as many VCs as nodes in the network, and associates each VC to a different destination. Another scheme is VOQsw [24], in which VCs are selected according to the next output port the packet will use. VOQsw and VOQnet are not scalable as the number of VCs they require depends on the system or switch size, respectively.

8.4 Experimental Evaluation

In this section, we compare by simulation adaptive versus deterministic routing in meshes and tori. Each node has a switch based on a full crossbar with queues of two packets both at their input and output ports. Switch and link bandwidth is one flit per clock cycle, and link fly time is 1 clock cycle. We also assume that each node require 20 clock cycles to apply the routing algorithm in the baseline deterministic routing (with one VC). To consider the increased complexity of adaptive routing, we have assumed a higher routing time. In particular, we have used Chien's model ([57, 58]). This model states that routing time depends on the degree of freedom (i.e., the number of routing options) of the routing algorithm. It is the sum of a constant time (i.e. the one required to apply the routing function) plus a component that grows logarithmically with the number of routing options (i.e. the selection function delay). If this fact is taken into account, OODET and adaptive routing would take more clock cycles to route a packet than IODET, DBBM, and VOQsw.

In Figure 8.1.(a) we can see the behavior of 2D mesh with 64 nodes. As expected, adaptive routing outperforms the baseline deterministic routing, with one VC. But adaptive routing uses two VCs, each one with space for two packets. For this reason, we have also included in the

TABLE 8.1: Comparison of the number of switching elements for the torus topology

Dimensions	Virtual channels	OODET	Adaptive	IODET
2	2	48	64	40
3	2	96	144	84
4	2	160	256	144
6	2	336	576	312
2	4	160	224	112
3	4	336	528	264
4	4	576	960	480
6	4	1248	2208	1104
Dimensions	Virtual channels	OODET	Adaptive	IODET
2	6	336	480	216
3	6	720	1152	540
4	6	1248	2112	1008
6	6	2736	4896	2376
2	8	576	832	352
3	8	1248	2016	912
4	8	2176	3712	1728
6	8	4800	8640	4128

comparison a deterministic routing with one VC but with double size (4 packets, big VC in the Figure). Although it improves the baseline deterministic routing, it does not reach the performance of adaptive routing. On the other hand, IODET and OODET with 2 VCs outperform adaptive routing. This is due to the ability of IODET to classify packets and the increased routing delay of adaptive routing. Notice that DBBM does not improve very much the performance of deterministic routing. The reason is that, as it uses the modulo of the whole packet destination identifier, packets may not classified in all dimensions. In fact, in some dimensions, all packets may use the same VC, wasting the other VCs. Finally, regarding VOQsw, in spite of using 5 VCs, it obtains a worse performance than adaptive routing.

Figure 8.1.(b) shows the behavior of a 2D torus with 64 nodes. In this case, adaptive routing works better than in mesh. This is because the mesh is not a true regular topology and the adaptive routing algorithm concentrates the traffic in the center of the network, lowering performance. We can see that OODET, VOQsw and adaptive routing outperform IODET. However, IODET has a lower cost, because the number of switching elements required is strongly reduced if the restrictions introduced by the routing algorithm are considered in the design of the switch. This can be seen in Table 8.1. Notice that VOQsw needs the same number of switching elements as OODET if the same number of VCs are used, because in a given dimension a packet

in VOQsw can change the VC at each hop, like OODET. As it can be seen, IODET requires the lowest number of switching elements in all analyzed cases.

8.5 Conclusions

This paper proposes a new HoL-blocking-aware deterministic routing algorithm (IODET) for direct regular topologies. It uses virtual channel flow-control, and assigns packets to VCs according to a subset of bits of the destination identifier (i.e., the component that corresponds to the dimension the packet is traversing). The result is a deterministic routing algorithm which exhibits its well-known advantages over adaptive routing: simplicity, low routing times and in-order delivery of packets. If routing times are scaled according to the number of routing options of each routing algorithm, IODET is able to outperform adaptive routing in meshes, while it reaches a performance half-way between the baseline deterministic and adaptive routing algorithms in torus. In addition, IODET also simplifies switch design. This combination of a moderate improvement in performance with a simple implementation makes IODET an interesting alternative to consider when selecting the routing algorithm.

Chapter 9

HoL-blocking Avoidance Routing Algorithms in Direct Topologies

Authors: Roberto Peñaranda (Universidad Politécnica de Valencia), Crispín Gómez (Universidad de Castilla La Mancha), María Engracia Gómez, Pedro López, Jose Duato (Universidad Politécnica de Valencia).

Type: Conference.

Conference: IEEE International Conference on High Performance Computing and Communications (HPCC).

Location: Paris, France.

Year: 2014.

DOI: $http://dx.doi.org/10.1109/HPCC.2014.9$

URL: $http://ieeexplore.ieee.org/xpls/abs\_all.jsp?arnumber=7056591$

Abstract

Routing is a key parameter in the design of the interconnection network of large parallel computers. Depending on the number of routing options available for each packet, routing algorithms are classified into two different categories: deterministic (one available path) and adaptive (several ones). It is well-known that adaptive routing outperforms deterministic routing. Usually, adaptive routing uses virtual channels to provide routing flexibility and to guarantee deadlock freedom. On the other hand, deterministic routing is simpler and therefore has lower routing delay and does not introduce out-of-order packet delivery. This is why, in this paper, we take the challenge of developing new routing algorithms for direct topologies that exploit virtual channels in an efficient way while still maintaining the good properties of deterministic routing. This is accomplished by tackling one of the main performance degradation contributors of interconnection networks, which is the HoL-blocking effect. To do that, this paper analyzes several simple mechanisms to perform an efficient distribution of packets among virtual channels based on their destination. The resulting deterministic routing mechanisms obtain similar or even better performance than adaptive routing while keeping the simplicity of deterministic routing and guaranteeing in-order delivery of packets by design.

9.1 Introduction

The interconnection network strongly impacts the performance of large parallel computers. Latency and throughput are the key performance metrics of interconnection networks [2, 3]. Latency is the elapsed time between message injection into the network and its arrival at destination, and is the sum of two components: one related to the time required to traverse the network in absence of traffic and the other one related to the delay suffered by messages due to contention. If minimal routing is used, as commonly done, then the first latency component is constant for each source-destination pair as the number of hops does not change. The second component highly depends on the routing algorithm and the traffic load. Throughput refers to the maximum amount of data the network can deliver per time unit. The designers' goal is to minimize message latency while maximizing network throughput. To achieve this goal, the designer manipulates, among others, two main parameters [2, 3]: topology and routing. The topology provides the connection pattern among the nodes and routing decides the paths followed by messages through the network. This paper focuses on direct topologies, which are the

ones used in several machines that have occupied the topmost positions of the Top500 list of supercomputers [1].

Routing algorithms can be either deterministic or adaptive. In deterministic routing, an injected packet traverses a unique, predetermined path between source and destination. Opposite to this, adaptive routing schemes allow several paths for each source-destination pair, which, on one hand, helps to avoid congested network areas by allowing packets to take alternative paths to reach their destination, but, on the other hand, this also has a negative impact on packet contention because it increases the Head-of-Line (HoL) blocking effect. This effect occurs when a packet at the head of a queue blocks, preventing the rest of packets in that queue from advancing, even if they could do so. The HoL-blocking effect may limit the throughput of the switch up to about 58% of its peak value [60–62]. To reduce it, it is very important to isolate as much as possible those packets destined to different nodes [22, 59]. However, adaptive routing tends to spread packet destinations in all the network.

Adaptive routing algorithms typically outperform deterministic ones [3], thus improving network throughput and reducing message latency. Nevertheless, deterministic routing has other interesting properties such as an easier implementation and easier deadlock-freedom guarantee. In adaptive routing, as several paths are available for each packet, a selection function must be implemented to choose the path that will be finally used. As a consequence, routing delay is higher compared to deterministic routing. In addition, deterministic routing provides, by design, in-order packet delivery, which is important for cache coherence protocols and communication libraries, while adaptive routing requires complementary techniques to guarantee in-order delivery of packets. On the other hand, adaptive routing usually relies on virtual channels [63]. The availability of several virtual channels could be also exploited to design routing algorithms that try to reduce the HoL-blocking effect, which is the main contribution of this paper.

The rest of the paper is organized as follows. Section 9.2 introduces some background. In Section 9.3, we present some deterministic routing algorithms that use virtual channels to reduce the HoL-blocking effect, evaluating them in Section 9.4. Finally, some conclusions are drawn.

9.2 Routing in Direct Topologies

In direct topologies, nodes are organized in an n–dimensional space. The regularity of these networks greatly simplifies their deployment and the routing algorithm implementation. The

movement of a packet in a dimension does not modify the number of remaining hops in the other dimensions to reach the packet destination. The most commonly–used direct topologies are the mesh, the torus, and the hypercube. Direct topologies have been used in several of the most powerful supercomputers (see the Top500 list [1]).

In meshes and tori, the distance between current and destination nodes is calculated as the sum of the offsets in each dimension. Minimal routing algorithms will reduce one of those offsets at each routing step. The simplest minimal routing algorithm, known as dimension-order routing (DOR) [3], consists of reducing an offset to zero before considering the offset in the next dimension. For n-dimensional meshes, to enforce deadlock-freedom, DOR routes packets by crossing dimensions in strictly increasing (or decreasing) order.

However, in tori, crossing network dimensions in order is not enough to obtain a deadlock-free routing algorithm as the channel dependency graph is not acyclic [3]. Cycles are broken by splitting each physical channel into two virtual channels (VCs) [54]. More than two VCs may be used for performance improvement purposes [63]. Another technique used to avoid deadlocks in tori with deterministic routing is the bubble flow control mechanism [55]. This mechanism avoids deadlocks in each ring of the torus by ensuring that there is always an empty buffer that allows packets to advance along the ring.

Adaptive routing in meshes and tori allows packets to cross dimensions in any order. Therefore, all the minimal paths between each source-destination pair can be used by packets. However, by doing so, deadlock-freedom has to be ensured with additional mechanisms. According to [56], some VCs may be used to cross network dimensions in any order if deadlock freedom is guaranteed by providing an *escape path* to packets. This escape path is provided by means of a deadlock-free routing algorithm (for instance, DOR) in another set of VCs. With the bubble flow control mechanism, only one VC is required for escape path implementation in tori and meshes, and the remaining VCs are used for adaptive routing.

9.3 HoL-Blocking-Aware Deterministic Routing Algorithms

As mentioned above, adaptive routing provides more flexibility to move packets in the network. This routing freedom has two opposite effects. The positive one is that temporally congested

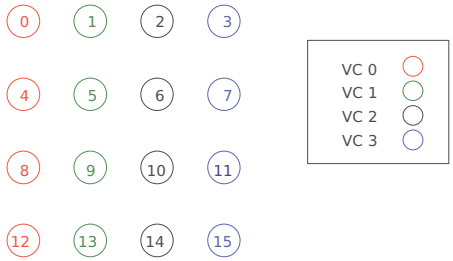


FIGURE 9.1: How DBBM assigns destinations to VCs in a 4×4 mesh with 4 VCs.

TABLE 9.1: How DBBM assigns destinations to node 0 VCs in a 4×4 mesh with 4 VCs.

Dim	VC#0	VC#1	VC#2	VC#3
X	No dest.	4: 1,5,9,13	4: 2,6,10,14	4: 3,7,11,15
Y	3: 4,8,12	No dest.	No dest.	No dest.

network areas can be avoided. However, the negative effect is that packets with different destination nodes may be highly interleaved in the switch queues, which significantly increases the HoL-blocking effect, which could degrade network performance.

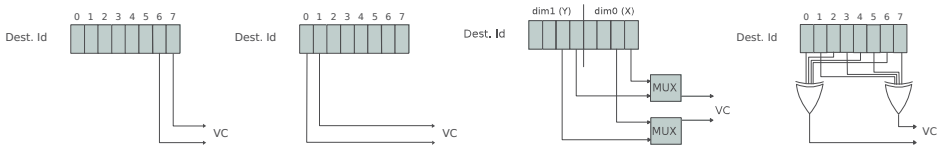
In this paper we tackle the challenge of reducing the HoL-blocking effect by taking advantage of the availability of VCs to classifying destinations among them while still maintaining the good properties of deterministic routing (in-order delivery of packets and easier implementation). The main idea is that the HoL-blocking reduction will allow the network to achieve a performance close to or even greater than adaptive routing.

The aim of reducing the HoL-blocking effect by using VCs has been pursued before by previous approaches. VOQnet [23] needs as many VCs as nodes in the network and associates each destination to a different VC. VOQnet completely removes HoL-blocking from the network, but the required number of VCs is unaffordable even in small networks since it grows linearly with the network size. However, it is used for comparison purposes since it achieves the hypothetical maximum performance that could be achieved by completely removing HoL-blocking from a network. Another option is VOQsw [24], which has as many VCs as switch output ports, and associates the set of reachable destinations through a given output port to the same VC. Therefore, VCs are selected according to the next output port the packet will use. VOQsw leads to a worse classification of packets than VOQnet and it is also not scalable, as the number of required VCs depends on switch degree.

DBBM was introduced in [25], as a scalable version of VOQnet. This mechanism selects VCs by using the destination identifier modulo the number of VCs. However, when using DBBM in a 2D torus, all the nodes in a given column are assigned to the same VC. Figure 9.1 shows

how DBBM assigns destinations with 4 VCs per physical channel. Indeed, Table 9.1 shows the assignment of destinations to VCs for node 0. For instance, VC#1 of the X-dimension is used to reach 4 nodes (1, 5, 9, 13). As it can be seen, all the VCs in the Y-dimension are never used but one per port. This lack of classification in the last dimension (Y) would lead to a huge congestion due to HoL-blocking that, at the end, could be propagated to the whole network due to upstream flow control pressure.

Regarding implementation of the VC selection mechanism, DBBM is very simple, provided that the number of VCs is a power of two. In that case, the modulo operation by the #VCs is as easy as selecting the $\log(\#VCs)$ least significant bits of the packet destination (see Figure 9.2(a)). On the other hand, notice that, as VC assignment only depends on the packet destination, packets use the same VC while it traverses the network. Therefore, VC assignment can be done once in the source node, since the rest of nodes that a packet crosses across the network merely forwards the packet through the same VC from which the packet arrived. Indeed, this fact also leads to a very low switch complexity. As there is not need to move packets from one VC to another, the internal switch of the nodes can be implemented as one independent switch per VC, instead of deploying a fully-connected crossbar. We will further analyze switch complexity later.



(a) DBBM (b) BBQ (c) IODET (d) XORDET

FIGURE 9.2: Implementation of VC selection for a 256-node 2-D network and 4 VCs.

In order to overcome the bad classification of packets in the last dimension, BBQ mechanism [26] was proposed. BBQ also uses the destination identifier to choose the VC for each packet. Indeed, it uses the destination $\log(\#VCs)$ most significant bits, see Figure 9.2(b). That is, BBQ divides the network in as many horizontal bands as VCs, in such a way that the nodes in each column are spread as much as possible among the VCs. However, the problem is that all the nodes inside each horizontal band use the same VC, and therefore they may suffer from HoL-blocking in the first dimension. As in DBBM, BBQ never changes the VC of a packet, and it can be assigned once at injection time.

In [64] we proposed the IODET mechanism, that we better explore in this paper. IODET assigns destinations to VCs considering not the whole destination identifier but the component of the

TABLE 9.2: How IODET assigns destinations to node 0 VCs in a 4×4 mesh. #VCs is 4

Dim	VC#0	VC#1	VC#2	VC#3
X	No dest.	4: 1,5,9,13	4: 2,6,10,14	4: 3,7,11,5
Y	No dest.	1: 4	1: 8	1: 12

packet destination corresponding to the dimension in which the packet is being routed. Again, the final VC to be used is obtained by performing the modulo operation, but, in this case, it considers the dimension coordinates of the destination. That is, given a packet destined to node $\{p_{n-1}, ..., p_1, p_0\}$, when routed in dimension d it will use the VC given by $p_d \bmod \#VCs$. This mechanism does a good job classifying packets compared to DBBM, as shown in Table 9.2, which shows the assignment of destinations to VCs for node 0 for 4 VCs. Almost all the VCs are used in this case, but VC#0, that is not used because the network has only 4 nodes per dimension. If the network had more than 4 nodes per dimension this VC would be also used.

Regarding implementation of VC selection, IODET is also very simple, as can be seen in Figure 9.2(c). However, as the assignment of destinations to VCs depends on the dimension the packet is traversing, it must be computed at each node (at least when there is a dimension change). Indeed, the node internal switch must allow the change in the VC assignment. Therefore, switch implementation is not as easy as DBBM one. We will analyze this issue later.

9.3.1 XORDET: XOR DETerministic Routing

In this paper, in order to overcome the lack of classification in one of the dimensions of DBBM and BBQ and, at the same time, provide a simpler switch implementation than IODET, we propose a new mechanism to assign destinations to VCs. As the previous ones do, our proposal assigns destinations to VCs by applying an arithmetic-logic transformation to the destination identifier. This transformation is required to comply with two conditions: *i)* it must be simple and *ii)* it must provide a good destinations classification. In particular, we propose to use the bitwise exclusive or (xor) operation since it provides a balanced distribution of destination nodes between the different VCs. As the resulting routing algorithm is also deterministic, the proposed mechanism will be referred to as XORDET. Destinations are assigned to VCs by performing a bitwise xor operation to the bits of the destination, as follows. Given n bits of the destination identifier and provided that there are v VCs available, $l = \log v$ bits are required to denote a virtual channel, and these bits are obtained by xoring $\frac{n}{l}$ bits of the destination, considering them in an interleaved fashion.

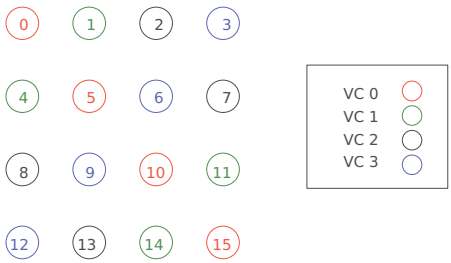


FIGURE 9.3: How XORDET assigns destinations to VCs in a 4×4 mesh with 4 VCs.

TABLE 9.3: How XORDET assigns destinations to node 0 VCs in a 4×4 mesh with 4 VCs.

Dim	VC#0	VC#1	VC#2	VC#3
X	3: 5,10,15	3: 1,11,14	3: 2,7,13	3: 3,6,9
Y	No dest.	1: 4	1: 8	1: 12

Note that by means of applying the operation to interleaved bits of the destination, the destinations are also shuffled through the VCs. Figure 9.3 shows how destination nodes are assigned to VCs for a 16-node 2D-network for 4 VCs. Packets destined to the different nodes of any row or column are assigned to different VCs in a balanced way, leading to a uniform utilization of the VCs in all the dimensions, see Table 9.3. Again, VC#0 is not used in some cases as the number of nodes per dimensions is low; it would be used in any other scenario.

In addition, XORDET implementation of VC selection is very simple. As shown in Figure 9.2(d), only some xor gates are required per source node. In particular, for v VCs, $l = \log v$ xor gates are required. Each one of them will have $\frac{n}{l}$ inputs, n being the number of bits of the destination identifier. If n is not divisible by l, some gates will have an extra input. We would like to highlight that, as in DBBM, the assignment of destinations to VCs does not change as packet travels through the network. Therefore, this assignment is performed only once when the packet is injected into the network. As a consequence, the network could be considered as several virtual independent networks, without interconnection among them, and the internal node switch can be implemented as several independent switches. As a consequence, the implementation of XORDET is very simple (as in DBBM) but it also allows maximizes the VC utilization (as in IODET).

9.3.2 Implementation issues

As stated above, the different routing algorithms demand different complexity in the internal switch of the nodes. Usually, a switch that allows connections among all input ports to all the output ports is used. In fact, such a switch is required for adaptive routing, where any input port

can forward packets to any output port. However, in the case of deterministic routing, some of the connections provided by the internal switch are unused due to routing restrictions. For instance, if DOR deterministic routing is used, a packet can only use those ports that connect to the same or higher dimensions than the one it arrived. Therefore, switch could be simplified if routing restrictions are considered, most important, without affecting performance. We will analyze in more depth switch complexity in Section 9.4.2.

Besides switch complexity, the routing complexity of selecting the VC must be taken into account for the different approaches. Adaptive routing requires the use of both a routing and a selection function. Deterministic routing only requires the routing function. In any case, both the output port and the VC to be used will be returned by the routing algorithm. Regarding the HoL-blocking aware routing algorithms considered in this paper, minor changes are required in the routing function to compute the VC to use at the output port (see Figure 9.2). In the case BBQ and DBBM, the VC to be used is obtained directly from the destination node in the packet header. For IODET, some multiplexers to select the proper bits are required in the routing function of each switch. For XORDET, a few xor gates to calculate the VC are required in the source nodes. As it can be seen, these overhead is negligible compared to the switch complexity, with no changes in router complexity and speed.

9.4 Experimental Evaluation

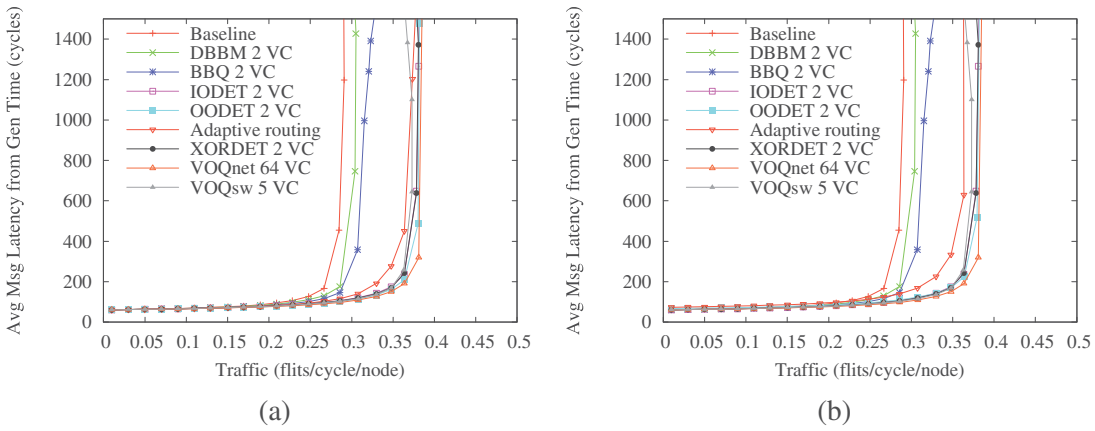


FIGURE 9.4: Avg. packet lat. vs. accepted traffic. 64-node 2D-mesh. 2 VCs are used. Uniform traffic. Routing times are scaled in (b).

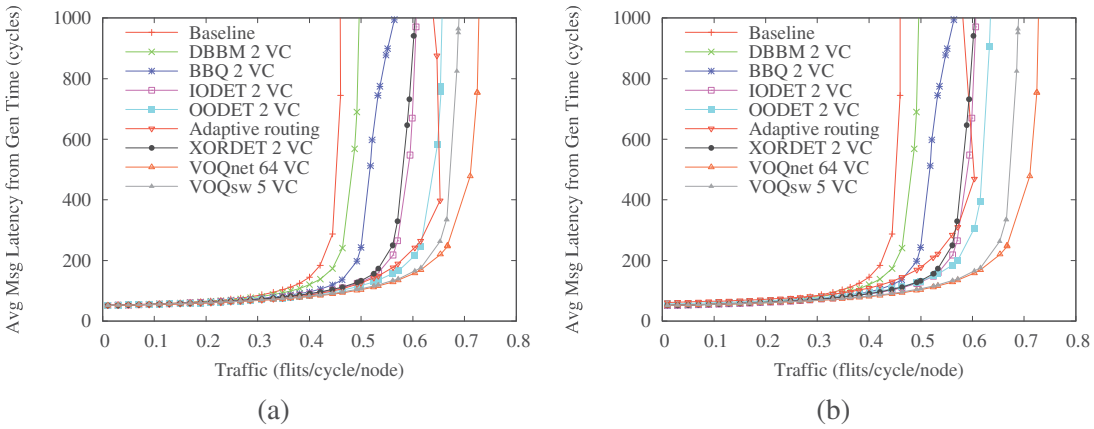


FIGURE 9.5: Avg. packet lat. vs. accepted traffic. 64-node 2D-torus. 2VCs are used. Uniform traffic. Routing times are scaled in (b).

In this section, we evaluate by simulation the deterministic routing algorithm proposed in this paper, comparing it with other routing algorithms. Deterministic routing algorithms such as IODET, DBBM, BBQ, VOQsw and VOQnet are included in the comparison, but also adaptive routing and a deterministic routing that randomly selects a VC. This latter algorithm, which is actually partially adaptive, does not guarantee in-order packet delivery. For this reason, we will refer to it as OODET (Out-of-Order DETerministic routing). To guarantee deadlock-freedom in tori, the bubble flow control mechanism is used. As adaptive routing uses several VCs per physical channel, to perform a fair comparison, we will use the same number of VCs with deterministic routing. If not stated the contrary, 2 VCs will be assumed. Different torus and meshes configurations with virtual cut-through switching are evaluated. Each node has a switch based on a full crossbar with 4-packet queues both at their input and output ports. Packet length is 16-flit. We assume a pipelined router with a latency of 4 clock cycles and switch and link bandwidth is assumed to be one flit per clock cycle. If not stated the contrary, for traffic load, a uniform distribution of message destinations is used.

9.4.1 Performance analysis

First, we focus on evaluating the mesh topology. Figure 9.4.(a) shows results for a 8×8 mesh. We have also included in the comparison the baseline deterministic routing, based on XY routing with only one VC. As it can be seen, the best results are obtained by VOQnet, which requires 64 VCs, followed by OODET. As expected, DBBM does not improve very much the performance of the baseline deterministic routing due to its poor classification of destinations. IODET and

XORDET obtains similar results. Finally, despite VOQsw requires 5 VCs, it does not obtains better performance than XORDET. Figure 9.5.(a) shows results for a 8×8 torus. As expected, the best results are obtained by VOQnet, followed by VOQsw, adaptive routing and OODET. IODET and XORDET follows closely these algorithms, obtaining a 93% of the throughput of adaptive routing, but providing in addition in-order delivery of packets[1]. Notice that VOQsw and VOQnet require a higher number of VCs (5 and 64, respectively). As expected, DBBM does not improve very much the performance over the baseline system due to its already-commented poor classification of packets, as it happens in BBQ.

TABLE 9.4: Routing times (in cycles) for adaptive and OODET.

Dimensions	Virtual channels	OODET	Adaptive
2	2	5	6
3	2	5	6
4	2	5	6
6	2	5	7
2	4	6	7
3	4	6	7
4	4	6	8
6	4	6	8
Dimensions	Virtual channels	OODET	Adaptive
2	6	7	8
3	6	7	8
4	6	7	8
6	6	7	9
2	8	7	8
3	8	7	9
4	8	7	9
6	8	7	10

However, adaptive routing is more complex, which may lead to a higher routing delay. We have used a simple model ([57, 58]) to estimate the routing delays of the different routing algorithms. This model states that the routing delay depends on the number of routing options. In particular, it is the sum of a constant time (i.e. the one required to apply the routing function) plus a component that grows logarithmically with the number of routing options. Adaptive routing would take more clock cycles to complete the routing of a packet than OODET; and OODET more than IODET, XORDET, VOQsw, VOQnet and DBBM. The more options to choose in the selection function, the more clock cycles required for routing. The effect of considering the

[1]The required reordering mechanism to provide in-order delivery of packets in the adaptive algorithms would reduce their performance if in-order delivery of packets is required [22].

complexity of the different routings is shown in Table 9.4[2]. Figure 9.4.(b) presents the same results as Figure 9.4.(a) but now scaling the routing times according to Table 9.4. In this case, we can see that adaptive routing increases its latency and achieves a lower throughput. The same happens with OODET, although at a lower extent. In torus, Figure 9.5.(b) shows the results with scaled routing times. In this case, IODET and XORDET obtain closely the same throughput as adaptive routing, but with much lower latency and providing in-order delivery of packets. In what follows, results are shown with scaled routing times to perform fairer comparison.

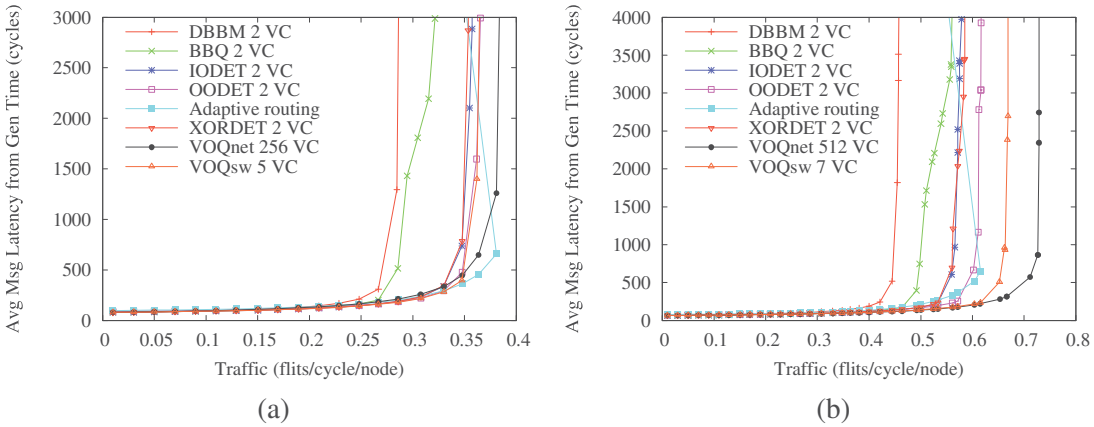


FIGURE 9.6: Average packet latency vs. accepted traffic. Uniform traffic. (a) 256-node 2D-torus and (b) 512-node 3D-torus.

We have also analyzed the impact of increasing the number of nodes per dimension (Figure 9.6.(a)). The best results are obtained by VOQnet with 256 VCs and adaptive routing, followed by VOQsw (with 5 VCs) and OODET. The best results for deterministic routing with 2 VCs are obtained by IODET and XORDET. For more network dimensions (Figure 9.6.(b)), similar conclusions can be drawn. Although the number of routing options of adaptive routing is strongly increased, the impact of routing delay kills its advantages. The case of VOQnet and VOQsw is different. As they have the highest number of VCs by design (512 and 7 VCs, respectively), they obtain better performance than any other routing algorithm.

We also analyzed the impact of the number of VCs. Figure 9.7 shows results for a 256-node 2D torus with 4 and 8 VCs, taking into account the higher routing delay of OODET and adaptive routing. As the number of VCs is increased (even with a few VCs), the throughput of all the HoL-blocking-aware deterministic routing algorithms strongly increases, being able to reach

[2]The deterministic routing algorithms have the base delay of 4 cycles.

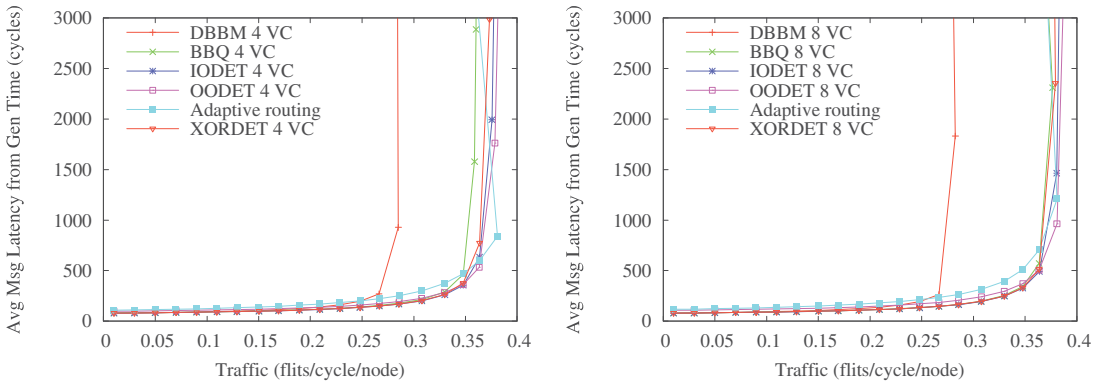


FIGURE 9.7: Avg. packet lat. vs. accepted traffic. 256-node 2D-torus. Uniform traffic. (a) 4 VCs and (b) 8 VCs.

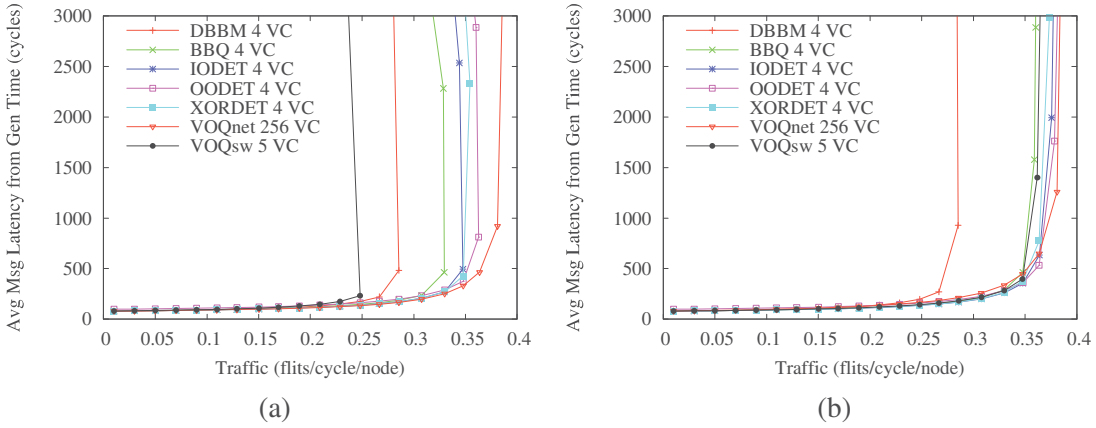


FIGURE 9.8: Avg. packet lat. vs. accepted traffic. 256-node 2D-torus and 4 VCs. Uniform traffic (a) X+Y+X-Y- routing. (b) XY routing.

the same performance as adaptive routing. But remember that they provide in-order delivery of packets by design. The exception is DBBM, which obtains a worse performance.

These results are for uniform traffic and DOR (XY) routing. However, the designer may be interested in using other routing algorithms that, for instance, provide fault-tolerance like X+Y+X-Y- [65]. Figure 9.8.(a) shows the behavior of the analyzed deterministic routing algorithms in a 256-node 2D torus with X+Y+X-Y- routing. Figure 9.8.(b) shows results with XY routing for comparison purposes. 4 VCs are used. IODET and XORDET roughly obtains the same results in both figures (although IODET shows more differences) showing that both are able to obtain a good performance with different routing algorithms. However, VOQsw and BBQ are strongly affected when changing the routing algorithm.

Next, we will analyze a scenario where the HoL-blocking awareness of the routing algorithm has a great importance. Assuming that we have uniform traffic in the network, but we introduce a hot-spot where 25% of nodes send packets only to the hot-spot node at some point of time. Traffic injection rate to the hot-spot is computed in such a way that it does not exceed the node ejection bandwidth. In particular, we assumed 0.4 flits/cycle to the hot-spot. In addition, the remaining nodes continue generating traffic with a uniform pattern. In this situation, a HoL-blocking-aware routing algorithm should be able to isolate the traffic destined to the hot-spot. Figure 9.9 shows the results for a 256-node 2D torus with 8 VCs. We can see a completely different behavior of the analyzed routing algorithms. Adaptive routing, OODET and VOQsw rapidly spread congestion as packets destined to hot-spot interferes other packets, leading to a reduction in the delivered traffic rate and strongly increasing latency. Only when the hot spot traffic disappears and after a high number of cycles, the network recovers its initial status. Notice that, after this point, traffic increases for some cycles, due to the high number of messages queued at the injection nodes. On the contrary, the HoL-blocking-aware deterministic routing algorithms have a very good behavior, close to the one of VOQnet (which requires 256 VCs). The exception is DBBM, which requires more cycles to recover from the hot-spot.

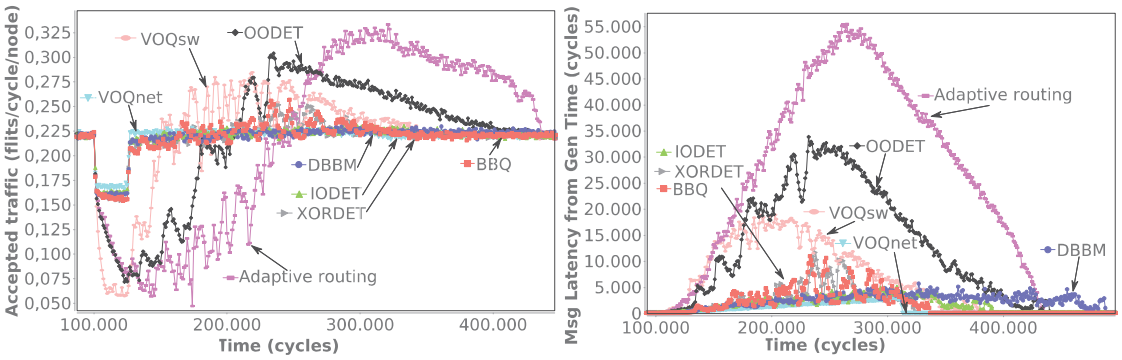


FIGURE 9.9: Results for hot-spot. 256-node 2D-torus and 8 VCs

9.4.2 Switch Cost Analysis

This section estimates the cost, measured in switching elements per switch, for the different configurations analyzed in this paper. To do so, we compare the number of required connections in the crossbar for each configuration. Several switch configurations can be used with virtual channels [63]. We will assume a fully demultiplexed crossbar to implement the internal switch of routers. Although multiplexed crossbar configurations leads to less hardware, it

requires more complex arbitration and internal speedup. Although a full crossbar (i.e. with a number of ports equal to the product of the number of physical channels by the number of VCs) is able to cope with any of the analyzed routing algorithms, some connections are not actually required, which may lead to simplify it. For instance, packets are not forwarded to the same port it arrived. Indeed, with DOR, packets may only be forwarded to dimensions higher than the one they entered the router. As a consequence, the output multiplexers corresponding to higher dimensions will have more inputs (and hence, more switching elements) than the ones located in the lower dimensions' channels. On the other hand, with adaptive routing, a packet entering through a channel may be forwarded to any other output channel. When several VCs are used and/or the number of network dimensions is high, the number of options grows considerably. In addition, the injection and ejection channels must be also taken into account. To quantify switch complexity, we will measure the number of required switching elements per switch. We assume that a $i-$input multiplexer needs i switching elements. Table 9.5 show the expressions used to compute the number of switching elements for each analyzed routing algorithm taking into account routing restrictions, n being the number of network dimensions and v the number of virtual channels per physical channel. Notice that DBBM, BBQ (both not shown) and XORDET have the same number of switching elements as XORDET because all of them rely on using virtual networks; and that VOQsw (also not shown) needs the same number of switching elements as OODET (for the same number of VCs, but VOQsw usually requires a higher number of them) because at a given dimension a packet can change the VC at each hop, like OODET. VOQnet requires the same number of connections in the switch as DBBM one, but remember that it needs as many VCs as nodes.

TABLE 9.5: Number of switching elements for each routing algorithm

Routing	Switching Elements
Adaptive	$4v^2n^2 - 2v^2n + 4vn$
OODET	$2v^2n^2 + 4vn$
IODET	$2v^2n^2 - 2v^2n + 6vn$
XORDET	$2vn^2 + 4vn$

Table 9.6 shows the number of required switching elements for adaptive routing, OODET and the four HoL-blocking-aware deterministic routing algorithms considered in this paper: IODET, DBBM, BBQ and XORDET. As it can be seen, DBBM, BBQ and XORDET requires a crossbar with fewer connections than the other configurations. That is, it requires not only a cheaper but also a simpler crossbar which may also help in reducing switch delay (this issue is not considered

TABLE 9.6: Comparison of the number of switching elements

Dimensions	Virtual channels	OODET	Adaptive	IODET	XORDET
2	1	16	-	16	16
3	1	30	-	30	30
4	1	48	-	48	48
6	1	96	-	96	96
2	2	48	64	40	32
3	2	96	144	84	60
4	2	160	256	144	96
6	2	336	576	312	192
Dimensions	Virtual channels	OODET	Adaptive	IODET	XORDET
2	4	160	224	112	64
3	4	336	528	264	120
4	4	576	960	480	192
6	4	1248	2208	1104	384
2	8	576	832	352	128
3	8	1248	2016	912	240
4	8	2176	3712	1728	384
6	8	4800	8640	4128	768

in this paper). This is due to the fact that messages traversing a given dimension cannot return to the previous dimensions, since DOR routing is used. Therefore, we could dispose of those switching elements that connect with lower dimensions. In addition to using DOR, the VC used by a given packet does not change along the path in the network, like in virtual networks. Hence, there is no need to have a crossbar connection (and the corresponding switching elements) to allow packets to perform VC transitions. Opposite to that, in OODET, the crossbar would require those connections since packets are allowed to change from one VC to another in the same dimension. Regarding IODET, the VC used by a given packet does not change along the path in the same dimension. Hence, IODET does not need the connections to switch between VCs of the same dimension, but it needs these connections when changing the dimension. Finally, adaptive routing would require a crossbar that enables almost all combinations of physical and VCs (the only exception are connections related to escape paths, which must be traversed in order).

As it can be seen, as the number of VCs is increased, the number of required switching elements further increases, specially for adaptive routing with a high number of dimensions, which more than doubles this number over XORDET.

9.5 Conclusions

This paper revisits the pros and cons of adaptive routing, analyzing alternative deterministic routing algorithms specially designed to reduce the HoL-blocking effect. In particular, we proposed a new HoL-blocking-aware deterministic routing algorithm (XORDET) for direct regular topologies. It uses virtual channels to classify packets based on their destination node. In particular, the VC used by a packet is obtained by performing bitwise xor operations to selected bits of its destination. As a result, we obtain an efficient and simple routing algorithm that takes advantage of the available VCs, keeping the good properties of deterministic routing such as in-order delivery of packets, and low routing delay. Compared to other deterministic routing algorithms that also classify packets, XORDET allows a simpler design and/or offers more routing flexibility, as its good destination classification properties are not affected by the order the underlying routing algorithm traverses dimensions. The evaluation shows that it is able to obtain performance results halfway between the baseline deterministic and adaptive routing. However, when the impact of adaptivity on routing time is considered, the differences between adaptive and deterministic routing are reduced, or even eliminated for a relatively small number of VCs. We have also shown the ability of HoL-blocking-aware routing algorithms to cope with hot-spot traffic situations, being able to isolate the packets destined to the hot-spot, reducing the interference to the rest of traffic. Furthermore, if we also consider the estimated switch cost of each routing algorithm, the proposed deterministic routing algorithm becomes absolutely the best routing option.

Chapter 10

XORAdap: A HoL-blocking aware adaptive routing algorithm

Authors: Roberto Peñaranda (Universidad Politécnica de Valencia), Crispín Gómez (Universidad de Castilla La Mancha), María Engracia Gómez, Pedro López (Universidad Politécnica de Valencia).

Type: Conference.

Conference: 23rd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP).

Location: Turku, Finland.

Year: 2015.

DOI: $http://dx.doi.org/10.1109/PDP.2015.50$

URL: $http://ieeexplore.ieee.org/xpls/abs\_all.jsp?arnumber=7092699$

Abstract

Routing is a key parameter in the design of the interconnection network of large parallel computers. Depending on the number of routing options available for each packet, routing algorithms can be deterministic (one available path) or adaptive (several ones). Adaptive routing usually outperforms deterministic routing but it also may increase the Head-of-Line blocking effect. Usually, adaptive routing uses virtual channels to provide routing flexibility and to guarantee deadlock freedom. On the other hand, deterministic routing is simpler and therefore it has lower routing delay. In this paper, we take the challenge of developing new routing algorithms for direct topologies that exploit virtual channels in an efficient way combining the good properties of both routing algorithms types: flexibility and reduced HoL blocking. To do that, this paper proposes several hybrid (combination of adaptivity and determinism) simple mechanisms to perform an efficient distribution of packets among virtual channels based on their destination. The resulting routing mechanisms are able to adapt to the different traffic patterns to obtain the best performance while keeping the simplicity of routing.

10.1 Introduction

The interconnection network strongly impacts the performance of large parallel computers. Latency and throughput are the key performance metrics of interconnection networks [2, 3]. Latency is the sum of two components: one related to the time required to traverse the network and the other one related to the delay suffered by messages due to contention. If minimal routing is used, as commonly done, then the first latency component is constant for each source-destination pair as the number of hops does not change. The second component highly depends on the routing algorithm and the traffic load. Throughput refers to the maximum amount of data the network can deliver per time unit. The designers' goal is to minimize message latency while maximizing network throughput. To achieve this goal, the designer manipulates, among others, two main parameters [2, 3]: topology and routing. The topology provides the connection pattern among the nodes and routing decides the paths followed by messages through the network. This paper focuses on direct topologies, which are the ones used in several machines that have occupied the topmost positions of the Top500 list of supercomputers [1].

Routing algorithms can be either deterministic or adaptive. In deterministic routing, an injected packet traverses a unique, predetermined path between source and destination. Opposite to this,

adaptive routing allow several paths for each source-destination pair, which, on the one hand, helps to avoid congested network areas by allowing packets to take alternative paths to reach their destination, but, on the other hand, this also has a negative impact on packet contention because it increases the Head-of-Line (HoL) blocking effect. This effect occurs when a packet at the head of a queue blocks, preventing the rest of packets in that queue from advancing, even if they could do so. The HoL-blocking effect may limit the throughput of the switch up to about 58% of its peak value [60]. To reduce HoL-blocking, it is very important to isolate as much as possible those packets destined to different nodes [22]. However, adaptive routing tends to spread packet destinations in all the network.

In general, adaptive routing algorithms outperform deterministic ones [3], thus improving network throughput and reducing message latency. Nevertheless, deterministic routing can be designed to isolate destinations in order to reduce the HoL blocking effect, which enables deterministic routing to outperform adaptive routing for some traffic patterns such as hot-spot traffic. Moreover, deterministic routing has other interesting properties such as an easier implementation and easier deadlock-freedom guarantee. In adaptive routing, as several paths are available for each packet, a selection function must be implemented to choose the path that will be finally used. As a consequence, routing delay is higher compared to deterministic routing. On the other hand, adaptive routing usually relies on virtual channels [63]. In this paper we focus on combining the good properties of adaptive (routing flexibility) and deterministic routing (reduced HoL blocking effect) to obtain routing algorithm that are able to do a good work under all the traffic pattern. The idea behind these routing algorithms is to exploit virtual channels in an hybrid way, providing some degree of adaptivity for each destination, but at the same time also determinism since the destinations are confined in a subset of virtual channels which reduces the HoL blocking effect.

The rest of the paper is organized as follows. Section 10.2 introduces some background and Section 10.3 present some deterministic routing algorithms that use VCs to reduce the HoL-blocking effect. In Section 10.4, a new HoL-blocking aware adaptive routing algorithm is proposed, evaluating it in Section 10.5. Finally, some conclusions are drawn.

10.2 Routing in Direct Topologies

In direct topologies, nodes are organized in an n–dimensional space. The regularity of these networks greatly simplifies their deployment and the routing algorithm implementation. The most commonly–used direct topologies are the mesh, the torus, and the hypercube. Direct topologies have been used in several of the most powerful supercomputers (see the Top500 list [1]).

In meshes and tori, the distance between current and destination nodes is calculated as the sum of the offsets in each dimension. Minimal routing algorithms reduce one of those offsets at each routing step. The simplest minimal routing algorithm, known as dimension-order routing (DOR) [3], consists of reducing an offset to zero before considering the offset in the next dimension. For n-dimensional meshes, to enforce deadlock-freedom, DOR routes packets by crossing dimensions in strictly increasing (or decreasing) order.

However, in tori, crossing network dimensions in order is not enough to obtain a deadlock-free routing algorithm as the channel dependency graph is not acyclic [3]. Cycles are broken by splitting each physical channel into two virtual channels (VCs) [54]. More than two VCs may be used for performance improvement purposes [63]. An alternative is the bubble flow control mechanism [55], which avoids deadlocks in each ring of the torus by ensuring that there is always an empty buffer that allows packets to advance along the ring.

Adaptive routing in meshes and tori allows packets to cross dimensions in any order. Therefore, all the minimal paths between each source-destination pair can be used by packets. However, additional mechanisms are required to avoid deadlocks. According to [56], some VCs may be used to cross network dimensions in any order if deadlock freedom is guaranteed by providing an *escape path* to packets. This escape path is provided by means of a deadlock-free routing algorithm (for instance, DOR) in another set of VCs. With the bubble flow control mechanism, only one VC is required for escape path implementation in tori and meshes, and the remaining VCs can be used for adaptive routing.

10.3 HoL-Blocking-Aware Deterministic Routing Algorithms

As mentioned above, adaptive routing provides more freedom to move packets in the network. This has two opposite effects. First, its flexibility allows avoiding temporally congested network areas. However, packets with different destination nodes may be highly interleaved in the switch

queues, which significantly increases the HoL-blocking effect, which could degrade network performance. On the other hand, deterministic routing does not have such flexibility but its HoL blocking effect is lower.

In this paper we tackle the challenge of combining flexibility and HoL blocking reduction. The idea is to confine destinations in a subset of the VCs. Thus, a given destination can use several VCs, having some degree of adaptivity, but also they are confined in a subset of the available VCs, which limits the HoL-blocking effect.

The aim of reducing the HoL-blocking effect by using VCs has been pursued before. VOQnet [23] needs as many VCs as nodes in the network and associates each destination to a different VC. It completely removes HoL-blocking from the network, but the required number of VCs is unaffordable even in small networks since it grows linearly with the network size. Another option is VOQsw [24], with as many VCs as switch output ports, and associates the set of reachable destinations through a given output port to the same VC. VCs are selected according to the next output port the packet will use. VOQsw leads to a worse classification of packets than VOQnet and it is also not scalable, as the number of required VCs depends on switch degree.

DBBM was introduced in [25] as a scalable version of VOQnet. It selects VCs by using the destination identifier modulo the number of VCs. However, in a 2D torus, all the nodes at a given column are assigned to the same VC. Its implementation is simple. Provided that the number of VCs is a power of two, the modulo operation by the #VCs is as easy as selecting the $\log(\#VCs)$ least significant bits of the packet destination. As packets use the same VC while it traverses the network, VC assignment can be done once in the source node. Moreover, as there is not need to move packets from one VC to another, the internal switch of the nodes can be implemented as one independent switch per VC, instead of deploying a fully-connected crossbar.

In order to overcome the bad classification of packets in the last dimension, BBQ [26] uses the destination $\log(\#VCs)$ most significant bits to assign destinations to VCs thus dividing the network into as many horizontal bands as VCs. However, all the nodes inside each horizontal band use the same VC and, therefore, they may suffer from HoL-blocking in the first dimension. As in DBBM, BBQ never changes the VC of a packet, and it can be assigned once at injection time.

FIGURE 10.1: Paths of source-destination pairs of the first row with Bit-reversal pattern traffic.

IODET [64] assigns destinations to VCs considering not the whole destination identifier but the component of the packet destination corresponding to the dimension the packet is being routed in. Again, the final VC to be used is obtained by performing the modulo operation, but, in this case, it considers the dimension coordinates of the destination. However, as the assignment of destinations to VCs depends on the dimension the packet is traversing, it must be computed at each node (at least when there is a dimension change). Thus, switch implementation is not as easy as DBBM one.

In XORDET [66] destinations are assigned to VCs by performing a bitwise xor operation to the destination bits. Let n be the bits of the destination identifier. With v available VCs, $l = \log v$ bits are required to denote a virtual channel, which are obtained by xoring $\frac{n}{l}$ bits of the destination, considering them in an interleaved fashion. In addition, XORDET implementation of VC selection is very simple. In particular, $l = \log v$ xor gates are required, each one with $\frac{n}{l}$ inputs. If n is not divisible by l, some gates will have an extra input. In addition, as in DBBM, the assignment of destinations to VCs does not change as packet travels through the network.

Although these deterministic routing approaches are able to isolate traffic destined to different nodes, their performance depends on the traffic pattern injected in the network. While they work very well to avoid bottlenecks caused by hot-spots, for some other traffic patterns, adaptive routing is preferred. For this reason, in this paper we propose a hybrid mechanism that is able to combine the routing flexibility provided by adaptive routing with the traffic isolation provided by HoL-blocking aware routing to obtain good performance results for all traffic patterns.

10.4 XORADAP: XOR ADAPtive Routing

As stated above, adaptive routing usually improves network performance but it does not work well for some specific traffic patterns. In particular, when there is a hot-spot node in the network, a HoL-blocking aware deterministic algorithm works better than adaptive routing [66]. The problem arises in the VCs that provide the routing flexibility (i.e., the ones that can be used

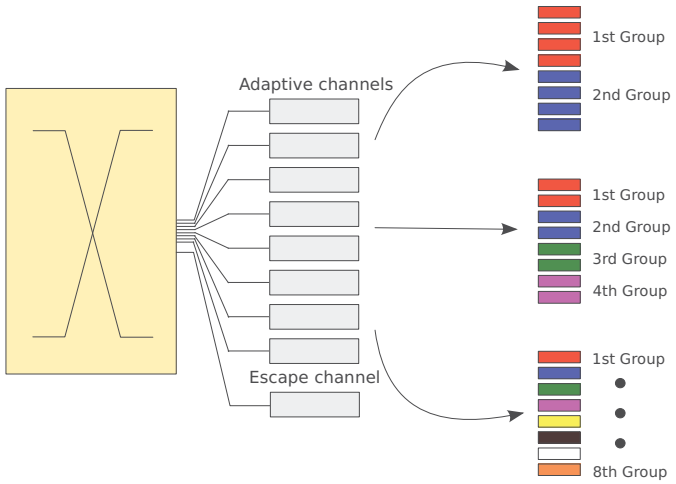


FIGURE 10.2: How XORADAP assigns groups to VCs with 8 VCs.

to cross the network dimensions without following any order). When there is a hot-spot node, adaptive routing trends to distribute traffic among all the available VCs, filling all the buffers with packets destined to the hot spot that will interfere with other traffic, thus creating the HoL-blocking problem.

On the contrary, although HoL-blocking aware routing algorithms like IODET or XORDET have a very good behavior with hot-spot traffic and even with uniform traffic pattern, they obtain a poor performance for other traffic patterns. For instance, for the bit-reversal or matrix transpose traffic patterns, a lot of source-destination pairs use the same links, leaving many unused links. This fact creates a bottleneck since many packets have to cross through the same link. We can see this behavior in Figure 10.1, which shows the paths used by the source nodes belonging to the first row in a 2D torus for the bit-reversal traffic pattern. For these kinds of traffic patterns, adaptive routing can take advantage of this unused links, providing a better utilization of the network, and, therefore, improving network performance.

In order to provide flexibility and also reduce the negative effect of HoL-blocking, we propose a new adaptive routing algorithm. This routing algorithm has several "adaptive VCs" that provide the routing flexibility and also the set of VCs that implement the escape paths. Up to this point, it is similar to the traditional adaptive routing algorithm. But, in this case, although the routing algorithm allows to cross the network dimensions following any order, we restrict the use of VCs, thus classifying the traffic depending on the destination node. To assign VCs to destinations, any mechanism could be used. In this paper, taking into account its good behaviour, we propose using the same xor function used in XORDET. For this reason, the resulting routing algorithm will be referred to as XORADAP. Therefore, we split the adaptive VCs in different groups, so

each group can be composed by 1, 2 or more VCs. Given a packet, it will be forwarded to one of the groups, selecting any of the VCs that composes the group.

Figure 10.2 shows an example of the different configurations that can be set with 9 VCs, one escape VC and 8 adaptive VCs. As the number of groups must be a power of two, in this case, three configurations are possible: 2, 4 and 8 groups. As it can be seen, the result is a set of different virtual networks, each one with several VCs and the packets of the different virtual networks are not mixed together, effectively separating flows. Notice that, if we use a group for all channels, we obtain the generic adaptive algorithm, and, if each group has only one VC, we obtain something very similar to the XORDET deterministic routing algorithm (XORADAP also provides the escape path). We use a function similar to that used in XORDET, but this time we use groups instead of VCs to classify traffic. In particular, for g groups, $l = \log g$ xor gates are required. Each one of them will have $\frac{n}{l}$ inputs, n being the number of bits of the destination identifier.

10.5 Experimental Evaluation

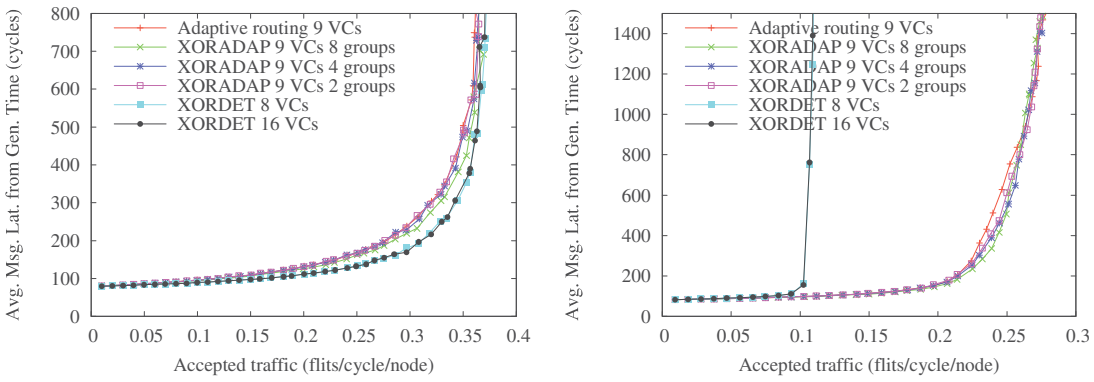


FIGURE 10.3: 256-node 2D-torus. Average packet latency vs accepted traffic: Uniform (a) and Bit-reversal (b) traffic patterns.

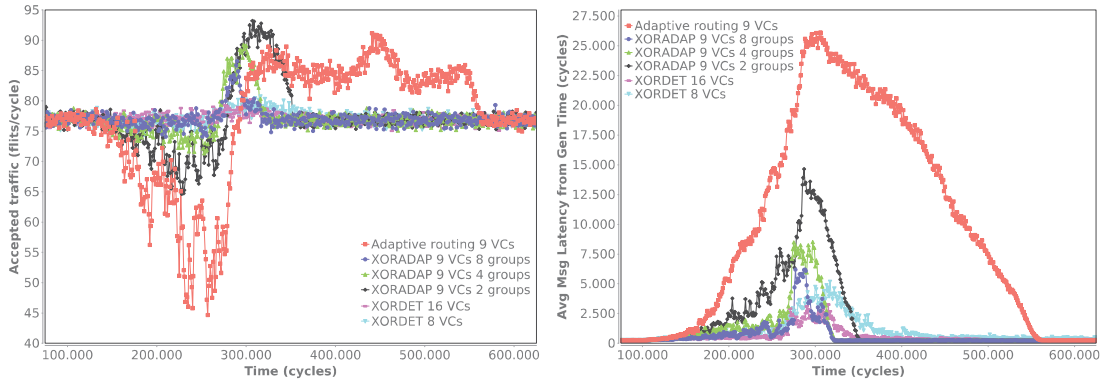
In this section, we evaluate by simulation the HoL-blocking aware adaptive algorithm proposed in this paper, XORADAP. We compare it with XORDET [66], which obtained the best results among the HoL-blocking aware deterministic algorithms (IODET, DBBM, BBQ...). Also, a pure adaptive routing algorithm is evaluated and compared. To guarantee deadlock-freedom in tori, the bubble flow control mechanism was used. Regarding the number of VCs per physical

channel, it must be a power of two in XORDET. In XORADAP, the number of VCs must be a power of two plus one (the escape channel). To perform a fair comparison, traditional adaptive routing will use the same number of VCs as the one used in XORADAP. Each node has a switch based on a full crossbar with 4-packet queues both at their input and output ports. Packet length is 16-flit. We assume a 4-stage pipelined router, and switch and link bandwidth is assumed to be one flit per clock cycle.

First, we analyze network performance for the uniform traffic pattern. Figure 10.3.(a) shows the results for a 2-D torus with 16 nodes per dimension. 9 VC (8 adaptive channels and 1 escape channel) were used in adaptive and XORADAP routing algorithms, and 8 VCs and 16 VCs in XORDET. In XORADAP, three different configurations were tested: two groups with 4 VCs each, 4 groups with 2VCs and 8 groups with only one VC per group. We can see that all of them achieve roughly the same throughput. Regarding latency, for medium to high traffic rates, the more routing flexiblity leads to higher latency values, due to the HoL-blocking effect generated by interfering traffic flows. Therefore, we can see how the XORADAP with more groups, less adaptive, has a lower latency; and both configurations of XORDET obtain the lowest latency values.

The behavior for bit-reversal traffic pattern is shown in Figure 10.3.(b), where there is a bottleneck when using deterministic routing. In this case, XORDET obtains a lower throughput than any adaptive routing algorithm, even using more VCs. In particular, adaptive routing achieves almost 3X throughput than deterministic routing. Any of the configurations of XORADAP helps to reach such performance. The poor behavior of XORDET, and, in general, of any deterministic routing, is due to the unbalanced distribution of traffic, which leads to an over-utilization of some links while other are unused.

As the results presented up to now show, XORADAP achieves one of its design goals. It is as good as adaptive routing for the traffic patterns the latter works well, improving XORDET, and deterministic routing in general (not shown). Now, we will analyze a different scenario, where the HoL-blocking awareness of the routing algorithm has a great importance. Assume that we have uniform traffic in the network, but we introduce a hot-spot, where 25% of nodes send packets only to one node (the hot-spot node) at some point in time. So, we have two traffic flows: 75% of nodes generate ing packets with an uniform pattern and 25% generating packets destined to the hot-spot. In this situation, a HoL-blocking-aware routing algorithm should be able to isolate the traffic destined to the hot-spot, thus avoiding interfering the other



(a) (b)

FIGURE 10.4: 256-node 2D-torus. Uniform traffic with hot-spot. Accepted traffic (a) and average packet latency (b) vs. time.

flows. On the other hand, a pure adaptive algorithm mixes the different flows, spreading the possible congestion to the whole network. This behavior can be seen in Figure 10.4 for a 256-node 2-D torus. As it can be seen, for adaptive routing, as soon as traffic to the hot-spot is injected, delivered traffic reduces and packet latency strongly increases. Only when the hot spot traffic disappears (at cycle 260.000 approx.) and after a high number of cycles, the network with adaptive routing recovers its initial performance. On the other hand, XORDET performs a good isolation of the hot-spot traffic flows, and, thus, limits the congestion it generates. Delivered traffic is almost unaltered and latency is only slightly increased. Regarding the number of VCs per physical channel, as expected, the greater the better performance, as hot-spot flows affect less other flows. Regarding XORADAP, it achieves a behavior halfway between adaptive and XORDET routing. XORADAP configurations with more groups can better isolate the hot-spot traffic flows, obtaining a result very similar to XORDET in the best case (with 8 groups).

The analysis shown before demonstrates that XORADAP achieves its second design goal. It is as good as a HoL-blocking aware deterministic routing algorithm classifying and isolating traffic, thus improving adaptive routing. Therefore, summarizing, the hybrid routing algorithm proposed in this paper, XORADAP, combines the best features of both adaptive routing and HoL-aware deterministic routing, being an interesting design option to consider when choosing the routing algorithm.

10.6 Conclusions

This paper proposes a new HoL-blocking-aware adaptive routing algorithm (XORADAP) for direct regular topologies. This routing algorithm tries to combine the best features of both adaptive (flexibility) and deterministic routing algorithms specially designed to reduce the HoL-blocking effect (traffic isolation). To do so, it uses virtual channels to classify packets based on their destination node while it allows to use different paths to arrive to the destination. In particular, virtual channels are split into groups, and the routing algorithm can select any virtual channel of a group, like in adaptive routing; and the group used by a packet is obtained as a function of its destination, like in HoL-blocking aware deterministic routing. In particular, in this paper, the group is selected by performing bitwise xor operations to selected bits of its destination, but other strategies are also possible. As a result, we obtain a hybrid routing algorithm that takes advantage of the HoL-blocking awareness, while keeping the good properties of adaptive routing such as flexibility to route. The evaluation results show that XORADAP obtains similar performance results than either deterministic or adaptive routing with uniform traffic pattern. Most important, it is the best analyzed routing option for these complementary scenarios: i) it achieves a similar behavior to adaptive routing for those traffic patterns where routing flexibility is required to avoid bottlenecks, and ii) it is able to cope with hot-spot traffic situations, being able to isolate the packets destined to the hot-spot, reducing the interference to the rest of traffic, like HoL-blocking aware deterministic routing does.

Chapter 11

XOR-based HoL-blocking reduction Routing Mechanisms for Direct Networks

Authors: Roberto Peñaranda, Crispín Gómez, María Engracia Gómez, Pedro López (Universidad Politécnica de Valencia).

Type: Journal.

Journal: Parallel Computing.

Publisher: Elsevier.

ISSN: 0167-8191.

State: Under review.

Impact Factor: 1.511

JRC ranking: Q1 (2014)

Abstract

Routing is a key design parameter in the interconnection network of large parallel computers. Routing algorithms are classified into two different categories depending on the number of routing options available for each source-destination pair: deterministic (there is one path available) and adaptive (there are several ones). Adaptive routing has two opposed effects on network performance. On one hand, it provides routing flexibility that may help on avoiding a congested network area, thus improving network performance. On the other hand, it also may increase the Head-of-Line blocking effect due to more destination nodes sharing the port queues. Usually, adaptive routing uses virtual channels to provide routing flexibility and to guarantee deadlock freedom. Deterministic routing is simpler, which implies lower routing delay and it introduces less Head-of-Line blocking effect. In this paper, we propose an adaptive and HoL-blocking reduction routing algorithm for direct topologies that tries to combine the good properties of both worlds: It provides routing flexiblity but also reduces the Head-of-Line blocking effect. To do that, this paper proposes several functions which use the XOR operation to efficiently distribute the packets among virtual channels based on their destination node. The resulting routing mechanisms have different properties depending on whether they enforce routing flexibility or Head-of-Line blocking reduction.

11.1 Introduction

A key component in the performance of large parallel computers is the interconnection network. Performance of these systems is increasingly determined by how data is communicated among the huge number of computing resources. Latency and throughput are the key performance metrics of interconnection networks [2, 3]. Latency is the elapsed time between message injection into the network and its arrival at its destination, and it is the sum of two components, one related to the time required to traverse the network in absence of traffic (base latency) and the other one related to the delay suffered by messages due to contention. If minimal routing is used, as commonly done, then the base latency is constant for each source-destination pair as the number of hops does not change. The second component of latency depends on network contention. Throughput refers to the maximum amount of data the network can deliver per time unit. The main goal is to minimize message latency while maximizing network throughput. To achieve this goal, we have to consider, among others, two main parameters [2, 3]: topology and

routing. The topology provides the connection pattern among the nodes. This paper focuses on direct topologies, which are one of the options used to build large parallel machines. In fact, several machines that have occupied the topmost positions of the Top500 list of supercomputers [1] are based on direct topologies, like the ones that occupy the 3rd, 4th and 5th positions in the June 2016 list.

The routing algorithm decides the paths followed by messages through the network. A routing algorithm can be either deterministic or adaptive. In deterministic routing, an injected packet traverses a unique, predetermined path between each source-destination pair. Opposite to this, adaptive routing schemes allow several paths for each source-destination pair. This, on the one hand, helps avoiding congested network areas by allowing packets to take alternative paths to reach their destination. However, this flexibility has a negative impact on packet contention because it may increase the Head-of-Line (HoL) blocking effect. This effect occurs when a packet at the head of a queue blocks, and prevents the rest of packets in that queue from advancing, even if they could do so because the required resources are free. The HoL-blocking effect may be highly pernicious and may limit the throughput of the switch up to about 58% of its peak value [60–62]. In order to reduce the HoL-blocking effect, it is very important to isolate as much as possible those packets destined to different nodes [22, 59]. However, adaptive routing tends to spread packet destinations all over the network, which may have a very negative effect when a destination is saturated since it will spread the congestion to other network areas and prevent more packets to arrive to other non-saturated destinations.

Adaptive and deterministic routing algorithms have different properties. While adaptive routing algorithms outperform deterministic ones [3] for some traffic patterns because of their routing flexibility, thus improving network throughput and reducing message latency deterministic routing better isolates destinations reducing the HoL blocking effect, which enables deterministic routing to outperform adaptive routing for some other traffic patterns such as traffic with hot-spot destinations. Moreover, adaptive routing usually leads to a more complex implementation and it is more deadlock-prone [73, 74]. Adaptive routing usually relies on the use of virtual channels (VCs) [63] to avoid deadlocks. On the other hand, adaptive routing requires a selection function to choose the path that will be finally used, as several paths are available for each packet. As a consequence, routing delay for adaptive routing is usually higher compared to deterministic routing [58, 75, 76].

In this paper we focus on combining the good properties of adaptive (routing flexibility) and deterministic routing (reduced HoL blocking effect) to design a hybrid routing algorithm. The idea behind this routing algorithm is to take advantage of virtual channels, usually used in adaptive routing to provide flexibility, but with a revisited aim: confining destinations in subsets of virtual channels in order to reduce the HoL blocking effect while providing some degree of flexibility. In order to select the VCs that can be used by a given packet, the proposed routing algorithm uses a XOR function of the destination identifier which provides a balanced usage of VCs for all traffic patterns. A deterministic version of the proposed routing algorithm was presented in [66] and a first version of the adaptive routing algorithm based on the XOR function was published in [77]. The current paper unifies both proposals under a common framework, explaining in more depth how the use of the XOR operation helps reducing the Head-of-Line blocking effect both for deterministic and adaptive routing. This paper also include new performance evaluation results.

The rest of the paper is organized as follows. Section 11.2.1 introduces some background on routing in direct topologies. In Section 11.2.2, we present some previous deterministic routing algorithms that use virtual channels to reduce the HoL-blocking effect. In Section 11.3.1, we present the XOR-based HoL-blocking reduction deterministic routing algorithm. And, in Section 11.3.2, we extend the proposal of Section 11.3.1 to propose the HoL-blocking reduction adaptive routing algorithm that is able to combine the benefits of deterministic and adaptive routing algorithms. These algorithms are evaluated in Section 11.4. Finally, some conclusions are drawn.

11.2 Background

11.2.1 Direct Topologies

A direct network consists of a set of nodes, each one being directly connected to a subset of other nodes in the network. The most popular direct topologies organize nodes in an orthogonal n–dimensional space. The regularity of these networks greatly simplifies their deployment and routing algorithm implementation. The movement of a packet in a dimension does not modify the number of remaining hops in the other dimensions to reach the packet destination. The most commonly–used direct topologies are the mesh, the torus, and the hypercube. These topologies have been used in several of the most powerful supercomputers (see the Top500 list [1]).

The distance between source and destination nodes is computed as the sum of the offsets in each dimension. Minimal routing algorithms will reduce one of those offsets at each routing step. The simplest minimal routing algorithm, known as dimension-order routing (DOR) [3], consists of reducing an offset to zero before considering the offset in the next dimension. For n-dimensional meshes, to enforce deadlock-freedom, DOR routes packets by crossing dimensions in strictly increasing (or decreasing) order.

However, in tori, crossing network dimensions in order is not enough to obtain a deadlock-free routing algorithm as the channel dependency graph is cyclic [3]. Cycles are broken by splitting each physical channel into two virtual channels (VCs) [54]. More than two VCs may be used for performance improvement purposes [63]. Another technique used to avoid deadlocks in tori with deterministic routing is the bubble flow control mechanism [78]. This mechanism avoids deadlocks in each ring of the torus by ensuring that there is always an empty buffer that allows packets to advance along the ring.

Many adaptive routing algorithms have been published in the literature [79–82]. Fully adaptive routing [3, 56] in meshes and tori allows packets to reduce dimension offsets following in any order. Therefore, all the minimal paths between each source-destination pair can be used by packets. However, this may introduce cycles and deadlock-freedom has to be ensured with additional mechanisms. According to [56], VCs may be used to cross network dimensions in any order if deadlock freedom is guaranteed by providing an *escape path* to packets. This escape path is provided by means of a deadlock-free routing algorithm (for instance, DOR) in another set of VCs. Notice that with the bubble flow control mechanism, only one VC is required for escape path implementation in tori and meshes, and the remaining VCs can be used for adaptive routing.

11.2.2 Related Work

As mentioned in the previous section, adaptive routing provides flexibility in the path followed by packets and in the use of VCs since it uses VCs with complete freedom, except the ones used as escape channels. This routing freedom has two opposite effects over performance. The positive one is that temporally congested network areas can be avoided and therefore, for some traffic patterns, packets can make a better use of the network resources. However, the negative effect is that packets with different destination nodes may be highly interleaved in the switch queues, which significantly increases the HoL-blocking effect with hot-spot traffic

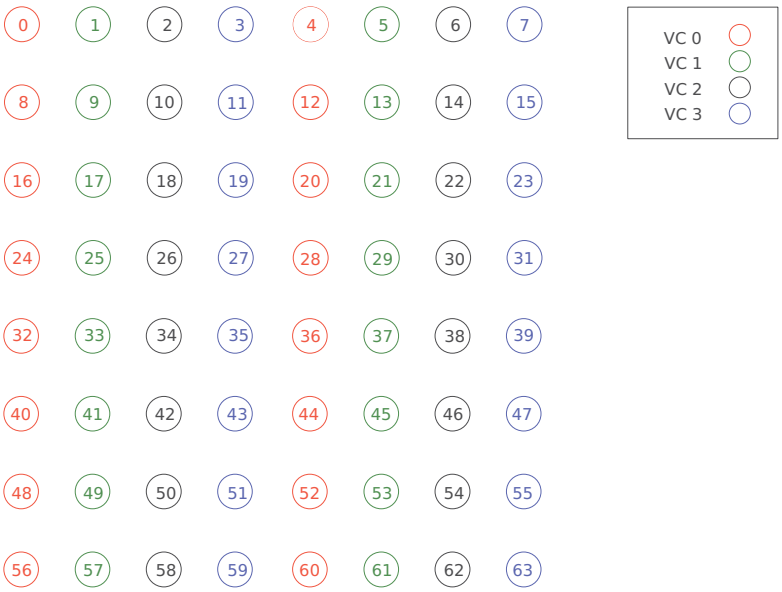


FIGURE 11.1: How DBBM assigns destinations to VCs in a 8×8 mesh with 4 VCs.

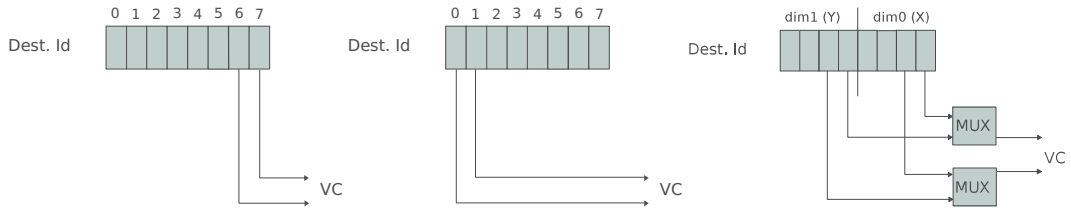
patterns, leading to degrade network performance for all the network (as we can see in Section 11.4). On the other hand, deterministic routing does not provide that flexibility, which may negatively affect performance for some traffic patterns, but its contribution to the HoL-blocking effect is lower.

The idea of reducing the HoL-blocking effect by using VCs has been pursued before by previously proposed deterministic routing algorithms. The key idea behind these proposals is to classify destinations into VCs, according to some criteria. Virtual Output Queueing at network level (VOQnet) [23] needs as many VCs as nodes in the network and associates each destination to a different VC. VOQnet completely removes the HoL-blocking effect from the network, but the required number of VCs is unaffordable even in small networks since it grows linearly with the network size. However, it is often used for comparison purposes since it provides an upper bound that could be achieved by completely removing HoL-blocking from a network. Another option is Virtual Output Queueing at switch level (VOQsw) [24], which requires as many VCs as switch output ports, and associates the set of reachable destinations through a given output port to the same VC. Therefore, VCs are selected according to the next output port the packet will use. VOQsw leads to a worse classification of packets than the one obtained with VOQnet and it is also not scalable, as the number of required VCs depends on switch degree.

Destination-Based Buffer Management (DBBM) was introduced in [83], as an attempt to obtain a scalable version of VOQnet. This mechanism selects VCs by using the destination node identifier modulo the number of VCs. While it works for other topologies, when using DBBM

Dim	VC#0	VC#1	VC#2	VC#3
X	8	16	16	16
Y	7	No dest.	No dest.	No dest.

TABLE 11.1: How many destinations DBBM assigns to node 0 VCs in a 8×8 mesh with 4 VCs.



(a) DBBM (b) BBQ (c) IODET

FIGURE 11.2: Implementation of VC selection for a 256-node 2D network and 4 VCs.

in a 2D mesh or torus, all the nodes in a given column are assigned to the same VC, as shown in Figure 11.1 for an 8x8 mesh with 4 VCs per physical channel. Indeed, Table 11.1 shows the number of destinations assigned to different VCs for a node of the network (node 0). For instance, VC#0 of the X-dimension is used to reach 8 nodes (the 4^{th} row), while VC#1 of the X-dimension is used to reach 16 nodes (the 1^{st} and 5^{th} row). As it can be seen, all the VCs in the Y-dimension are never used but one per port. This lack of classification in the last dimension (Y) lead to congestion due to the HoL-blocking effect that, at the end, could be propagated to the whole network due to upstream flow control pressure.

If we analyze the implementation complexity of the VC selection mechanism, DBBM is very simple, provided that the number of VCs is a power of two. This mechanism uses the modulo operation by the #VCs and its implementation is as easy as selecting the $\log(\#VCs)$ least significant bits of the packet destination (see Figure 11.2(a)). Additionally, notice that, as VC assignment depends only on the packet destination, packets use the same VC while it traverses the network. This is a nice feature, as VC assignment can be done once at the source node, and the rest of nodes that a packet crosses across the network merely forwards the packet through the same VC from which the packet arrived to, like in virtual networks [84], without requiring VC transitioning [54]. Furthermore, this fact also leads to a reduced switch complexity. As there is not need to move packets in a switch from one input VC to another output VC, the internal switch of the nodes can be implemented as one independent switch per VC, instead of deploying a fully-connected crossbar. We will further analyze switch complexity later considering all these aspects.

Dim	VC#0	VC#1	VC#2	VC#3
X	8	16	16	16
Y	1	2	2	2

TABLE 11.2: How IODET assigns destinations to node 0 VCs in a 8×8 mesh. #VCs is 4

Band-Based Queuing (BBQ) mechanism [26] was proposed in order to overcome the bad classification of packets in the last dimension of DBBM. BBQ also uses some bits of the destination identifier to choose the VC for each packet, but opposite to DBBM, BBQ uses the destination $\log(\#VCs)$ most significant bits (see Figure 11.2(b)). That is, BBQ divides the network in as many horizontal bands as VCs, in such a way that the nodes in each column are distributed as much as possible among the VCs. However, the problem is that all the nodes inside each horizontal band use the same VC, and, therefore they may suffer from HoL-blocking in the first dimension. As in DBBM, BBQ never changes the VC of a packet during its path in the network. It can be assigned once at injection time keeping the same VC along its path in the network.

In-Order DETerministic routing (IODET) [64] follows a different approach and it selects the VC by considering not the whole destination identifier but the component of the packet destination corresponding to the dimension in which the packet is being routed. The VC to be used by a packet is obtained by performing the modulo operation of the dimension coordinates of the destination. That is, given a packet destined to node $\{p_{n-1}, ..., p_1, p_0\}$, when routed in dimension d it will use the VC given by $p_d \bmod \#VCs$. This mechanism does a better job classifying packets than DBBM as can be seen in Table 11.2, which shows the number of destinations assigned to different VCs for node 0 for an 8x8 mesh with 4 VCs. As it can be seen, all the VCs in both dimensions are used when applying IODET to distribute destination among VCs.

Considering the implementation complexity of the VC selection, IODET is also very simple, as displayed in Figure 11.2(c) which shows an example for 4 VCs. As it can be seen, the least significant bits of the component for each dimension is used to select the VC. However, as the assignment of destinations to VCs depends on the dimension the packet is traversing, the VC is changed when there is a dimension change and, therefore, in those nodes the new VC to use must be computed. As a consequence, the node internal switch must allow the change in the VC assignment and the switch implementation is not as easy as the DBBM one. We will analyze this issue later.

11.3 XOR-based HoL-blocking Reduction Routing

In this section, we present a mechanism to assign destinations to VCs based on the use of XOR function. We apply this destination distribution to two different routing algorithms: first a deterministic routing mechanism (XORDET) [66] is designed, and then an adaptive version (XORADAP) [77] is proposed. The idea of this second algorithm is to combine the good properties of both, deterministic and adaptive routing to adapt to any traffic pattern to obtain optimal performance results.

11.3.1 XORDET: XOR DETerministic Routing

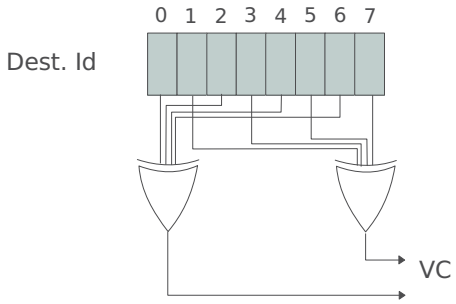
XORDET is a deterministic routing algorithm that reduces the HoL-blocking effect and performs a balanced distribution of destinations among VCs. For doing that, opposite to the previously presented deterministic algorithms that use a subset of the node destination bits, XORDET distributes destinations among VCs by performing a bitwise XOR operation to all the bits of the destination node, as follows. Assume that there are v VCs available. Then, $l = \log_2 v$ bits are required to denote a virtual channel. If destination identifiers are n bits long, then, for each destination, the VC to use is obtained by performing l XOR operations in parallel. In each XOR operation $\frac{n}{l}$ bits of the destination are XORed, taking them in an interleaved fashion. In particular, given a packet destined to node $\{p_{n-1}, ..., p_1, p_0\}$, it will use the VC given by the bits $\{VC_{l-1}, ..., VC_1, VC_0\}$, computed as follows:

$$VC_0 = p_0 \oplus p_{0+l} \oplus p_{0+2l} \cdots \oplus p_{0+(\frac{n}{l}-1)l}$$

$$VC_1 = p_1 \oplus p_{1+l} \oplus p_{1+2l} \cdots \oplus p_{1+(\frac{n}{l}-1)l}$$

$$VC_{l-1} = p_{l-1} \oplus p_{l-1+l} \oplus p_{l-1+2l} \cdots \oplus p_{l-1+(\frac{n}{l}-1)l}$$

Figure 11.3 shows how the VC selection will be implemented in a network with 4 VCs and 8-bit node identifiers. Each VC bit is obtained by XORing 4 bits of the node destination identifier in an interleaved fashion. Notice that XORDET implementation of VC selection is very simple, as only some XOR gates are required per source node. In particular, for v VCs, $l = \log v$ XOR gates are required. Each one of them will have $\frac{n}{l}$ inputs, n being the number of bits of the destination identifier. If n is not divisible by l, some gates will have an extra input. Notice that this implementation assumes that the number of VCs is a power of two. On the other hand, we would like to highlight that, as in DBBM, the assignment of destinations to VCs does not



(a) XORDET

FIGURE 11.3: Implementation of VC selection in XORDET for a 256-node 2D network and 4 VCs.

FIGURE 11.4: How XORDET assigns destinations to VCs in a 8×8 mesh with 4 VCs.

change as packet travels through the network. Therefore, this assignment can be performed only once when the packet is injected into the network. As a consequence, the network could be considered as several virtual independent networks, without interconnection among them, and the internal node switch can be implemented as several independent switches. As a consequence, the implementation of XORDET is very simple (as in DBBM) but, as will we shown in Section 11.4, it also allows the VCs to maximize its utilization (as in IODET).

XORDET is able to isolate traffic destined to different nodes and also balancing destinations among VCs. Figure 11.4 shows how destinations are distributed among VCs in a network with 64 nodes and 4 VCs. As it can be seen, XORDET does a very good job, as traffic destined to either rows or columns will be distributed among the VCs. Table 11.3 shows how many destinations are assigned to each VC of node 0 of the network.

Dim	VC#0	VC#1	VC#2	VC#3
X	14	14	14	14
Y	1	2	2	2

TABLE 11.3: How many destinations XORDET assigns to node 0 VCs in a 8×8 mesh with 4 VCs.

As shown, XORDET balances node destinations among VCs which will balance traffic for uniform random traffic pattern. But XORDET is a deterministic routing algorithm and this means that, for some adversarial traffic patterns, it may suffer from performance drops due to the limited routing flexibility. While XORDET works very well to avoid congestion caused by hot-spots, for some other traffic patterns, adaptive routing is able to outperform deterministic routing in general and, in particular, XORDET. For this reason, in this paper we propose an adaptive HoL-blocking reduction routing algorithm that is able to combine the routing flexibility provided by adaptive routing with the destination isolation provided by HoL-blocking reduction routing to obtain optimal performance results for all traffic patterns.

11.3.2 XORADAP: XOR ADAPtive Routing

As mentioned above, deterministic routing lacks flexibility to adapt to some adversarial traffic patterns while adaptive routing does not encourage HoL-blocking effect reduction, which is very important for some traffic patterns. In particular, with a hot-spot node in the network, a HoL-blocking reduction deterministic algorithm that isolates the hot-spot traffic works better than adaptive routing [66] that spreads that traffic over the network avoiding other traffic to progress in the network. Let us analyze what happens in this case.

With adaptive routing, the problem arises in the VCs of all the network dimensions that provide the routing flexibility (i.e. the ones that can be used to cross the network dimensions without following any order). When there is a hot-spot node, adaptive routing trends to distribute traffic among all the available VCs, filling all the buffers with packets destined to the hot-spot node. Those packets will interfere with other traffic flows all over the network, thus creating the HoL-blocking problem.

On the contrary, HoL-blocking reduction algorithms like IODET or XORDET have a very good behavior with hot-spot traffic because they confine the hot-spot traffic in just one of the VCs, allowing the rest of traffic to progress normally across other VCs. These routing algorithms also work well with uniform random traffic pattern as it will be shown, but they obtain a poor

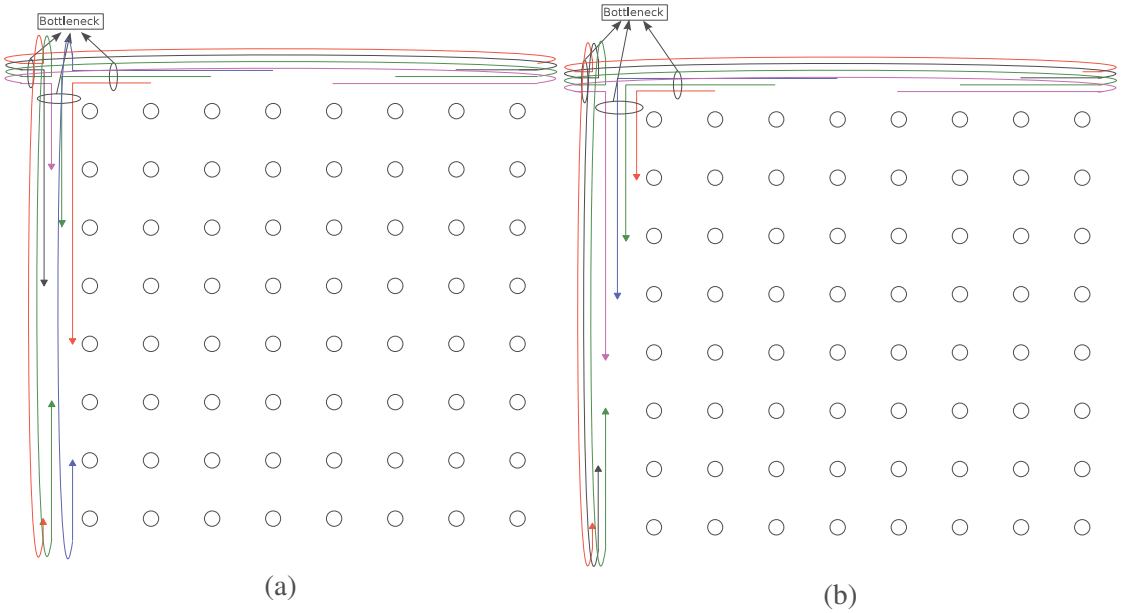


FIGURE 11.5: Paths of source-destination pairs of the first row with different pattern traffics: (a) Bit-reversal and (b) Matrix Transpose.

performance for some adversarial traffic patterns. For instance, consider the bit-reversal or matrix transpose traffic patterns [3]. In these cases, if deterministic routing is used, a lot of source-destination pairs will use the same links due to the destination distribution leaving many links unused. This fact creates a bottleneck since many messages have to cross the same link. We can see this behavior in Figure 11.5. This figure shows the paths used by the source nodes belonging to the first row in a 2D torus for the bit-reversal and matrix transpose traffic patterns using a deterministic routing algorithm. In particular, DOR was used. As it can be observed in the figure, the links of the topmost leftmost node become a bottleneck with deterministic routing. For these kinds of traffic patterns, adaptive routing can take advantage of all the network resources, providing a better utilization of the network links and therefore, improving overall network performance for this adversarial traffic pattens.

In order to provide flexibility for adversarial traffic patterns and also reduce the negative effects of HoL-blocking, we propose an adaptive HoL blocking-reduction routing algorithm. In this routing algorithm, VCs are organized as in a fully adaptive routing algorithm: there is a *group* of adaptive VCs that can be used without restrictions and also there is an escape channel. We assume that bubble flow control is used in the escape channel. However, this routing algorithm confines each node destination identifier in a subset of the adaptive VCs instead of allowing the use of any of them. Contrary to deterministic routing, the routing algorithm allows crossing the

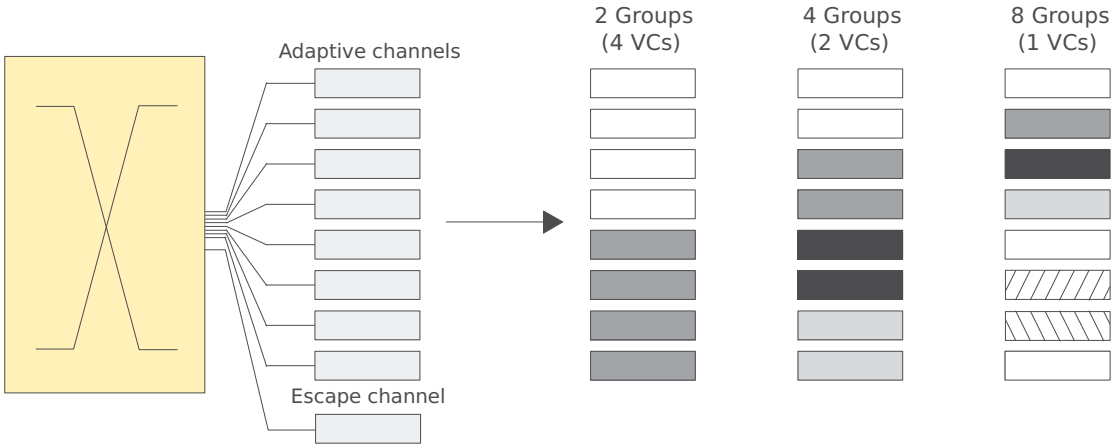


FIGURE 11.6: How XORADAP may assign VCs to groups with 8 VCs.

network dimensions following any order (and therefore allowing more flexibility) but restricting the use of VCs depending on the destination node and thus confining the congested destinations in some VCs and allowing the packets located in the rest of VCs to progress. As a consequence, it provides some degree of flexibility but, at the same time, it limits the impact of the HoL-blocking effect because only a subset of the VCs can be used for a given destination.

To assign destinations to VCs, any mechanism could be used. In this paper, taking into account its good balancing behavior, we propose to use a variant of the XOR function which is used in XORDET. For this reason, the resulting routing algorithm will be referred to as XORADAP (XOR ADAPtive). The VC assignment works as follows. We split the adaptive VCs into several groups. Each group can be composed of 1, 2 or more VCs. Given a packet, it will be forwarded to one of those groups depending on the packet destination, and any of the VCs of that group could be used.

As mentioned above, we use a function similar to the one used in XORDET, but, in this case, we select a group of VCs for each destination instead of just a single VC to classify traffic. In particular, with g groups of VCs, $l = \log g$ XOR gates are required. Given a packet destined to node $\{p_{n-1}, ..., p_1, p_0\}$, it will use the VC group given by the bits $\{VCG_{l-1}, ..., VCG_1, VCG_0\}$, computed as follows:

$$VCG_0 = p_0 \oplus p_{0+l} \oplus p_{0+2l} \cdots \oplus p_{0+(\frac{n}{l}-1)l}$$

$$VCG_1 = p_1 \oplus p_{1+l} \oplus p_{1+2l} \cdots \oplus p_{1+(\frac{n}{l}-1)l}$$

$$VCG_{l-1} = p_{l-1} \oplus p_{l-1+l} \oplus p_{l-1+2l} \cdots \oplus p_{l-1+(\frac{n}{l}-1)l}$$

Each of the XOR gates will have $\frac{n}{l}$ inputs, n being the number of bits of the destination identifier. As in XORDET, the $\frac{n}{l}$ inputs come from interleaved bits in the destination identifier. Notice that the number of groups og VCs must be a power of two.

As stated above, each group is composed of several VCs. Several configurations can be used. If there are V_a VCs available for adaptive routing, each group may contain from 1 to V_a virtual channels. Notice that, if we use only one group with all the virtual channels, we obtain the generic fully adaptive algorithm. On the contrary, if each group has only one VC, we obtain an adaptive version of XORDET that allows packets to cross dimensions following any order. Figure 11.6 shows an example of the different configurations that can be set for 9 VCs, one escape VC and 8 adaptive VCs. In this case, three configurations are possible for XORADAP: 2, 4 and 8 groups (with 4, 2 and 1 VC per group, respectively). As it can be seen, the resulting network is a set of different virtual networks, each one with several VCs. This means that packets of the different virtual networks are not mixed together, effectively separating flows. The escape channel is used by all the groups of virtual networks.

11.3.3 Implementation issues

As stated above, the different routing algorithms analyzed in this paper demand different implementation complexity in the internal switch of the nodes. A fully demultiplexed crossbar [63] provides the highest flexibility, allowing connections among all input VCs to all the output VCs (i.e. it can map any input VC onto any output VC). In fact, such a switch is required for adaptive routing, where any input port can forward packets to any output port. However, in the case of deterministic routing, some of the connections provided by the internal switch are unused due to routing restrictions. For instance, if DOR deterministic routing is used, a packet can only use those ports that connect to the same or higher dimensions than the one it arrived. Therefore, the switches could be simplified if routing restrictions are considered, most important, without affecting performance.

Let us consider the routing algorithms proposed in this paper. In XORDET, as the VC is selected as a function of the destination node, packets do not change the VC while they travel across the network, thus leading to a even simpler internal switch design than the traditional deterministic routing with the same number of VCs. In XORDET, VC assignment can be done once at the source node, and the rest of nodes that a packet crosses across the network merely forwards the packet through the same VC from which the packet arrived to. Traditional deterministic routing

with multiple VCs would have to select the output VC to forward the packet. As a consequence, as there is no need to move packets in a switch from one input VC to another output VC, the internal switch of the nodes can be implemented as one independent switch per VC (i.e. several virtual networks), instead of deploying a fully-connected crossbar, which is cheaper and faster, as switch delay depends on the number of switch ports [58, 75, 76].

In the same way, XORADAP also simplifies switch implementation. In this case, packets may change the VC used but they do not change the assigned group of VCs. The internal switch of the nodes can be implemented as one independent switch per group of VCs. Therefore, we could use a simpler internal switch design than fully adaptive routing. Notice that for a configuration of one group of all of the adaptive VCs, the complexity will be the same as fully adaptive routing.

We will analyze in more depth switch complexity for different routing algorithms in Section 11.4.2.

Concerning routing mechanics, deterministic routing only requires applying the routing function [3] while adaptive routing requires the use of both the routing and the selection function [3]. In any case, both the output port and the VC to be used will be returned by the routing algorithm. For both XORDET and XORADAP, a few XOR logic gates are required at the source nodes to compute the corresponding VC or group of VCs, respectively. In XORADAP, a selection function is also required to select the VC inside the assigned group. However, the number of routing choices is smaller than with fully adaptive routing. As routing delay depends on the number of routing choices [58, 75, 76], XORADAP may lead to a faster implementation than fully adaptive routing.

11.4 Experimental Evaluation

In this section, we evaluate by simulation the HoL-blocking reduction routing algorithms described in this paper, XORADAP and XORDET comparing them with previously proposed ones. We used a simulation environment developed at our research group. A prior version of this tool was used to provide evaluation results in [3]. First, we will compare XORDET with other HoL-blocking reduction deterministic algorithms like IODET, DBBM, BBQ, VOQnet and VOQsw. We will also consider a fully adaptive routing algorithm and a DOR deterministic routing which allows packets to use all the VCs of the selected dimension, that is a deterministic routing algorithm without destination node classification. Notice that this latter algorithm is

actually partially adaptive (as it allows several routing options) and does not guarantee in-order delivery of packets. For this reason, we will refer to it as Out of Order DETerministic routing (OODET). To guarantee deadlock-freedom in tori, the bubble flow control mechanism was used (either in all the VCs for deterministic routing or in the escape VC for fully adaptive routing).

After the evaluation of XORDET, we will evaluate XORADAP to analyze how it behaves under different traffic patterns and we will show how a hybrid approach is able to combine the best of two worlds and obtain good performance results for any traffic pattern.

Regarding the number of VCs per physical channel, it must be a power of two in XORDET. In XORADAP, the number of groups of VCs must be a power of two and also an escape channel is required. To perform a fair comparison, for traditional fully adaptive routing, we will use the same number of VCs as the one used in XORADAP. Each node has a switch based on a full crossbar with 4-packet queues both at their input and output ports. Packet length is 16 flits. We assume a pipelined router with a latency of 4 clock cycles, and switch and link bandwidth is assumed to be one flit per clock cycle. Source nodes implement VOQnet in the injection. This means that messages with different destinations do not harm the injection of each other. We have modeled different network sizes with different number of dimensions: 64 and 256 nodes with 2 dimensions and 512 nodes with 3 dimensions.

Regarding network traffic, we have considered several widely-used synthetic traffic patterns [3]: uniform random, matrix transpose, and bit-reversal. In addition, as we are interested in analyzing the impact of the HoL-blocking effect, we also evaluated a hot-spot traffic pattern, whose parameters will be described in detail later.

11.4.1 Performance analysis

11.4.1.1 XORDET evaluation

First, we will analyze the behavior of XORDET versus the other HoL-blocking reduction deterministic routing algorithms. Figure 11.7 shows the obtained results for a 2D torus with 256 nodes and uniform random traffic pattern. With only a few number of VCs (4 or 8), any HoL-blocking algorithm is able to reach nearly the same performance as VOQnet, which is the upper bound. The exception is DBBM that, due to its poor destination classification in the last dimension (see Section 11.2.2), it obtains a worse performance. Notice the importance of the

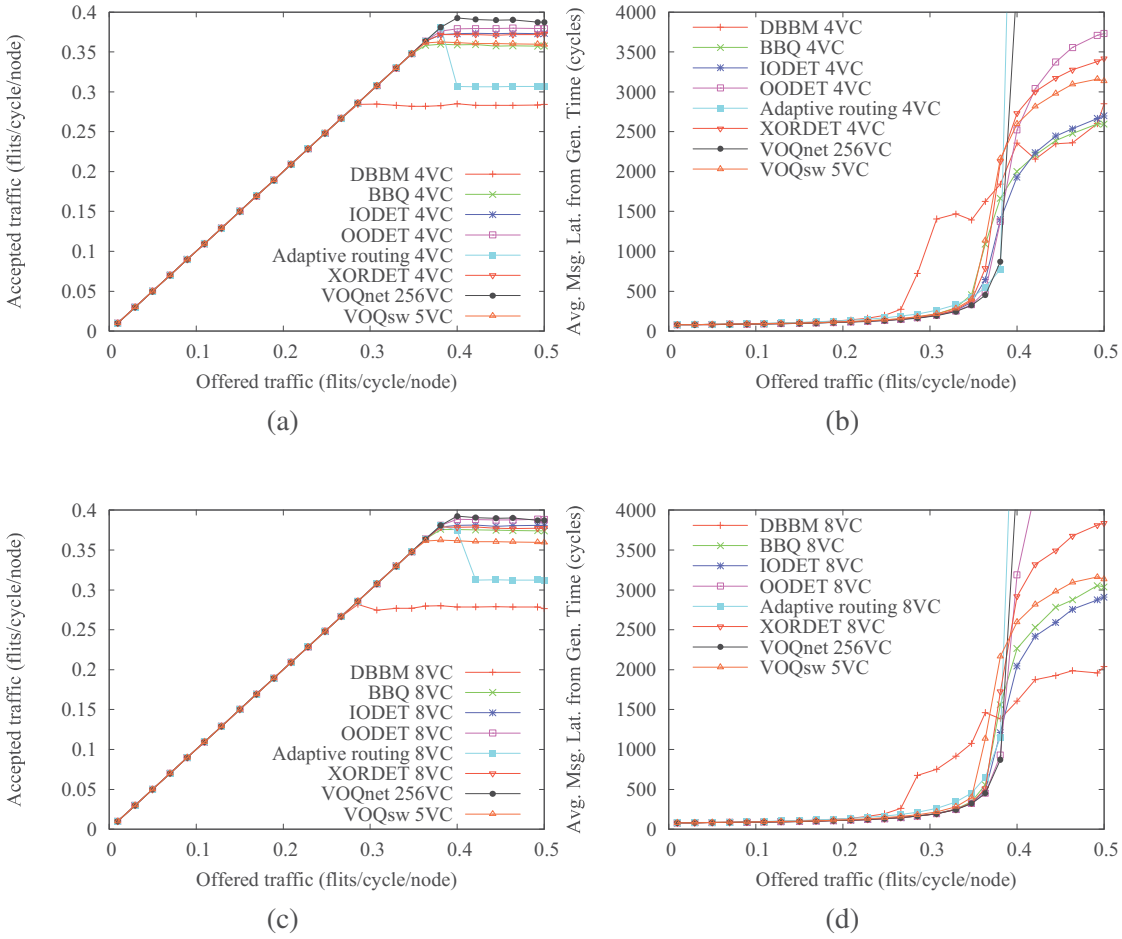


FIGURE 11.7: Average packet latency and accepted traffic vs offered load. 256-node 2D-torus. Uniform random traffic pattern. (a,b) 4 VCs and (c,d) 8 VCs.

dimension ordering followed by the routing algorithm. Unexpectedly, BBQ, which also considers a subset of the node destination identifier bits to classify packets, works quite well. The difference between DBBM and BBQ is that the former consider the least significant bits while the latter considers the most significant ones. As XORDET considers all the node destination bits to classify packets, it should not be affected by changes in the dimension ordering followed by the routing algorithm. On the other hand, fully adaptive routing suffers the typical performance rollback after network saturation [85]. Figure 11.8 shows the results for a 3D torus with 512 nodes and uniform random traffic. As it can be seen, results are qualitatively the same.

In Figure 11.7, the traditional DOR routing following XY order was used. However, using other deterministic routing algorithms could be interesting. For instance, in [65], X+Y+Z+X-Y-Z- direction-order routing was proposed instead of dimension order routing for fault tolerance purposes. Direction order routing allows packets to be routed in both directions of a dimension

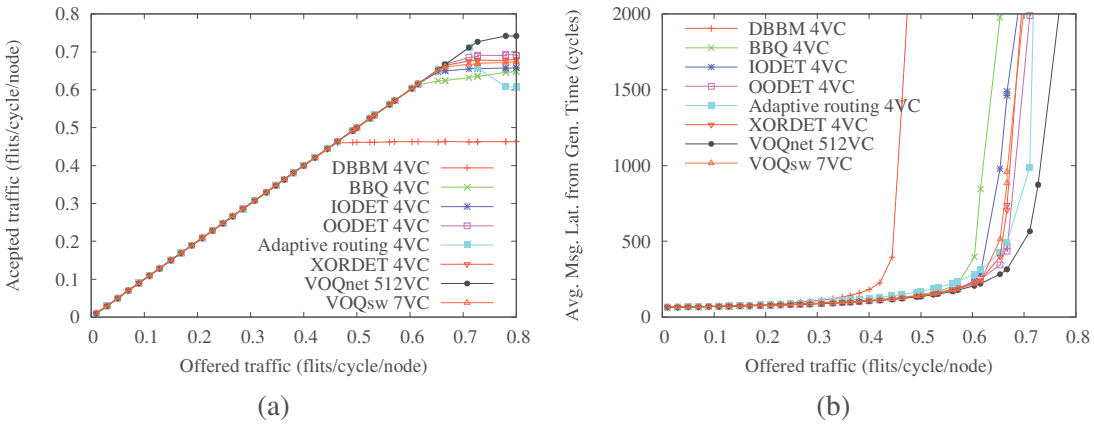


FIGURE 11.8: Average packet latency and accepted traffic vs offered load. 512-node 3D-torus. Uniform random traffic pattern. 4VCs.

and, therefore, offers greater flexibility to avoid faults. Furthermore, direction order routing allows routing through non-minimal paths. For 2D networks, the X+Y+X-Y- direction order routing counterpart would be used. Packets are routed following an ascending dimension order, but taking first the positive dimension directions, and then the negative ones. Figure 11.9 shows evaluation results for the different HOL-blocking reduction mechanisms but using X+Y+X-Y- with minimal paths as the baseline deterministic routing. We can see how the fact of traversing dimensions in a different order changes the behavior of some algorithms like BBQ, which drives down its performance significantly. This effect is similar to the one produced by DBBM before and is due to the fact of using a subset of the destination node identifier bits to select the VC to use. The dimension ordering followed by the routing algorithm may generate an unfair use of the VCs, overloading some of them while others are barely used. However, XORDET or IODET, which consider all the destination node identifier bits, are less affected by the change in the routing algorithm and obtain roughly the same performance as the one obtained with XY routing.

After analyzing the behavior of the different algorithms under uniform random traffic pattern, next we analyze them under other traffic patterns. First, we will analyze a scenario where the HoL-blocking reduction ability of the routing algorithm may have a great impact. Assume that we have uniform random traffic pattern in the network, but we also introduce a hot-spot node: 25% of network nodes send packets only to one node (the hot-spot node) during some period of time. Traffic injection rate to the hot-spot is computed in such a way that it does not exceed the node ejection bandwidth (1 flit/cycle). The hot-spot traffic starts at clock cycle

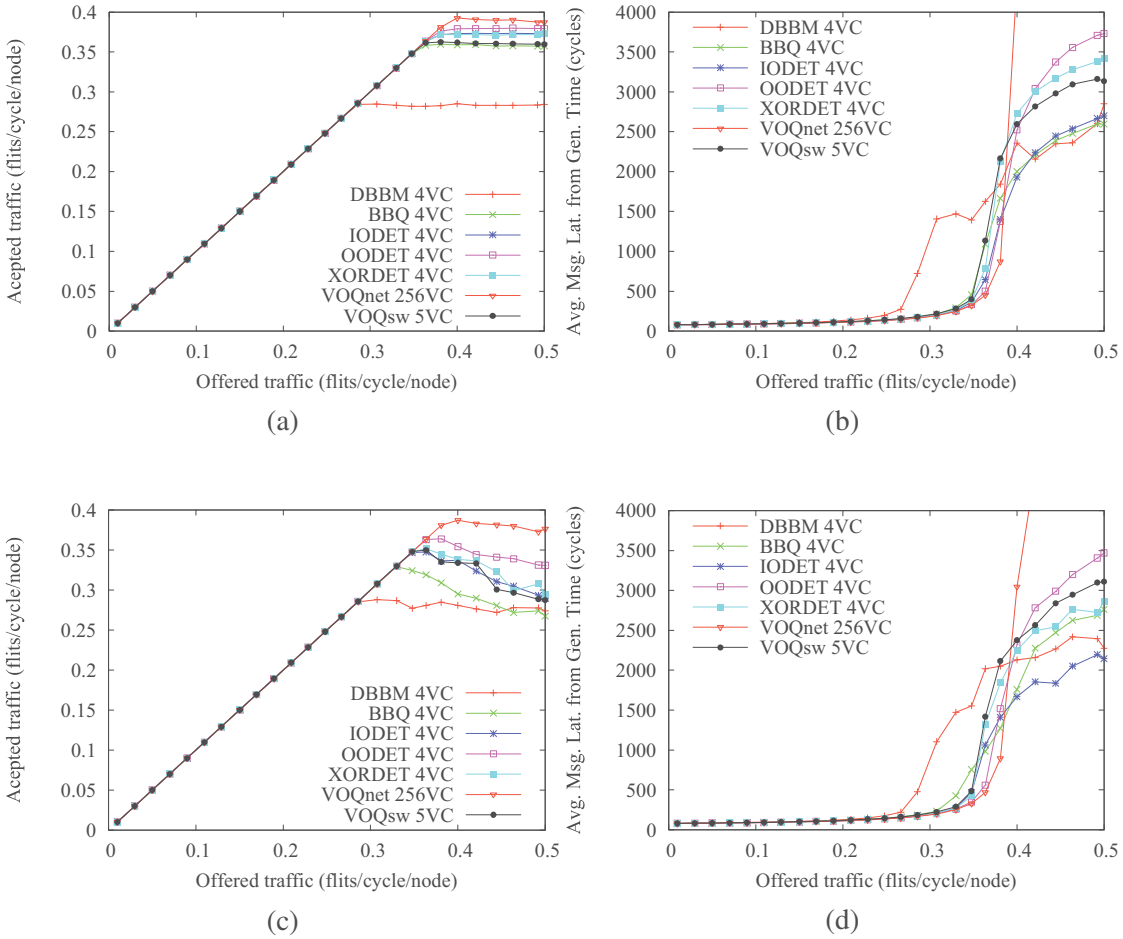


FIGURE 11.9: Average packet latency and accepted traffic vs offered load. 256-node 2D-torus. Uniform random traffic pattern. (a,b) XY routing and (c,d) X+Y+X-Y- routing. 4 VCs.

100,000 and is active until a number of packets (10,000 in our experiment) have been delivered. This corresponds to clock cycle 260,000. In addition, the remaining nodes (75%) continue generating traffic following a uniform random traffic pattern, that is, sending packets to all the destinations except the hot-spot node. Therefore, during this period of time, the network has two traffic flows: 75% of nodes generate packets with an uniform random traffic pattern and 25% generate packets destined to the hot-spot node. In such a situation, a HoL-blocking reduction routing algorithm should be able to isolate the traffic destined to the hot-spot (i.e. hot flows), thus avoiding interfering the other flows (i.e. cold flows). On the other hand, a fully adaptive algorithm mixes the different flows, spreading the possible congestion to the whole network. To perform this experiment, we have implemented large injection queues at source nodes so that they always can queue a packet if the packet cannot be injected into the network.

This scenario is evaluated in Figure 11.10 for a 256-node 2D torus. We can see a completely

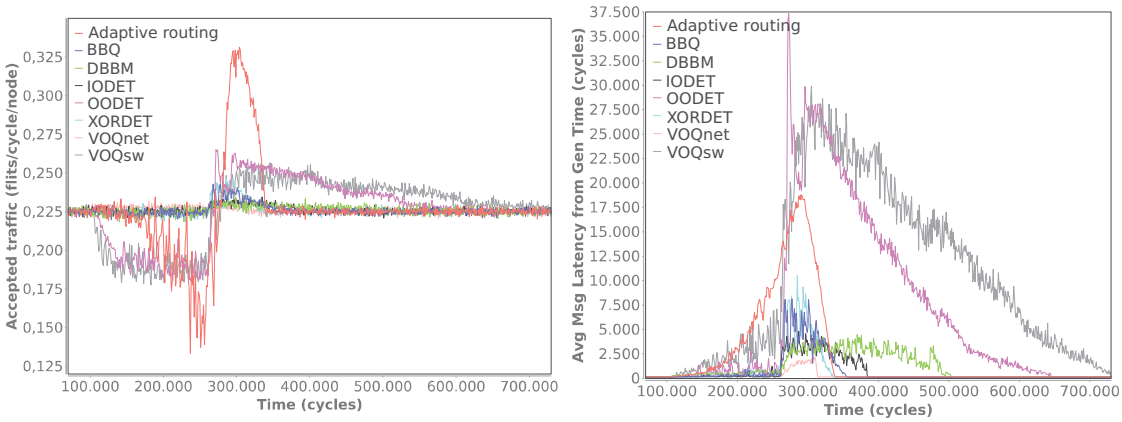


FIGURE 11.10: Results for hot-spot. 256-node 2D-torus and 8 VCs.

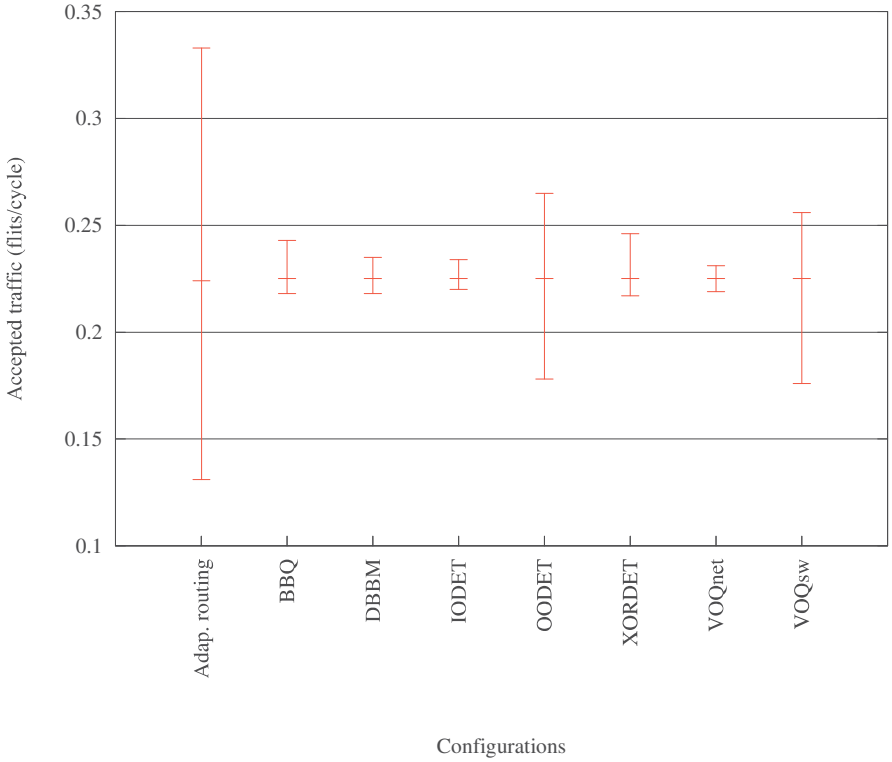


FIGURE 11.11: How the hot-spot traffic affects the different routing algorithms.

different behavior of the analyzed routing algorithms. On the one hand, fully adaptive routing, OODET and VOQsw rapidly spread congestion as packets destined to the hot-spot node interferes other packets, leading to a high reduction in the delivered traffic rate (Figure 11.10.(a)) and strongly increasing latency (Figure 11.10.(b)). Only when the hot-spot traffic disappears and after a high number of cycles, the network recovers. Notice that, after the hot-spot traffic is removed, accepted traffic increases for some cycles, due to the high number of messages that

have been queued at the injection nodes.

On the other hand, the HoL-blocking reduction deterministic routing algorithms show a much better behavior, close to the one of VOQnet (which requires 256 VCs) without impacting the network throughput and latency in spite of the hot-spot traffic. The exception is DBBM, which requires a higher number of cycles to recover from the hot-spot traffic. This is because the injected uniform random traffic pattern is on the edge of saturation in DBBM. For a better understanding of this behavior, Figure 11.11, shows the maximum, minimum and average values of accepted traffic in Figure 11.10.(a). The average value represents the accepted traffic corresponding to uniform random traffic pattern, when it is not affected by the hot-spot. The minimum value is reached when the hot-spot is active. And, finally, the maximum value is the one reached after the end of the hot-spot traffic, where queued messages at the injection nodes begin to be received. The closer the three plotted values, the better the behavior of the routing algorithm as it is less affected by the hot-spot traffic. We can see how fully adaptive routing, OODET and VOQsw are strongly affected, obtaining a minimum value more distant to the average value than the remaining routing algorithms.

To summarize, XORDET and IODET were able to reach (with only a few VCs) the same performance as fully adaptive algorithm for uniform random traffic pattern (see Figure 11.7). Contrary to DBBM and BBQ, they are less affected by changes in the routing algorithm (i.e. the order in which dimensions are crossed, see Figure 11.9) and they are able to efficiently isolate the hot-spot traffic (see Figure 11.10). The advantage of XORDET versus IODET is that it is simpler to implement at the internal switch. Remember that XORDET uses virtual networks, but IODET performs VCs changes in the network, which requires additional internal switch connections.

11.4.1.2 XORADAP evaluation

We will first analyze XORADAP with uniform random traffic pattern. Figure 11.12 shows the results (8 VCs for fully adaptive routing and 1 VC for the escape path). For XORADAP, we selected three different configurations with 9 VCs: two groups with 4 VCs each, 4 groups with 2 VCs and 8 groups with only one VC per group. Remember that more groups of VCs leads to a better packet classification but a lower routing flexibility. Regarding fully adaptive routing, we used the same number of VCs as XORADAP for the sake of fairness. As the number of VCs in XORDET must be a power of two, we evaluated it by using both 8 and 16 VCs.

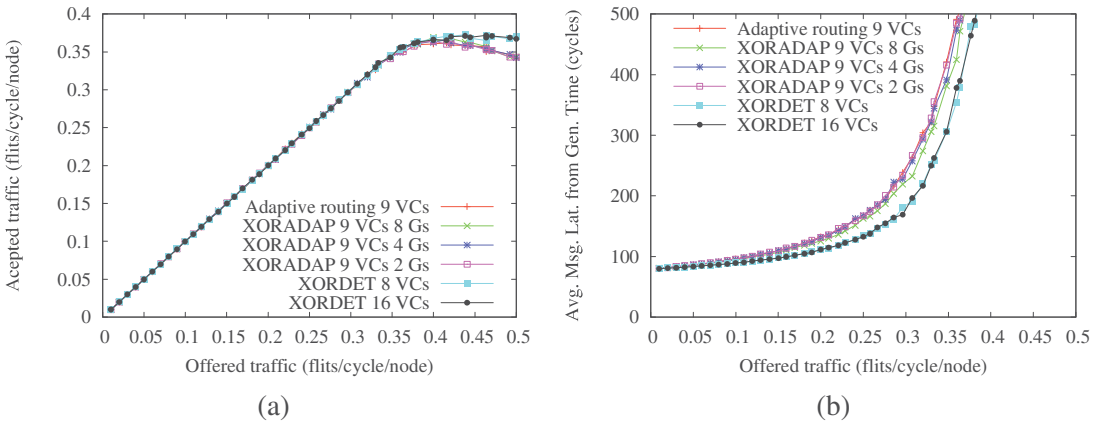


FIGURE 11.12: 256-node 2D-torus. Uniform random traffic pattern. (a) Accepted traffic and (b) average packet latency vs. offered traffic.

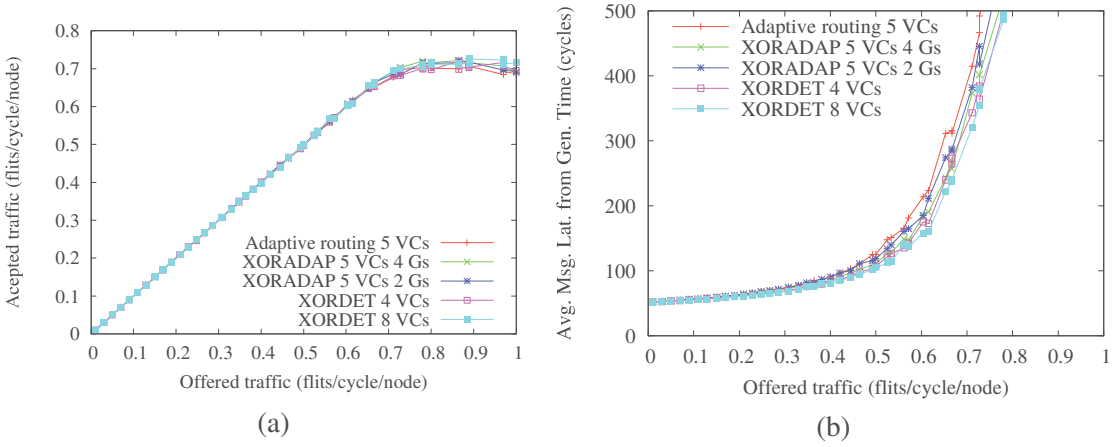


FIGURE 11.13: 64-node 2D-torus. Uniform random traffic pattern. (a) Accepted traffic and (b) average packet latency vs. offered traffic.

We can see that all the routing algorithms evaluated obtain roughly the same throughput. Notice, though, that the fully adaptive algorithms suffer a performance degradation after its saturation point [85]. Regarding latency, (Figure 11.12.(b)), for medium to high traffic rates (i.e. 0.3 flits/cycle/node), a higher routing flexibility (i.e., fully adaptive routing or XORADAP with less number of groups of VCs) leads to higher latency values due to the HoL-blocking effect generated by interfering traffic flows. We can see how the XORADAP routing algorithm with more groups of VCs, less adaptive behavior, obtains a slightly lower latency. Both configurations of XORDET obtain the lowest latency values, with almost no differences between them.

Let us analyze networks with different geometry. Figure 11.13 shows some results for a smaller network with a lower number of nodes per dimension (64-node 2D torus). Figure 11.14 shows

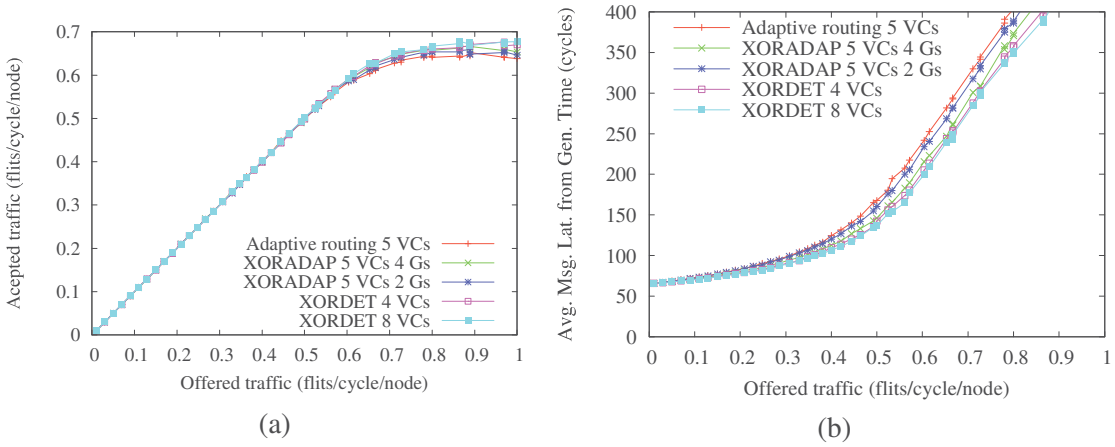


FIGURE 11.14: 512-node 3D-torus. Uniform random traffic pattern. (a) Accepted traffic and (b) average packet latency vs. offered traffic.

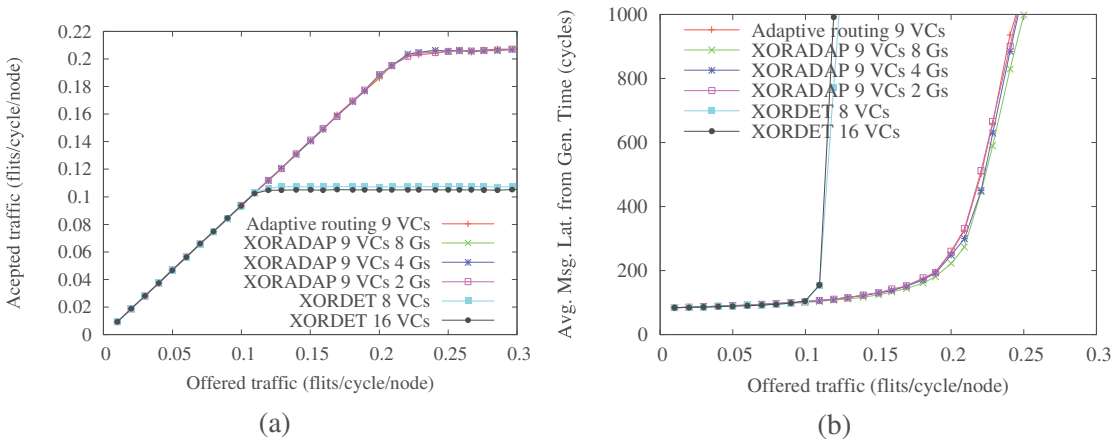


FIGURE 11.15: 256-node 2D-torus. Matrix transpose traffic. (a) Accepted traffic and (b) average packet latency vs. offered traffic.

results for a larger network with a higher number of dimensions (512-node 3D torus). In addition, we also tested a different number of virtual channels per physical channel. In particular, 5 VCs (4 adaptive channels plus 1 escape channel) were used for XORADAP and fully adaptive routing. In this case, for XORADAP, we have two groups with 2 VCs each and 4 groups with only one VC per group. 4 VCs and 8 VCs were used in XORDET. As it can be seen, in both cases, the network shows the same behavior we saw in the 256-node 2D torus. As expected, network throughput is higher in these configurations, since we have 8 nodes per dimension instead of 16 and, thus, a better bisection bandwidth. However, the results are qualitatively the same obtained for the 256-node network: more routing flexibility (i.e. fully adaptive routing or XORADAP with a low number of groups of VCs) leads to slightly higher latency.

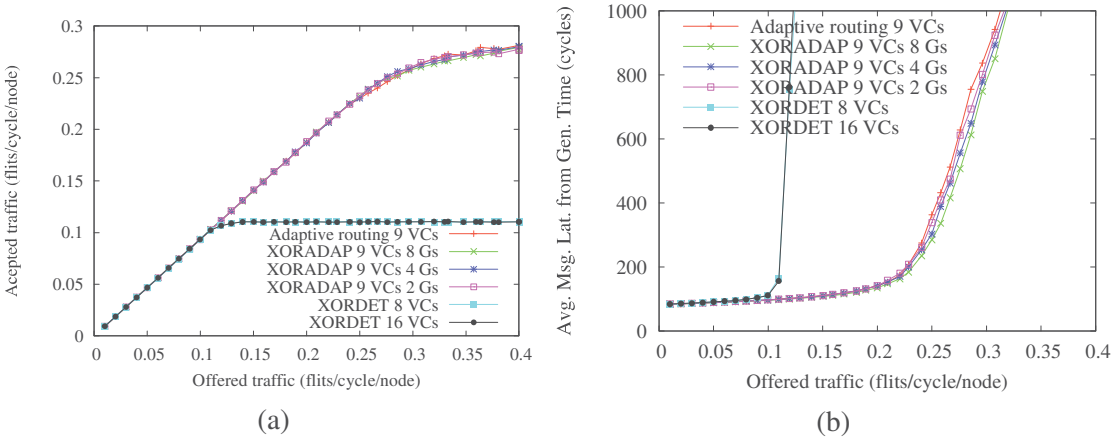


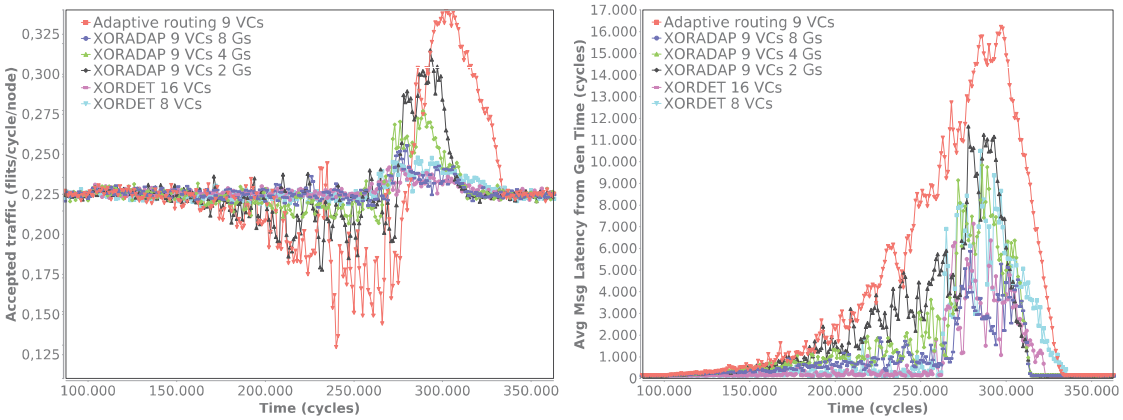
FIGURE 11.16: 256-node 2D-torus. Bit-reversal traffic. (a) Accepted traffic and (b) average packet latency vs. offered traffic.

As we mentioned in Section 11.3.2, there are some adversarial traffic patterns that significantly impact the performance of the network with deterministic routing algorithms. Nevertheless, XORADAP obtains good performance results not only with uniform random traffic pattern, but it is also able to obtain good results with those adversarial traffic patterns.

To illustrate this behavior, we have conducted some experiments with the matrix-transpose and bit-reversal traffic patterns. In Figure 11.15, we compare the behavior of XORDET, fully adaptive routing and the different configurations of XORADAP for the matrix transpose traffic pattern in a 256-node 2D torus. 9 VCs (8 adaptive channels and 1 escape channel) were used in fully adaptive and XORADAP routing algorithms, and 8 VCs and 16 VCs in XORDET. In XORADAP, the three aforementioned configurations were tested: two groups with 4 VCs each, 4 groups with 2VCs and 8 groups with only one per group.

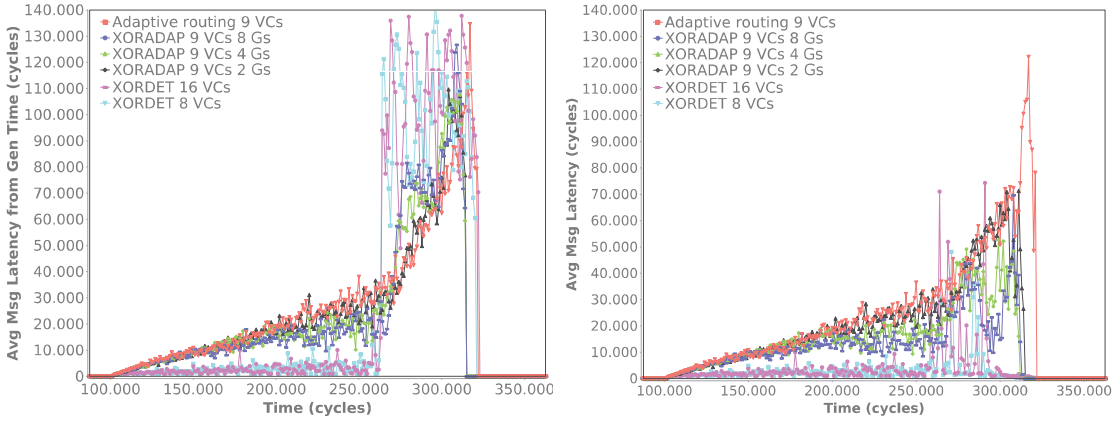
As expected, XORDET obtains a significantly lower throughput than any adaptive algorithm, in spite of using more VCs. In particular, fully adaptive routing more than doubles XORDET performance. This is the weakest point of deterministic routing. It is not able to efficiently cope with adversarial traffic patterns. The poor behavior of XORDET, and, in general, of any deterministic routing, is due to the unbalanced distribution of traffic for this pattern, which leads to overutilization of some links while other are unused [86]. Concerning the hybrid routing algorithm proposed in this paper, XORADAP, it obtains roughly the same results as fully adaptive routing, since it takes advantage of its flexibility making a better use of the links.

We can see a similar behavior for the bit-reversal traffic pattern in Figure 11.16. Again, there



(a) (b)

FIGURE 11.17: 256-node 2D-torus. Uniform random traffic pattern with hot-spot. Accepted traffic (a) and average packet latency (b) vs. simulation time.



(a) (b)

FIGURE 11.18: 256-node 2D-torus. Uniform random traffic pattern with hot-spot. Average packet latency (a) and network latency (b) for packets destined to the hot-spot vs. simulation time.

is a bottleneck when using XORDET deterministic routing. In particular, fully adaptive routing achieves almost 3X throughput than deterministic routing. Any of the configurations of XORADAP is able to reach the same performance obtained with fully adaptive routing.

Considering the results presented up to now, we can confirm that XORADAP achieves its first design goals. It is as good as fully adaptive routing for adversarial traffic patterns, thus improving XORDET and deterministic routing in general.

Next, we will analyze XORADAP behavior in the hot-spot scenario, where the HoL-blocking reduction is very important. Figure 11.17 shows the results for the same experiment performed in Section 11.4.1.1. Remember that the hot-spot traffic starts at clock cycle 100,000 and it is

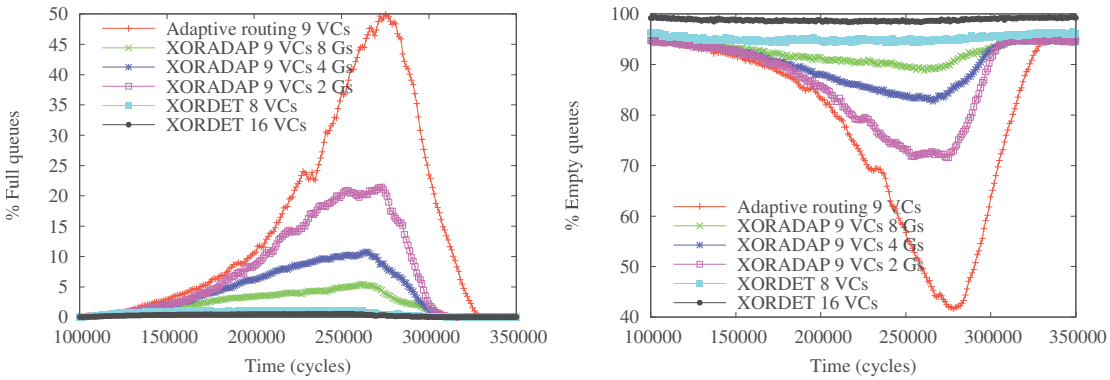


FIGURE 11.19: 256-node 2D-torus. Uniform random traffic pattern with hot-spot. Completely full (a) and empty (b) queues in the network vs. simulation time.

active until clock cycle 260,000. As expected, XORADAP helps to achieve a better behavior than fully adaptive routing. In particular, XORADAP configurations with more groups of VCs can better isolate the hot-spot traffic flows, obtaining a more stable value of accepted traffic (in fact very close to XORDET in the best case -XORADAP with 8 groups of VCs-) and a smaller average packet latency. On the other hand, if we use a XORADAP configuration with a few number of groups of VCs, two for example, we obtain a result more close to fully adaptive routing, but with smaller impact on the variability of delivered traffic and reducing packet latency with respect to fully adaptive routing.

Figure 11.18 confirms this behaviour. It shows the average message latency and network latency for packets destined to the hot-spot. As we can see in Figure 11.18-(a), latency strongly peaks at cycle 260,000 (i.e. when the hot-spot traffic becomes inactive) for XORDET and XORADAP. However, the network latency (i.e. without considering the time spent at source queues) plotted in Figure 11.18-(b) does not show the peak. As a consequence, this increase is due to packets that were waiting for long at the injection queues. As the routing algorithm restricted the resources (i.e. the VCs) they can use, packets destined to the hot-spot can not enter the network and must wait at the injection queues. Once the network is able to accept more traffic, these packets can be injected into the network, but the high time they waited at the source injection queues leads to a very high latency. Again, XORADAP with a high number of groups of VCs shows a behaviour close to XORDET, while XORADAP with a low number of groups of VCs is close to fully adaptive routing. Therefore, despite for uniform random and adversarial traffic patterns the number of XORADAP groups of VCs did not affect the performance results, for hot-spot traffic a configuration with a high number of groups of VCs seems the best design option.

Another interesting evidence of the behaviour of the evaluated routing algorithms is shown in Figure 11.19, which shows the percentage of completely full (Figure 11.19-(a)) or empty (Figure 11.19-(b)) VC queues in a 256-node 2D network. In order to perform a fair comparison, the shown percentages are relative to the number of VCs of the routing algorithm. As it can be seen, fully adaptive routing tends to fill up queues along the time the hot-spot is active, which is a symptom of spreading congestion. On the contrary, XORDET keeps most queues empty thanks to the traffic classification it performs. As expected, XORADAP shows a behavior that is half-way between fully adaptive and XORDET routing, depending on the number of groups of VCs.

The analysis shown before demonstrates that XORADAP also achieves its second design goal. It can be as good as a HoL-blocking reduction deterministic routing algorithm to classify and isolate traffic, outperforming fully adaptive routing under hot-spot traffic. To sum up, XORADAP routing algorithm combines the flexibility of adaptive routing with HoL-blocking reduction, being able to efficiently cope with varying networks loads, including uniform random traffic, adversarial or hot-spot traffic. Indeed, for a given number of VCs, several configurations are possible.

11.4.2 Switch Cost Analysis

This section estimates the cost of the different routing algorithms analyzed in this paper. To do so, we will take into account the number of required connections at the internal switch for each configuration.

Several internal switch configurations can be used with virtual channels [63]. We will assume a fully demultiplexed crossbar to implement the internal switch of routers. Although multiplexed crossbar configurations lead to less hardware, it requires more complex arbitration and also internal speedup.

However, although a full crossbar (i.e. with a number of ports equal to the product of the number of physical channels per the number of VCs) is able to cope with any of the analyzed routing algorithms, some connections are not actually required. By removing these connections, switch could be simpler. For instance, in all the routing algorithms, packets are never forwarded to the same port it arrived. Indeed, with DOR, packets may only be forwarded to dimensions higher than the one they entered the router. As a consequence, if the crossbar is implemented by, for

Routing	Switching Elements
Fully adaptive	$2v\sum_{i=1}^{n}(2nv-2n+2i-v+1)+2vn$
OODET	$2v\sum_{i=1}^{n}(2v(i-1)+v+1)+2vn$
IODET	$2v\sum_{i=1}^{n}(2v(i-1)+1+1)+2vn$
XORADAP	$2\sum_{i=1}^{n}(2nv-2n+2i-v+1)+2(v-1)\sum_{i=1}^{n}(\frac{2nv}{g}-\frac{2n}{g}+2i-\frac{v}{g}+\frac{1}{g})+2vn$
XORDET	$2v\sum_{i=1}^{n}(2(i-1)+1+1)+2vn$

TABLE 11.4: Number of switching elements for each routing algorithm.

instance, using a multiplexer at each output port, the ones corresponding to higher dimensions will have more inputs (and hence, more switching elements) than the ones located in the lower dimensions' ports. On the other hand, with fully adaptive routing, a packet entering through a port may be forwarded to any other output port. When several VCs are used and/or the number of network dimensions is high, the number of internal switch ports grows considerably. In addition, the injection and ejection ports must be also taken into account in any case.

To quantify switch complexity, we will measure the number of required switching elements per switch. We assume that an $i-$input multiplexer needs i switching elements. Table 11.4 shows the expressions of the number of switching elements for each analyzed routing algorithm taking into account its routing restrictions, n being the number of network dimensions, v the number of VCs per physical channel, and g the number of groups of VCs in the case of XORADAP routing algorithm. As an example, in OODET, output ports of last dimension can be requested by all input ports of lower dimensions (for each VC and for each direction) and by some ports of the same dimension (for each VC from the another direction) and, finally, by the injection port. So, we have $2v(n-1)+v+1$ possible requests per virtual channel for each direction of the last dimension and, in general, $2v(i-1)+v+1$ per virtual channel for each direction of the i dimension. Moreover, we have to add the necessary connections to the ejection port from each virtual channel per direction per dimension, $2vn$. The total number of required switching elements is then given by:

$$2v\sum_{i=1}^{n}(2v(i-1)+v+1)+2vn$$

Table 11.5 shows the number of required switching elements for the routing algorithms analyzed in this paper for different network configurations (number of network dimensions and number of VCs). Notice that we count an extra virtual channel in adaptive algorithms because XORADAP needs a number of virtual channels that needs to be power of to plus the escape channel. As

#Dim	#VC	Fully adaptive	OODET	IODET	XORADAP 2 Gs	XORADAP 4 Gs	XORADAP 8 Gs	XORDET
2	2(+1)	120	48	40	94	-	-	32
3	2(+1)	270	96	84	210	-	-	60
4	2(+1)	480	160	144	368	-	-	96
6	2(+1)	1080	336	312	816	-	-	192
2	4(+1)	320	160	112	224	176	-	64
3	4(+1)	750	336	264	510	390	-	120
4	4(+1)	1360	576	480	912	688	-	192
6	4(+1)	3120	1248	1104	2064	1536	-	384
2	8(+1)	1008	576	352	624	432	336	128
3	8(+1)	2430	1248	912	1470	990	750	240
4	8(+1)	4464	2176	1728	2672	1776	1328	384
6	8(+1)	10368	4800	4128	6144	4032	2976	768
2	16(+1)	3536	2176	1216	2000	1232	848	256
3	16(+1)	8870	4800	3360	4830	2910	1950	480
4	16(+1)	16048	8448	6528	8880	5296	3504	768
6	16(+1)	37536	18816	15936	20640	12192	7968	1536

TABLE 11.5: Comparison of the number of switching elements

expected, XORDET requires a crossbar with the fewest number of switching elements. This is due to two facts. First, messages traversing a given dimension cannot return to the previous dimensions, since DOR routing is used. Therefore, we could dispose those switching elements that connect input ports with output ports of lower dimensions. Second, the VC used by a given packet does not change along the path in the network, like in virtual networks. Hence, there is no need to have a crossbar connection (and the corresponding switching elements) to allow packets to perform VC transitioning in the same dimension and in the dimension changes.

Both IODET and OODET with DOR routing take also advantage of the first mentioned issue, that is, the connections to previous dimensions can be removed internally at the switch. But regarding the use of virtual networks, neither IODET nor OODET cannot use them. This is because IODET changes the VC when the dimension changes and OODET allows using any of the VCs in the dimension which is being currently crossed. However, for IODET, connections among input and output VCs of the same dimension are not required, but for OODET they are required. Regarding connections to the output VCs of higher dimensions, they are required by both routing algorithms. The worst case is the fully adaptive routing, as it requires a crossbar that enables almost all combinations of physical and VCs (the only exception are connections related to escape paths, which must be traversed in order). On the other hand, XORADAP also needs all combinations to different physical channels since dimensions are used in any order, but connecting only VCs of the same group. There is no need to have a crossbar connection to allow packets to perform VC transitions among different groups of VCs. The more the number

of groups of VCs, the less the number of VCs per group and the simpler the crossbar. The connections of escape paths are also required. XORADAP requires a switch complexity that is always lower than fully adaptive routing and, depending on the number of groups of VCs, it can be strongly reduced.

In all cases, as the number of VCs is increased, the number of required switching elements further increases, specially for fully adaptive routing with a high number of dimensions. The deterministic routing design, XORDET, obtains the lowest number of required switching elements. Therefore, XORDET is a good option because of its low cost. But if what is required is an algorithm which provides flexibility in addition to HoL-blocking reduction, XORADAP is a very good choice because it strongly reduces these requirements compared to fully adaptive routing algorithm, specially for the configurations with a high number of groups of VCs.

11.5 Conclusions

This paper presents a XOR-based routing mechanism which reduces the HoL-Blocking effect taking advantage of the available virtual channels. Packets are classified in the VCs by performing a XOR bitwise function of their destination node identifiers. This mechanism is used to design both deterministic and adaptive routing algorithms (XORDET and XORADAP, respectively). XORDET deterministic routing obtains performance results close to traditional fully adaptive routing under uniform random traffic pattern. Most important, with hot-spot traffic situations, it is able to isolate the packets destined to the hot-spot node, reducing the interference to the rest of traffic. Furthermore, XORDET keeps the good properties of deterministic routing such as in-order delivery of packets, and simpler switch implementation. However, there are some adversarial traffic patterns which reduce accepted traffic rate with deterministic routing, and XORDET is not an exception. XORADAP routing algorithm tries to combine the best features of both adaptive routing (flexibility under some adversarial traffic patterns) and deterministic routing algorithms specially designed to reduce the HoL-blocking effect (destination isolation). The evaluation results show that (i) XORADAP obtains similar performance results to either deterministic or fully adaptive routing with uniform random traffic pattern; (ii) it achieves a similar behavior to traditional fully adaptive routing for those traffic patterns where routing flexibility is required to avoid bottlenecks balancing link utilization; and (iii) it is also able to cope with hot-spot traffic situations, being able to isolate the packets destined to the hot-spot, reducing the interference to the rest of traffic, like HoL-blocking reduction deterministic

routing does. Indeed, several configurations of the VCs are possible in XORADAP to enhance either its routing flexibility (i.e. adaptive routing behavior flavour) or its destination isolation (i.e. HoL-blocking reduction deterministic routing behavior flavour).

Chapter 12

General Discussion of Results

In this thesis, we have made several proposals to increase the efficiency of interconnection networks, specially important for current large systems. First, the KNS topology family is proposed. This family is based on a hybrid topology where we combine the benefits of both direct and indirect topologies. We also propose some routing algorithms for these new topologies, including a fault-tolerant routing algorithm to deal with the increasing problem of fault-tolerance in current massively parallel systems.

Additionally, this thesis also deals with designing routing algorithms specially targeted to avoid the HoL-blocking effect. These routing algorithms are intended for direct topologies.

In this chapter we will do a brief description of these proposals and present some discussion about the obtained results. A complete description of each of them can be found in the corresponding chapters.

12.1 k_h–ary n_h–direct s_h–indirect (KNS) topology

This thesis proposes a new family of topologies for interconnection networks that allows to efficiently connect an extremely high number of processing nodes, given the huge size of current and near future supercomputers [1]. We propose an hybrid topology based on combining an n–dimensional direct network with small indirect topologies. The aim is to combine the advantages of direct and indirect topologies to obtain a family of topologies that is able to connect a high number of processing nodes, providing low latency, high throughput and a high level of fault–tolerance at a lower hardware cost than indirect networks. In the proposed topology, the nodes

of each dimension are connected through an indirect topology that allows to have a large number of nodes per dimension without negatively affecting performance.

The new family of topologies will use two different kind of switches in the network (although this is not entirely true, as we will see later). First, low–degree switches are used to connect processing nodes to each dimension and move packets across dimensions. We will refer to these switches as *routers*. They have, at least, one processing node attached to them and as many ports as dimensions. Nowadays, it is common that processing nodes are composed of a large number of cores, and the core count per processing node trend is to increase even further. So, these processing nodes need network interfaces with high bandwidth to avoid bottlenecks in the end processing node connection to the network. In fact, there are already some commercial solutions which use network interfaces cards with dual ports [47]. Considering the core count increase trend it is expected that the bandwidth requirements of end processing nodes will be even higher. Another key point of these network interfaces with several ports is that they provide fault-tolerance. Using one port network interfaces causes having a single point of failure that will disconnect a high number of cores. Both, bandwidth and fault tolerance requirements will provoke increasing the number of ports in the near-future network interfaces.

In this thesis we take advantage of these network interface cards with several ports to implement the routers used in the new family of topologies.

Additionally to these routers, the proposed topology also uses other *switches* to implement the indirect networks that interconnect the routers of each dimension.

12.1.1 Description of the family

The newly proposed topology family arranges processing nodes and their associated routers (node) in n dimensions like a direct topology. But, contrary to mesh and torus topologies, routers of a given dimension are not only connected with their adjacent routers in that dimension. Instead, all the routers of a given dimension are connected by means of a small indirect network. This indirect network could be even a single switch, considering the number of ports of current commercial high-radix switches. However, if the number of routers per dimension exceeds the number of ports of the available switches, the indirect network will be arranged as a small MIN. We will refer to this MIN as the *indirect subnet*. This will provide a low latency communication among routers in the same dimension with a small hardware extra cost

compared to direct topologies. In this way, the number of routers per dimension stops being a bottleneck from the point of view of the performance, as the time to communicate two routers of the same dimension is constant or grows logarithmically with the number of routers per dimension. Notice also that each router only requires a bidirectional link per dimension to connect to the switch of each dimension. On the contrary, a mesh or torus require two bidirectional links per dimension, one per neighbor node.

The proposed family of topologies is defined by three parameters. Two of them are inherited from direct networks: the number of dimensions n_h[1] and the number of routers per dimension k_h. The number of processing nodes it can interconnect is given by $N = k_h^{n_h}$. In addition to these two parameters, there is an additional parameter, the number of stages of the indirect subnet, referred to as s_h. This number depends on the number of routers per dimension k_h and on the number of ports of the switches used to implement the indirect subnet (which will be referred to as d_h). The ratio between k_h and d_h defines the way to interconnect the k_h routers of each dimension. If $k_h \leq d_h$, a simple switch would be able to interconnect all the routers of the dimension, and s_h will be equal to 1. On the contrary, if $k_h > d_h$, a MIN would be required to interconnect the routers of each dimension, and the number of required stages will be given by $log_{\frac{d_h}{2}} k_h$ (remember that in an indirect network built with d_h-port switches, the network radix is equal to $\frac{d_h}{2}$). We will refer to the new family of topologies proposed in this thesis as k_h–ary n_h–direct s_h–indirect (KNS), where k_h is the number of routers per dimension, n_h is the number of dimensions and s_h is the number of stages of the indirect subnets.

In this thesis, we have considered two different MINs to connect the routers of a given dimension. The first one is a BMIN, the fat–tree. The second one is RUFT [5], a UMIN derived from a fat–tree using a load–balanced deterministic routing algorithm [22], which requires less hardware resources.

Figure 12.1 shows an example of the new topology with 2 dimensions ($n_h = 2$), and 4 routers per dimension ($k_h = 4$), with a total number of 16 routers. In this case, the routers of the same dimension are connected by a single 4-port crossbar. However, for a higher number of routers per dimension, say $k_h = 8$ and the same switch size, a MIN should be used. In the case of using fat–tree subnets, for interconnecting the 8 routers of each dimension, it will require 3 stages and 12 bidirectional 4-port switches, implementing in this way a 8–ary 2–direct 3–indirect. In

[1]The subscript "h" stands for hybrid.

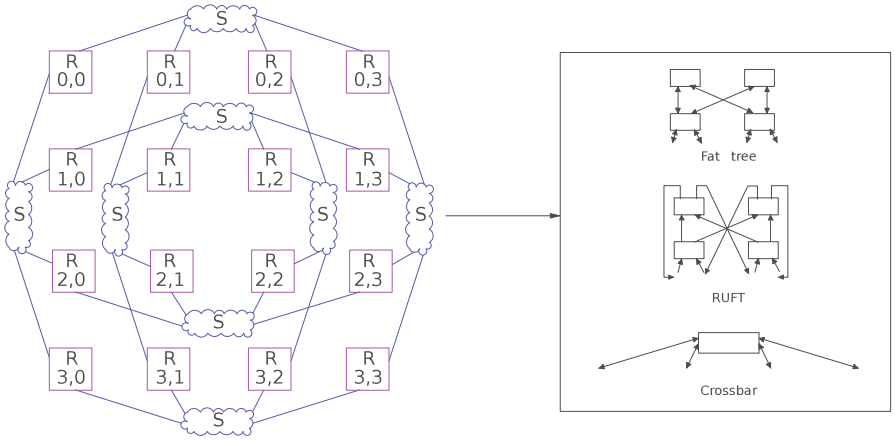


FIGURE 12.1: An example of the KNS topology with $n_h = 2$, $k_h = 4$ and $d_h = 4$.

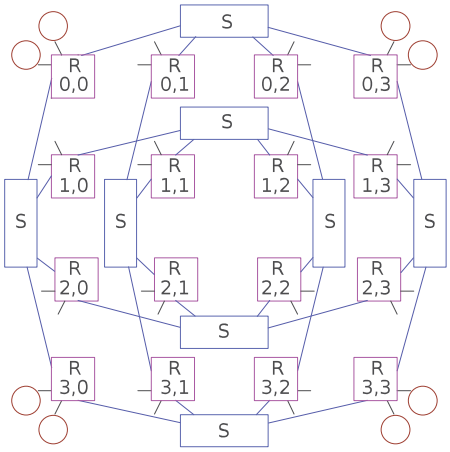


FIGURE 12.2: An example of the KNS topology with $n_h = 2$, $k_h = 4$, $s_h = 1$ and $p_h = 2$.

the case of using RUFT as indirect subnets, it is also a 8–ary 2–direct 3–indirect but built with 4-port unidirectional switches.

As it can be seen in Figure 12.2, the routers are connected to all dimensions through a different link (one per dimension). Notice that it is also possible to attach several processing nodes to the same router. Having more than one processing node attached to each router is known as concentration in the literature. In fact, the topology shown in the figure already has two processing nodes attached to each router. Only the corner processing nodes are shown in the figure for the sake of clarity. This introduces a new parameter on the topology family, p_h, the number of processing elements that are connected to each router. In this case, the number of processing nodes is given by $N = p_h k_h^{n_h}$.

In Figure 12.1, two different components with switching capabilities can be distinguished. The

TABLE 12.1: Parameters of the different analyzed topologies.

Topology	Parameters	
Mesh or Torus	n_d	# dimensions
	k_d	# of nodes per dimension
Fat–tree	n_i	# of stages
	k_i	switch arity
k_h–ary n_h–direct s_h–indirect (KNS)	n_h	# of dimensions
	k_h	# of routers per dimension
	s_h	# of stages of the indirect subnet
	d_h	switch degree
	p_h	# processing nodes per router

switches[2] (in blue and labelled with a "S" in Figure 12.1) are only connected to other switching components, whereas the routers (in pink and labelled with an "R" in Figure 12.1) are used to connect the processing nodes to the network through several dimensions. These routers are connected on one side with processing nodes and on the other side with switches, and their degree is $n_h + p_h$, that is, the sum of the number of dimensions and the number of concentrated processing nodes. As it can be seen in Figure 12.1, if a crossbar is used as indirect subnet, switches allow packets to change to any position in a given dimension by traversing only two links, whereas routers allow to travel between dimensions. Thus as, at most, only two hops are performed per dimension, the diameter of the new topology is $2n_h$, which is a very low value. If a MIN is used instead of a crossbar as indirect subnet, the diameter of the network will be the diameter of the used MIN (for a fat–tree, i.e. a BMIN, $2\times$ the number of stages, which is expected to be a small number due to using high-radix switches) multiplied by n_h, that is $2s_h * n_h$.

Table 12.1 summarizes the parameters that define the hybrid family of topologies. Notice that only one of d_h or s_h parameter is necessary, as the another one can be derived from the other parameters. In addition, the parameters that define traditional direct and indirect topologies are also shown.

The proposed hybrid topologies can take advantage of the high number of ports available in current switches. For example, edge switches of up to 36 ports are commercially available [68], while chassis switches offer up to 648 ports [68]. With such switches, in most cases, a small MIN with only 2 or 3 stages or even a single switch will be enough to connect the routers in

[2]It may be a single switch or a set of switches forming a MIN.

each dimension. As the MIN is very small, it will introduce low latency and it can be easily implemented with low wiring complexity.

On the other hand, as already stated, this topology can take advantage of new network interfaces (like the new commercial HCA cards [47]), that have several ports to connect the processing nodes to the network. By using these network interfaces, the proposed topologies can be implemented by integrating the router into the processing node as part of the network interface. These network interfaces will have switching capabilities, so processing nodes equipped with these new network interfaces will be able, apart from injecting messages in the network, to route packets that are not destined to them to other processing nodes without ejecting messages from the network.

The resulting topologies could seem similar to BCube [20] or Hypercrossbar network [21], but the great difference between our proposal and BCube is that the processing nodes do not eject messages from the network; the new topology family forwards the messages in the network interface which highly reduces latency. However, these cases could be considered as particular configurations of the KNS family of topologies.

The topologies proposed in this thesis have several main advantages. First, they allow to highly reduce the diameter compared to direct topologies. This will lead to network performance improvements, decreasing latency and increasing network throughput. Additionally, the number of required switches and links is reduced compared to an indirect topology that connects the same number of processing nodes. Therefore, it is expected that the proposed family of topologies reduces the cost of the network. Finally, it also provides a good fault–tolerance level.

12.1.2 Routing Algorithms for the KNS Family of Topologies

In this section, we describe the routing algorithms proposed for the new family of topologies. We will first describe the ones proposed for k_h–ary n_h–direct 1–indirect topologies (i.e., a crossbar is used as indirect subnet). Then, we describe the ones proposed for the general case, that is, for KNS topologies, using a fat–tree or a RUFT as indirect subnets.

First, we explain how routers and switches are labeled in the KNS topology. Each router is labeled as in meshes and tori, with a set of components or coordinates (as many as network dimensions) $\langle r_{n_h-1}, r_{n_h-2}, \ldots, r_1, r_0 \rangle$. Each coordinate represents the position of each router in each of the dimensions. On the other hand, the switches are labeled by a 2-tuple *[d,p]*, where

d is the dimension the switch is located at, and *p* is the position of that switch in that dimension. Notice that routers do not belong to any dimension, since they are connected to all of them, and packets change dimensions through them. On the contrary, switches do not allow changing the dimension packets are traversing, they just allow packets to move through that dimension.

12.1.2.1 Routing in k_h–ary n_h–direct 1–indirects

Although both deterministic and adaptive routing algorithms could be used, taking into account that adaptive routing may introduce out-of-order delivery of packets and that leads to a more complex implementation, we will focus only on deterministic routing.

The deterministic routing algorithm for k_h–ary n_h–direct 1–indirect topologies, which will be referred to as Hybrid–DOR, is a variation of the dimension ordered deterministic routing algorithm (DOR) for meshes, adapted to the k_h–ary n_h–direct 1–indirect topology. In DOR, packets are routed through the different dimensions following an established order until the destination processing node is reached. At each dimension of the mesh, packets traverse several routers until the movement in that dimension is exhausted. On the other hand, as each mesh router has two links per dimension, packets must be forwarded in each dimension through the direction that guarantees the minimal path.

In Hybrid–DOR, network dimensions are also crossed in an established order to guarantee deadlock freedom, as in DOR. However, there is a unique link per dimension that connects the current router to a switch that allows directly reaching any of the processing nodes in that dimension. So, packets do not perform several hops at each dimension. Instead, in Hybrid–DOR, routers directly forward packets through the unique link of the dimension they have to traverse, and this link is connected to the corresponding crossbar that moves the packet to the destination component in that dimension. Notice that, contrary to meshes and tori, in k_h–ary n_h–direct 1–indirect topologies, it is not required to choose the direction at each dimension, as there is only one link per dimension. The routing in the switches is very straightforward since, they just must forward packets through the link indicated by the destination component in the current dimension, requiring just one hop to reach next router.

Next, we show the Hybrid–DOR pseudo–code for the routers and the crossbars of the network. The number of dimensions of the topology is n_h and the destination and current router coordinates are given by $\langle x_{n_h-1}, \ldots, x_{d+1}, x_d, x_{d-1}, \ldots, x_1, x_0 \rangle$, and $\langle r_{n_h-1}, \ldots, r_{d+1}, r_d, r_{d-1},$

$\ldots, r_1, r_0\rangle$, respectively. In the case of crossbars, the current switch is given by *[d,p]* (the p^{th} switch of the d dimension). The chosen link to send the packet is returned in $link$.

Routers:

$i = 0;$
$Done = False$
while $(i < n_h) \wedge (!Done)$ **do**
  if $x_i! = r_i$ **then**
    $Done = True$
    $link = i$
  end if
  $i = i + 1$
end while

Crossbars:

$link = x_d$

As can be seen, routers select the next dimension to forward the packet, which it is also the link of the router to be used, since there is just one link per dimension, and crossbars merely select the link given by the destination coordinate of the corresponding dimension to reach the destination component in that dimension.

12.1.2.2 Routing in k_h–ary n_h–direct s_h–indirects

In k_h–ary n_h–direct s_h–indirect topologies, the crossbars are replaced by small MINs. As stated above, the MINs considered in this thesis are fat–trees or RUFTs. In these topologies, all the switches of a given fat–tree or RUFT are always in the same dimension and in the same position relative to the routers. In order to identify the switches inside a given fat–tree or RUFT, we extend the classical switch coordinates from MINs by including the coordinates of the MIN in the direct topology. In this way, the switch coordinates in k_h–ary n_h–direct s_h–indirect topologies will be given by a 4-tuple *[d,p,e,o]*, where d is the dimension the MIN belongs to, p is the position of the MIN in that dimension, e is the stage of the switch inside the MIN, and o is the order of that switch in that stage. Remember that d and p are the coordinates of the equivalent crossbar in k_h–ary n_h–direct 1–indirect topologies.

Since the routers are the same regardless of the indirect topology used, its routing algorithm is the same as the one shown for k_h–ary n_h–direct 1–indirect topologies. However, switch routing algorithm depends on the particular indirect network used.

First, we focus on the k_h–ary n_h–direct s_h–indirect topology that uses fat–trees in the indirect subnets. Despite the fact that a fat–tree has several paths for each source–destination pair (i.e., it allows adaptive routing), we propose to use the deterministic routing algorithm presented in [22] since it is simpler and is able to outperform adaptive routing. We will summarize that routing algorithm here. Routing is composed of two subpaths. First, packets are sent upwards to the common ancestor switch of the source and destination processing nodes and, then, they are sent downwards to its destination. Traffic is balanced by carefully selecting the links to be used according to the destination processing node. In particular, the link to be used in both subpaths is given by the destination coordinate corresponding to the stage where the switch is located at. For instance, if a switch located at stage e_1 routes a packet whose destination (in the fat–tree) is $\langle t_{n_i-1}, \ldots, t_{e_1+1}, t_{e_1}, t_{e_1-1}, \ldots, t_1, t_0 \rangle$, then the packet is sent through the link $k_i + t_{e_1}$ in the upwards phase and through link t_{e_1} in the downwards phase. Remember that k_i is the arity of the switches of the fat–tree topology. Please see [22] for more details.

In the fat–trees subnets of the k_h–ary n_h–direct s_h–indirect topologies, the routing algorithm is the same, but only the part of the destination identifier corresponding to the dimension the fat-tree belongs to (i.e., x_d in our notation) is used. In this way, the packet is delivered to the same router that would be reached through the corresponding crossbar in a k_h–ary n_h–direct 1–indirect topology.

This routing algorithm is shown below. Assume that destination coordinates are $\langle x_{n_h-1}, \ldots, x_{d+1}, x_d, x_{d-1}, \ldots, x_1, x_0 \rangle$, the dimension where the fat–tree is located at is d, $GetFTIdentifier$ returns from x_d the fat–tree coordinates to route locally in the fat–tree, $UpwardsPhase$ returns *true* if the packet is in its upwards subpath, or *false* otherwise, and that the stage in the fat-tree of the switch that is routing the packet is given by e:

Switches:

$t = GetFTIdentifier(x_d)$
if $UpwardsPhase()$ **then**
  $link = k_i + t_e$
else
  $link = t_e$
end if

Let us consider the case where RUFT is used in the indirect subnets of the KNS topology. In RUFT, there is a unique path between each source–destination pair and packets have to cross

all the stages, reaching the last stage, which is directly connected back to the processing nodes. The link to be used by a packet at a particular switch is given by the destination component corresponding to the stage the switch is located at. Please see [5] for details. In this case, the pseudo–code for the switch routing algorithm is the following:

Switches:

$t = GetFTIdentifier(x_d)$

$link = t_e$

If Hybrid-DOR is used in the routers jointly with the aforementioned algorithms for the switches in the indirect networks, the resulting routing algorithm for the new topology is deterministic and deadlock-free, since dimensions are crossed in order in the direct topology and the routing algorithm used in the indirect networks has not any loop in its channel dependency graph [56].

12.2 Fault tolerance for k_h–ary n_h–direct s_h–indirect

Next, we deal with proposing a fault-tolerant routing algorithm for the new topologies. For this proposal, we will assume a KNS topology using crossbars as indirect subnetworks. Concerning routing, Hybrid-DOR [67] is used. As above mentioned, Hybrid-DOR is a deterministic routing algorithm that crosses network dimensions in increasing order.

Let us analyze how to deal with network faults. We will only consider link faults, since a switch fault can be easily modeled as a switch with failures in all of its links. However, notice that in this case, link failures of a switch would be correlated. Moreover, as network links are bidirectional, we assume that if there is a link fault, then it fails in both directions. In this thesis, we do not focus on how the failure information propagates to network nodes. In this way, a static fault model is assumed. This means that when a fault is discovered all the processes are stopped, the network is emptied, and a management application is run in order to deal with the fault. Checkpointing techniques must also be used so that applications can be brought back to a consistent state prior to the fault occurred. Detection of faults, checkpointing, and distribution of routing info is assumed to be performed as part of the static fault model, and are therefore not further discussed in this thesis.

For each source-destination pair without failures in their path, packets are routed using Hybrid-DOR following minimal paths. But, if there is any fault in the path of a given source-destination

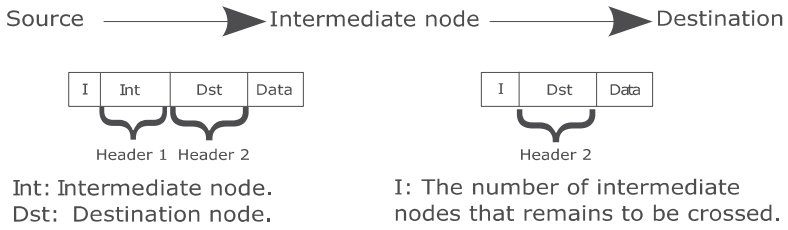


FIGURE 12.3: Header for packets using the intermediate node methodology.

pair, the methodology routes packets through intermediate nodes, like in [46]. The use of intermediate nodes was proposed in [45] for other purposes, such as traffic balancing. In our case, the idea is to avoid the faults by deviating the packet to an intermediate node. Therefore, a suitable intermediate node for each source-destination pair with faults in their path needs to be selected. The routing algorithm avoids faults by first sending the packet to an intermediate node and then, from this intermediate node, to the destination node. Several intermediate nodes could be also used for each pair of nodes, where packets are forwarded through these nodes until the destination node is reached. Notice that the Hybrid-DOR algorithm is used in all sub-paths. Using more than one intermediate node allows the mechanism to tolerate more faults by having more control over the global path followed by the packet. Notice also that the packets are not ejected from the network when they reach intermediate nodes. In Sections 12.2.1 and 12.2.2 we will describe in detail how intermediate nodes are selected for the case of requiring only one, or several of them, respectively.

Regarding the structure of packets, several fields should be added to the packet header to support routing through intermediate nodes. Figure 12.3 shows the packet header. First, the number of intermediate nodes that the packet has to cross is stored in a new field (I in the figure). In addition to the destination node, the addresses of intermediate nodes are also stored in the packet header. Every time an intermediate node is reached, its address is removed from the packet header and the I field is decreased. Other implementations are possible, such as storing in the packet header a pointer to the field that should be considered for routing. As soon as the packet reaches an intermediate node, this field will point to the next intermediate one or, finally, to the destination node.

For each source–destination pair, the mechanism checks whether the deterministic Hybrid-DOR path is fault-free or not. If not, routing through intermediate nodes is required. A list of intermediate nodes should be computed for paths with faults. This list will be stored in a table at every source node. There is a table entry for each destination node that requires routing through

intermediate nodes. This table can be implemented as linear (i.e., with as many entries as the network size) or as random (i.e., as content-addressable memories) tables. The size of the latter table depends on the number of faults the network has to tolerate. The higher this number, the higher the number of affected source–destination pairs. In practice, though, it is expected that it should be enough to tolerate a relatively low number of faults, as if a network suffers a high number of faults, the problem should be solved in another way.

However, the size of these tables also depends on how we codify the info stored in them. For example, considering a 2-D network, if the fault is located at the first dimension link of a node (i.e. the one that connects to the nodes of the same row), each source node in the same row (except the one connected to the faulty link) requires routing through intermediate nodes to reach every other node located in the same column of the fault. However, in the same example, the source node that is connected with the faulty link needs an intermediate node for each destination node in every other columns. This means that $k*(k-1)$ destination nodes need an intermediate node to reach them from this source node. Generally speaking, for more network dimensions, if a fault occurs at a link of the i dimension, the source node requires using intermediate nodes to reach $(k^{(n-i-1)}) * (k-1)$ destinations. Moreover, if there are more faulty links in this source node, each fault will involve a number of destinations according to this equation. Therefore, a table design where there is an entry for each destination that needs an intermediate node could require a big size in large networks. In such cases, an alternative design based on the use of masks, similar to IP routing tables, could be used. The table would have two fields (id to compare and mask to select the bits) and an option flag. The option flag indicates whether the corresponding entry is considered when the comparison is satisfied or not. Of course, there is also a field to store the possible intermediate nodes. In general, each entry can be used for a set of destinations, thus reducing the size of the table. However, some destination nodes may need specific intermediate nodes. Additional entries for them will be included in the table, which will lead to several hits for the same destination node. The solution is to insert the entries in the table following a given priority order and then selecting the entry with higher priority.

As in [46], we will refer to the source node as S and the destination node as D. For each intermediate node, we will use the notation I_x, where x represents the index of the intermediate node (I_1 for the first one, I_2 for the second one and so on). To ensure deadlock freedom, the routing algorithm needs at least as many additional virtual channels as intermediate nodes for fault-tolerant routing. For example, if we use up to two intermediate nodes, we need at least three virtual channels (i.e., the original plus two additional ones). When a packet reaches an

intermediate node, the packet is re-injected into the network using a new virtual channel to avoid deadlocks. For instance, assume that two intermediate nodes are used. One virtual channel (v_1) is used from S to I_1, another one (v_2) from I_1 to I_2, and the last one (v_3) from I_2 to D. In this way, deadlocks are avoided because the network is split into three virtual networks, deadlock-free routing algorithm is used within each virtual network, and each virtual network transition is performed following a strict order ($v_1 \rightarrow v_2 \rightarrow v_3$ in the example). Adaptive routing could also be used. In this case, several virtual channels can be used for adaptive routing provided that there is an escape channel to break cyclic dependencies for each subpath [56]. Routing in the escape channels uses Hybrid-DOR. Adaptive channels can be used in any of the sub-paths. However, in such a case, a different escape channel is required in each sub-path to ensure deadlock freedom. In this section, though, we only focus on deterministic routing.

There are some combinations of failures that physically disconnect one or more nodes of the network. As the proposed methodology does not add new resources to the network, these sets of faults can not be supported. The aim of the proposed methodology is to provide a path for every source–destination pair, provided that they are physically connected by the interconnection network.

Next, we will present how the intermediate nodes are selected. First, we will focus on using only one intermediate node. After that, we show how to extend the methodology to use multiple intermediate nodes.

12.2.1 One Intermediate Node

In this case, we will use only one intermediate node when one or more faults affect the minimal path provided by Hybrid-DOR between a pair of nodes. This intermediate node, I_1, has to satisfy two rules:

R1. I_1 is reachable from S.

R2. D is reachable from I_1.

We say that a node Y is reachable from X if there is a minimal path provided by Hybrid-DOR between this pair of nodes, and there is no fault in it.

For direct topologies like tori or meshes, the choice of this intermediate node is very important, because the number of hops can greatly increase, depending on the selected intermediate node.

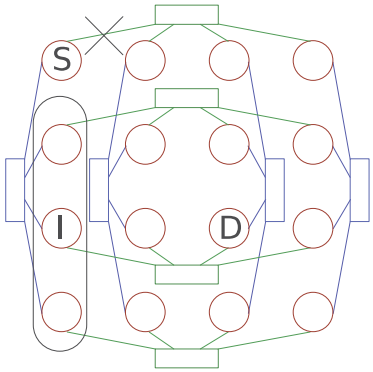


FIGURE 12.4: Example of a faulty link in a KNS topology with $n = 2$ and $k = 4$.

However, in the KNS topology, the number of hops only depends on the number of dimensions that the packet must cross. The proposed methodology prioritizes the use of those intermediate nodes which allow the packet to reach the destination node without additional hops compared to the original minimal path between the source and the destination nodes.

For instance, Figure 12.4 shows a 4-ary 2-direct 1-indirect network with a fault at the source node S in the x dimension link. So, it cannot reach the destination node D using Hybrid-DOR. In this case, the possible intermediate nodes are all the nodes of the same column (surrounded by the dashed line in the figure). The best choice, however, is the node that is located in the same row as the destination node, because it allows to reach the destination node by using a minimal path.

We say that a fault-tolerant routing algorithm is able to tolerate f failures if it can provide a valid path between every source-destination pair for any combination of up to f failures.

12.2.2 Multiple Intermediate Nodes

There are cases where only one intermediate node is not enough to handle the network fault combination. In these cases, the routing algorithm can use more than one intermediate node to have more chances of finding a set of fault-free deterministic Hybrid-DOR paths between the source and destination nodes. Assume that a number of x intermediate nodes $I_1, I_2, \ldots I_x$ are required. Intermediate nodes are selected according to the following rules:

R1. I_1 is reachable from S.

R2. I_{i+1} is reachable from I_i, for $0 < i < x, x > 1$.

R3. D is reachable from I_x.

Therefore, we can guarantee that the packet is able to reach its destination node following the path S-I_1-...-I_i-I_{i+1}-...-I_x-D.

In order to avoid non-minimal routing, the methodology tries to use a set of intermediate nodes that does not increase the number of hops beyond a minimal path. In particular, if there are non-minimal available paths using i intermediate nodes and also a minimal path using j intermediate nodes, where $i < j$, the latter will be finally selected.

12.2.3 Extension To Any Indirect Subnetwork

In the proposal of this fault-tolerant routing algorithm, we have focused on KNS topologies that use crossbars as indirect subnetworks. However, we can extend the methodology to KNS topologies that use other indirect subnetworks like fat-trees or RUFT. To do this, a specifically designed methodology should also be used to tolerate faults on each indirect subnetwork. Therefore, intermediate nodes are used globally, while a specific methodology to tolerate faults will be used locally on each subnetwork.

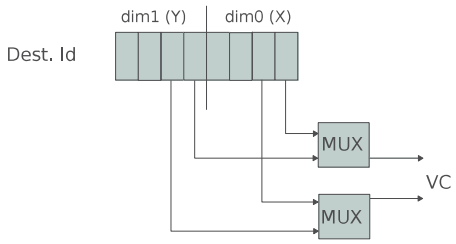
While the subnetworks can avoid faults, the direct routers will work normally. However, if a node becomes unreachable due to a fault located at a given subnetwork, it will be modeled like a link fault at this node, in the link of the corresponding dimension of this faulty subnetwork.

12.3 Reducing the HoL-blocking effect in Direct Topologies

The HoL-blocking effect has a great impact in network performance. Direct topologies were initially chosen due to their popularity and because they are the basis of the KNS topology. To achieve this goal, we have proposed some routing algorithms that takes advantage of virtual channels of switches in order to confine packets depending on their destinations. The algorithm used to classify the different packets in virtual channels is very important and affects interconnection network performance. An important point to consider is the number of destinations that are assigned to a given virtual channel. If some virtual channels are overloaded and other ones are less used, the most saturated channels probably kill network performance. In this thesis we

Dim	VC#0	VC#1	VC#2	VC#3
X	8	16	16	16
Y	1	2	2	2

TABLE 12.2: How IODET assigns destinations to node 0 VCs in a 8×8 mesh. #VCs is 4



(a) IODET

FIGURE 12.5: Implementation of VC selection in IODET for a 256-node 2D network and 4 VCs.

have proposed three different routing algorithms with the objective of achieving a good performance when there is not network congestion and isolating different destinations when there is network congestion.

12.3.1 IODET: In Order DETerministic Routing

In-Order DETerministic routing (IODET) [64] selects the VC by considering not the whole destination identifier but the component of the packet destination corresponding to the dimension in which the packet is being routed. The VC to be used by a packet is obtained by performing the modulo operation of the dimension coordinates of the destination. That is, given a packet destined to node $\{p_{n-1}, ..., p_1, p_0\}$, when routed in dimension d it will use the VC given by $p_d \bmod \#VCs$. This mechanism does a better job classifying packets than DBBM as can be seen in Table 12.2, which shows the number of destinations assigned to different VCs for node 0 for an 8x8 mesh with 4 VCs. As it can be seen, all the VCs in both dimensions are used when applying IODET to distribute destination among VCs.

Considering the implementation complexity of the VC selection, IODET is also very simple, as displayed in Figure 12.5 which shows an example for 4 VCs. As it can be seen, the least significant bits of the component for each dimension is used to select the VC. However, as the assignment of destinations to VCs depends on the dimension the packet is traversing, the VC is changed when there is a dimension change and, therefore, in those nodes the new VC to use

must be computed. As a consequence, the node internal switch must allow the change in the VC assignment and the switch implementation is not as easy as the DBBM one.

12.3.2 XOR-based HoL-blocking Reduction Routing

In this section, we present a mechanism to assign destinations to VCs based on the use of XOR function. We apply this destination distribution to two different routing algorithms: first a deterministic routing mechanism (XORDET) [66] is designed, and then an adaptive version (XORADAP) [77] is proposed. The idea of this second algorithm is to combine the good properties of both, deterministic and adaptive routing to adapt to any traffic pattern to obtain optimal performance results.

12.3.2.1 XORDET: XOR DETerministic Routing

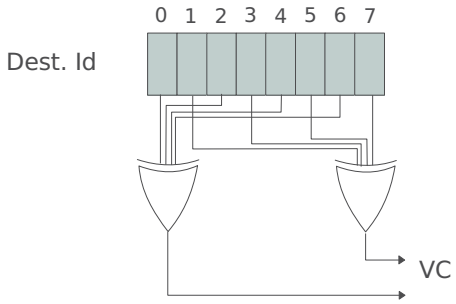
XORDET is a deterministic routing algorithm that reduces the HoL-blocking effect and performs a balanced distribution of destinations among VCs. For doing that, opposite to the previously presented deterministic algorithms that use a subset of the node destination bits, XORDET distributes destinations among VCs by performing a bitwise XOR operation to all the bits of the destination node, as follows. Assume that there are v VCs available. Then, $l = \log_2 v$ bits are required to denote a virtual channel. If destination identifiers are n bits long, then, for each destination, the VC to use is obtained by performing l XOR operations in parallel. In each XOR operation $\frac{n}{l}$ bits of the destination are XORed, taking them in an interleaved fashion. In particular, given a packet destined to node $\{p_{n-1}, ..., p_1, p_0\}$, it will use the VC given by the bits $\{VC_{l-1}, ..., VC_1, VC_0\}$, computed as follows:

$$VC_0 = p_0 \oplus p_{0+l} \oplus p_{0+2l} \ldots \oplus p_{0+(\frac{n}{l}-1)l}$$

$$VC_1 = p_1 \oplus p_{1+l} \oplus p_{1+2l} \ldots \oplus p_{1+(\frac{n}{l}-1)l}$$

$$VC_{l-1} = p_{l-1} \oplus p_{l-1+l} \oplus p_{l-1+2l} \ldots \oplus p_{l-1+(\frac{n}{l}-1)l}$$

Figure 12.6 shows how the VC selection will be implemented in a network with 4 VCs and 8-bit node identifiers. Each VC bit is obtained by XORing 4 bits of the node destination identifier in an interleaved fashion. Notice that XORDET implementation of VC selection is very simple, as only some XOR gates are required per source node. In particular, for v VCs, $l = \log v$ XOR gates are required. Each one of them will have $\frac{n}{l}$ inputs, n being the number of bits



(a) XORDET

FIGURE 12.6: Implementation of VC selection in XORDET for a 256-node 2D network and 4 VCs.

FIGURE 12.7: How XORDET assigns destinations to VCs in a 8×8 mesh with 4 VCs.

of the destination identifier. If n is not divisible by l, some gates will have an extra input. Notice that this implementation assumes that the number of VCs is a power of two. On the other hand, we would like to highlight that, as in DBBM, the assignment of destinations to VCs does not change as packet travels through the network. Therefore, this assignment can be performed only once when the packet is injected into the network. As a consequence, the network could be considered as several virtual independent networks, without interconnection among them, and the internal node switch can be implemented as several independent switches. As a consequence, the implementation of XORDET is very simple (as in DBBM) but it also allows the VCs to maximize its utilization (as in IODET).

XORDET is able to isolate traffic destined to different nodes and also balancing destinations among VCs. Figure 12.7 shows how destinations are distributed among VCs in a network with

Dim	VC#0	VC#1	VC#2	VC#3
X	14	14	14	14
Y	1	2	2	2

TABLE 12.3: How many destinations XORDET assigns to node 0 VCs in a 8×8 mesh with 4 VCs.

64 nodes and 4 VCs. As it can be seen, XORDET does a very good job, as traffic destined to either rows or columns will be distributed among the VCs. Table 12.3 shows how many destinations are assigned to each VC of node 0 of the network.

As shown, XORDET balances node destinations among VCs which will balance traffic for uniform random traffic pattern. But XORDET is a deterministic routing algorithm and this means that, for some adversarial traffic patterns, it may suffer from performance drops due to the limited routing flexibility. While XORDET works very well to avoid congestion caused by hot-spots, for some other traffic patterns, adaptive routing is able to outperform deterministic routing in general and, in particular, XORDET. For this reason, in this thesis we propose an adaptive HoL-blocking reduction routing algorithm that is able to combine the routing flexibility provided by adaptive routing with the destination isolation provided by HoL-blocking reduction routing to obtain optimal performance results for all traffic patterns.

12.3.2.2 XORADAP: XOR ADAPtive Routing

As mentioned above, deterministic routing lacks flexibility to adapt to some adversarial traffic patterns while adaptive routing does not encourage HoL-blocking effect reduction, which is very important for some traffic patterns. In particular, with a hot-spot node in the network, a HoL-blocking reduction deterministic algorithm that isolates the hot-spot traffic works better than adaptive routing [66] that spreads that traffic over the network avoiding other traffic to progress in the network. Let us analyze what happens in this case.

With adaptive routing, the problem arises in the VCs of all the network dimensions that provide the routing flexibility (i.e. the ones that can be used to cross the network dimensions without following any order). When there is a hot-spot node, adaptive routing trends to distribute traffic among all the available VCs, filling all the buffers with packets destined to the hot-spot node. Those packets will interfere with other traffic flows all over the network, thus creating the HoL-blocking problem.

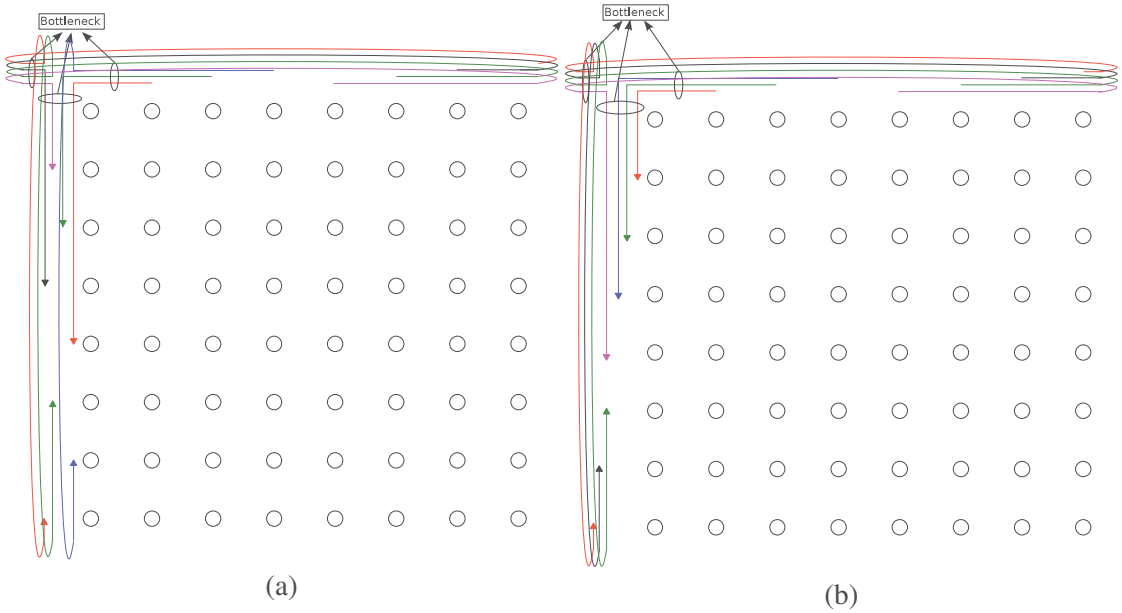


FIGURE 12.8: Paths of source-destination pairs of the first row with different pattern traffics: (a) Bit-reversal and (b) Matrix Transpose.

On the contrary, HoL-blocking reduction algorithms like IODET or XORDET have a very good behavior with hot-spot traffic because they confine the hot-spot traffic in just one of the VCs, allowing the rest of traffic to progress normally across other VCs. These routing algorithms also work well with uniform random traffic pattern as it will be shown, but they obtain a poor performance for some adversarial traffic patterns. For instance, consider the bit-reversal or matrix transpose traffic patterns [3]. In these cases, if deterministic routing is used, a lot of source-destination pairs will use the same links due to the destination distribution leaving many links unused. This fact creates a bottleneck since many messages have to cross the same link. We can see this behavior in Figure 12.8. This figure shows the paths used by the source nodes belonging to the first row in a 2D torus for the bit-reversal and matrix transpose traffic patterns using a deterministic routing algorithm. In particular, DOR was used. As it can be observed in the figure, the links of the topmost leftmost node become a bottleneck with deterministic routing. For these kinds of traffic patterns, adaptive routing can take advantage of all the network resources, providing a better utilization of the network links and therefore, improving overall network performance for this adversarial traffic pattens.

In order to provide flexibility for adversarial traffic patterns and also reduce the negative effects of HoL-blocking, we propose an adaptive HoL blocking-reduction routing algorithm. In this routing algorithm, VCs are organized as in a fully adaptive routing algorithm: there is a *group*

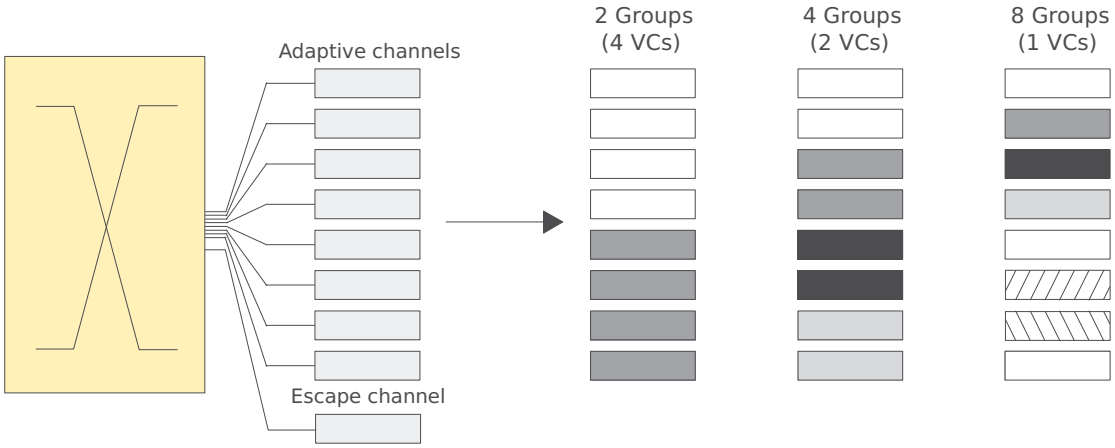


FIGURE 12.9: How XORADAP may assign VCs to groups with 8 VCs.

of adaptive VCs that can be used without restrictions and also there is an escape channel. We assume that bubble flow control is used in the escape channel. However, this routing algorithm confines each node destination identifier in a subset of the adaptive VCs instead of allowing the use of any of them. Contrary to deterministic routing, the routing algorithm allows crossing the network dimensions following any order (and therefore allowing more flexibility) but restricting the use of VCs depending on the destination node and thus confining the congested destinations in some VCs and allowing the packets located in the rest of VCs to progress. As a consequence, it provides some degree of flexibility but, at the same time, it limits the impact of the HoL-blocking effect because only a subset of the VCs can be used for a given destination.

To assign destinations to VCs, any mechanism could be used. In this thesis, taking into account its good balancing behavior, we propose to use a variant of the XOR function which is used in XORDET. For this reason, the resulting routing algorithm will be referred to as XORADAP (XOR ADAPtive). The VC assignment works as follows. We split the adaptive VCs into several groups. Each group can be composed of 1, 2 or more VCs. Given a packet, it will be forwarded to one of those groups depending on the packet destination, and any of the VCs of that group could be used.

As mentioned above, we use a function similar to the one used in XORDET, but, in this case, we select a group of VCs for each destination instead of just a single VC to classify traffic. In particular, with g groups of VCs, $l = \log g$ XOR gates are required. Given a packet destined to node $\{p_{n-1}, ..., p_1, p_0\}$, it will use the VC group given by the bits $\{VCG_{l-1}, ..., VCG_1, VCG_0\}$, computed as follows:

$$VCG_0 = p_0 \oplus p_{0+l} \oplus p_{0+2l} \ldots \oplus p_{0+(\frac{n}{l}-1)l}$$

$$VCG_1 = p_1 \oplus p_{1+l} \oplus p_{1+2l} \cdots \oplus p_{1+(\frac{n}{l}-1)l}$$

$$VCG_{l-1} = p_{l-1} \oplus p_{l-1+l} \oplus p_{l-1+2l} \cdots \oplus p_{l-1+(\frac{n}{l}-1)l}$$

Each of the XOR gates will have $\frac{n}{l}$ inputs, n being the number of bits of the destination identifier. As in XORDET, the $\frac{n}{l}$ inputs come from interleaved bits in the destination identifier. Notice that the number of groups og VCs must be a power of two.

As stated above, each group is composed of several VCs. Several configurations can be used. If there are V_a VCs available for adaptive routing, each group may contain from 1 to V_a virtual channels. Notice that, if we use only one group with all the virtual channels, we obtain the generic fully adaptive algorithm. On the contrary, if each group has only one VC, we obtain an adaptive version of XORDET that allows packets to cross dimensions following any order. Figure 12.9 shows an example of the different configurations that can be set for 9 VCs, one escape VC and 8 adaptive VCs. In this case, three configurations are possible for XORADAP: 2, 4 and 8 groups (with 4, 2 and 1 VC per group, respectively). As it can be seen, the resulting network is a set of different virtual networks, each one with several VCs. This means that packets of the different virtual networks are not mixed together, effectively separating flows. The escape channel is used by all the groups of virtual networks.

12.3.2.3 Implementation issues

As stated above, the different routing algorithms analyzed in this thesis demand different implementation complexity in the internal switch of the nodes. A fully demultiplexed crossbar [63] provides the highest flexibility, allowing connections among all input VCs to all the output VCs (i.e. it can map any input VC onto any output VC). In fact, such a switch is required for adaptive routing, where any input port can forward packets to any output port. However, in the case of deterministic routing, some of the connections provided by the internal switch are unused due to routing restrictions. For instance, if DOR deterministic routing is used, a packet can only use those ports that connect to the same or higher dimensions than the one it arrived. Therefore, the switches could be simplified if routing restrictions are considered, most important, without affecting performance.

Let us consider the routing algorithms proposed in this section. In XORDET, as the VC is selected as a function of the destination node, packets do not change the VC while they travel across the network, thus leading to a even simpler internal switch design than the traditional

deterministic routing with the same number of VCs. In XORDET, VC assignment can be done once at the source node, and the rest of nodes that a packet crosses across the network merely forwards the packet through the same VC from which the packet arrived to. Traditional deterministic routing with multiple VCs would have to select the output VC to forward the packet. As a consequence, as there is no need to move packets in a switch from one input VC to another output VC, the internal switch of the nodes can be implemented as one independent switch per VC (i.e. several virtual networks), instead of deploying a fully-connected crossbar, which is cheaper and faster, as switch delay depends on the number of switch ports [58, 75, 76].

In the same way, XORADAP also simplifies switch implementation. In this case, packets may change the VC used but they do not change the assigned group of VCs. The internal switch of the nodes can be implemented as one independent switch per group of VCs. Therefore, we could use a simpler internal switch design than fully adaptive routing. Notice that for a configuration of one group of all of the adaptive VCs, the complexity will be the same as fully adaptive routing.

Concerning routing mechanics, deterministic routing only requires applying the routing function [3] while adaptive routing requires the use of both the routing and the selection function [3]. In any case, both the output port and the VC to be used will be returned by the routing algorithm. For both XORDET and XORADAP, a few XOR logic gates are required at the source nodes to compute the corresponding VC or group of VCs, respectively. In XORADAP, a selection function is also required to select the VC inside the assigned group. However, the number of routing choices is smaller than with fully adaptive routing. As routing delay depends on the number of routing choices [58, 75, 76], XORADAP may lead to a faster implementation than fully adaptive routing.

12.4 Evaluation Results

In this section, some results are shown in order to discuss the different advantages of using the main proposals of this thesis. First, a brief description of the simulation tool is presented. After that, we show the results for the different proposals of the thesis.

12.4.1 Network Model

To perform the simulations, we have used a simulation environment developed at our research group. A prior version of this tool was used to provide evaluation results in [3]. The simulator models several topologies, including the new family of topologies presented in this thesis, the KNS. This simulator uses virtual cut–through switching. Each switch has a full crossbar with queues at their input and output ports. Credits are used to implement the flow control mechanism. We have performed the evaluation by using several synthetic traffic patterns: uniform, hot–spot and matrix transpose. In the uniform traffic pattern, message destination is randomly chosen among all destinations. In the hot–spot traffic pattern, a percentage of traffic is sent to one or a small subset of the processing nodes and the rest of the traffic is uniformly distributed. In matrix transpose, for 2-dimensional networks, the destination processing node is obtained by transposing the coordinates of the source processing node. Therefore, in this traffic pattern, the destination processing node of all packets generated at a given source processing node is always the same.

12.4.2 k_h–ary n_h–direct s_h–indirect (KNS) topology

In this section, we evaluate the KNS topology family, comparing its performance and cost with the ones provided by other topologies such as meshes, tori, fat-trees, and flattened-butterflies [11].

12.4.2.1 Performance Results

In this section, we compare the KNS using different indirect subnets (crossbar, fat-tree, and RUFT) against other well–known and frequently–used topologies such as tori, meshes, and fat–trees. Moreover, we also compare our proposal against the flattened–butterfly (FB) topology because it is becoming a popular topology in recent research papers. The FB topology is a variation of the butterfly topology obtained from using high-radix switches, that results in a direct topology. This topology can be seen as a generalized hypercube with concentration, as all the switches in the same dimension are directly connected, that is, there is a link from each switches to the others of the same dimension. Several FB configurations have been tested and compared to our proposal.

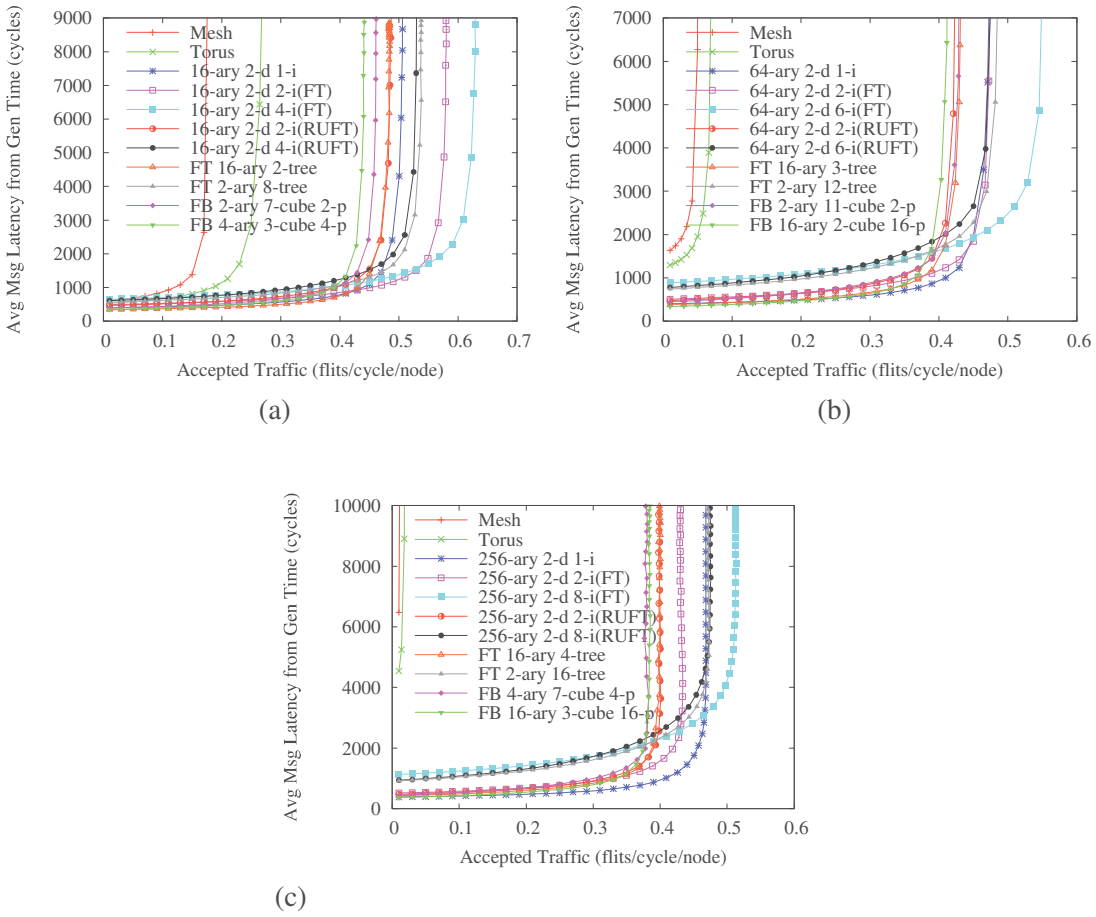


FIGURE 12.10: Average packet latency from generation vs. accepted traffic for uniform traffic and 2 dimensions for direct topologies. (a) 256 processing nodes. (b) 4K processing nodes. (c) 64K processing nodes.

We have evaluated a wide range of network sizes, from 256 to 64K processing nodes. Larger topologies have not been simulated due to simulator memory constraints. For direct topologies, we have tested different values of the number of dimensions and number of nodes per dimension. In particular, we show results for networks of 2 dimensions, with 16, 64, and 256 nodes per dimension; four dimensions, with 16 nodes per dimension; and eight dimensions, with 4 nodes per dimension. If not stated the contrary, only one processing node is attached to each router (i.e. without concentration). If several ones are attached, the x–p suffix is used, x being the number of processing nodes attached to each router. These networks are compared with fat–trees and FBs with the same number of processing nodes. Notice that, in some cases, several configurations are possible. For the sake of clarity, only a subset of the most representative simulations is shown.

Figure 12.10.(a) shows results for 2-D small networks (256 processing nodes) with uniform

traffic. As it can be seen, the mesh is the network that achieves the lowest throughput, followed by torus and the FBs configurations. In this latter case, we have selected a 4–ary 3–cube and a 2–ary 7–cube FB with concentration in order to compare our proposal against a topology with a hardware of similar complexity. The next topologies that achieve a better performance are the two fat–tree configurations. However, the best absolute throughput is achieved by the family of topologies proposed in this thesis. In particular, the different tested configurations ordered from lower to higher throughput are the topology that uses a RUFT with two stages as indirect subnet (16–ary 2–d 2–i (RUFT)), the ones that uses crossbar (16–ary 2–d 1–i), the one that uses RUFT with 4 stages (16–ary 2–d 4–i(RUFT)), the one that uses a FT with two stages (16–ary 2–d 2–i (FT)) and the one that uses a FT with four stages (16–ary 2–d 4–i (FT)). In particular, the best configuration of the new topology family obtains 3 times more throughput than the worst network (mesh), more than twice versus torus, more than 20% versus FT and about 40% improvement versus FB.

Figures 12.10.(b) and 12.10.(c) show how throughput is decreased in all the topologies as we increase the number of processing nodes in the network, keeping constant the number of dimensions in the direct topologies, and therefore increasing the number of routers (processing nodes) per dimension. In the case of mesh and torus topologies, throughput strongly decreases, as the average distance between two nodes is markedly higher than in the other topologies. Regarding the KNS topologies, all the tested configurations outperform both the FT and FB configurations analyzed. The best configurations are again the ones that use the tallest FT as indirect subnet. Notice that the different topologies have a different hardware cost, which is evaluated in following section.

In Figures 12.10.(a), 12.10.(b), and 12.10.(c) we can also see the impact of using more stages in the MINs of KNS topologies. For the same number of routers per dimension, if we decrease the number of stages, the arity of the switches is increased, and a lower latency should be obtained. The plots show that, the higher the number of stages, the higher the base latency (in more detail, the zero-load latency) as more switches have to be crossed by packets. Surprisingly, networks with more stages also achieve more throughput. This effect is explained by the reduction of the head-of-line (HoL) blocking effect. For a given number of routers per dimensions, a taller FT uses smaller switches (i.e. with lower number of ports). As a consequence, each switch port is potentially demanded by a lower number of input ports and, hence, the effect of HoL blocking is reduced. From another point of view, with fewer stages, each indirect topology has less switches

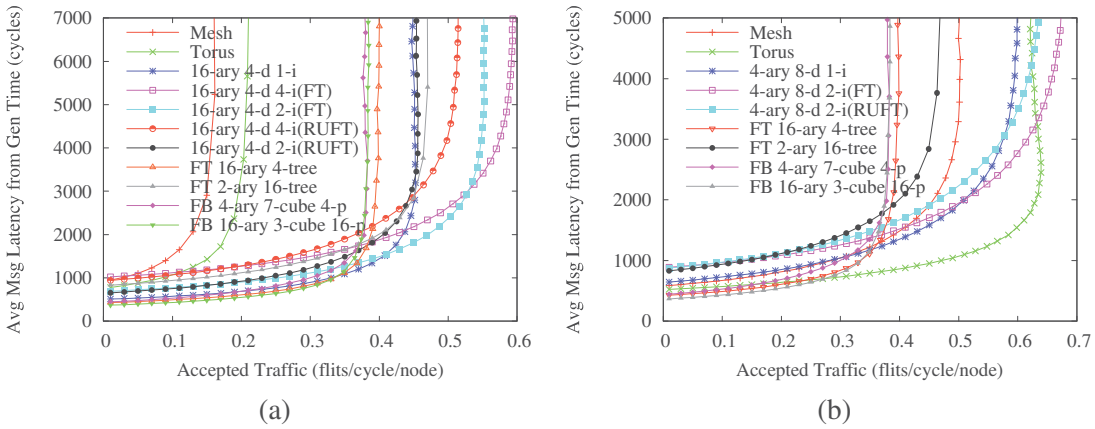


FIGURE 12.11: Average packet latency from generation vs. accepted traffic for uniform traffic with 64K processing nodes and different number of dimensions: (a) 4D and (b) 8D.

to serve the same number of routers. Thus, each switch has to deal with more traffic, leading to more HoL blocking effect and, hence, less throughput.

Let us analyze the base latency. In a k_h–ary n_h–direct 1–indirect, base latency does not depend on the number of routers per dimension. However, in KNS that uses MINs, the base latency increases with the number of processing nodes because the number of stages in the indirect subnets also grows in order to connect a larger number of routers. This effect is more prominent in RUFT, due to the fact that packets traverse always all stages since it is a UMIN topology. In the case of torus and mesh, base latency strongly depends on the number of nodes per dimension, as average distance between nodes is increased.

Figure 12.11 analyzes the impact of the number of dimensions in the different topologies. We analyze a network with 64K processing nodes implemented with a different number of dimensions. We can distinguish three different behaviors. First, Mesh and torus topologies have a similar behavior. The higher the number of dimensions, the fewer the number of nodes per dimension, and the higher the achieved throughput. Also, base latency decreases with the number of dimensions, because the average distance is reduced. However, the behavior of k_h–ary n_h–direct 1–indirect is different. Throughput also increases with the number of dimensions because the size of switches of the indirect network (a crossbar in this case) is reduced, and, hence, the pernicious effect of HoL blocking is reduced. However, base latency does not improve with the number of network dimensions. This is due to the fact that the number of hops that packets must perform also grows with the number of dimensions. Concerning the k_h–ary n_h–direct s_h–indirect, they have a similar behavior to the previous one, but with a difference. The base

latency, in this case, slightly decreases when the number of dimensions increases. Although network diameter increases with the number of dimensions, as started above, as there are fewer routers per dimension, indirect subnets have fewer stages, and, thus, packets have less stages to cross. Finally, the configurations of the FB shown and FT obtain an intermediate throughput value. Anyway, we would like to remark that the new family of topologies always obtains the best throughput regardless of the number of dimensions.

12.4.2.2 Cost–performance analysis

This section estimates and compares the hardware cost of each considered topology and analyzing the performance-cost ratio obtained for each of them.

In order to get an actual cost figure, we have calculated the cost (in $) that some of these configurations would have when implemented with real commercial products. We have used InfiniBand products with FDR technology of Mellanox [68] (February 2015) to calculate the cost.

When preparing the budget, if there are no switches with the number of ports required by the configuration, we selected the next one with greater number of ports.

As copper links are limited to 5 meters, if a longer link is required, fiber links must be used. We assumed an average length between cabinets (global links) of 10 meters, and 2 meters for connections in the same cabinet (local links).

The number of global and local links depends on the topology. For 256–ary 2–direct 1–indirect and 256–ary 2–direct 4–indirect with FT as subnets, the processing nodes of the same first dimension can be placed in the same cabinet or in two cabinets (one beside the other). The links of this dimension are local links, and the links of the second dimension are global links. Regarding the links interconnecting the stages of the MIN (a fat–tree in this case), we assume that they are local, since each subnet fits in a cabinet. In flattened–butterfly configurations, for example FB 16-ary 3-cube 16-p, we use the same approach. The links of the first dimension are local because the processing nodes of the same first dimension are placed in one or two cabinets and the links of the remaining dimensions are global. In fat–trees, the links which connect the processing nodes with the first stage are local, and the remaining links are global. In torus topology, the configuration is very similar to k_h–ary n_h–direct 1–indirect (local links for the first dimension and global links for the second dimension). However, in this case, a cabinet or a group of cabinets that contain processing nodes of the same first dimension, are connected

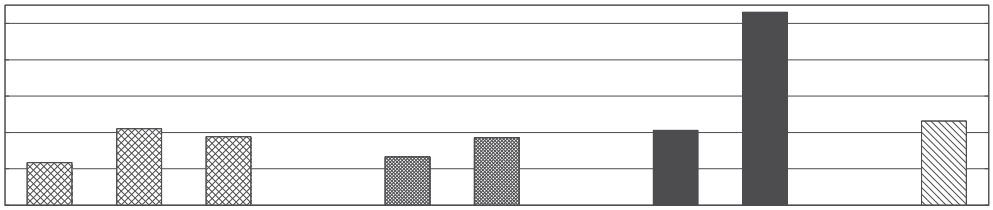


FIGURE 12.12: Total cost of different topology configurations with 64K processing nodes.

TABLE 12.4: Cost–performance analysis for different topology configurations with 64K processing nodes. Throughput is measured in flits/cycle/node. Throughput/cost is measured in flits/cycle/node/$.

Topology	256-ary 2-direct 1-indirect	256-ary 2-direct 2-indirect (FT)	256-ary 2-direct 2-indirect (RUFT)	FB 16-ary 3-cube 16-p	FB 4-ary 7-cube 4-p	FT 16-ary 4-tree	FT 4-ary 8-tree	Torus 256-ary 3-cube
Throughput	0.47	0.43	0.40	0.39	0.38	0.40	0.41	0.02
Total Cost ($)	235 M	420 M	376 M	266 M	371 M	412 M	1,062 M	463 M
Throughput /Cost	2.00 x10^{-9}	1.02 x10^{-9}	1.06 x10^{-9}	1.47 x10^{-9}	1.02 x10^{-9}	0.97 x10^{-9}	0.39 x10^{-9}	0.04 x10^{-9}

to the neighboring cabinet or group. So, they will be very close. For this reason, in this case we have used shorter global links of 5 meters.

Using these data, we calculated the cost of some selected configurations, which are shown in Figure 12.12. The configurations shown are the cheapest ones that their performance is not very far to the configuration that obtains the highest throughput.

As can be seen in Figure 12.12, the 256–ary 2–direct 1–indirect configuration obtains the lowest absolute cost. The fat–tree with 8 stages (FT 4-ary 8-tree)has a very high cost, despite having very good performance. In the case of torus, its cost is not very high, but it has a low performance. Flattened–butterfly has a competitive cost, but it is not lower than the 256–ary 2–direct 1–indirect and it does not reach a better performance.

To allow a better comparison of both cost and performance, Table 12.4 shows the ratio between cost and performance. If we use KNS topologies, configurations with more stages have better throughput but also a higher cost and more latency. The same applies to fat-trees. The k_h–ary n_h–direct 1–indirect combines a good performance with a low cost. Although the torus configuration is not very expensive, it has a very poor performance. Flattened–butterflies are

not very expensive and obtain good performance. However, the k_h–ary n_h–direct 1–indirect configuration obtains the best absolute results in terms of performance-cost ratio. As it can be seen, the worst ratio is provided by the torus, followed by a configuration of the fat–tree (FT 4-ary 8-tree).

12.4.3 Fault tolerance for k_h–ary n_h–direct s_h–indirect topologies

In this section, we analyze the fault-tolerance capability of the new family of topologies when using the routing algorithm methodology presented in Section 12.2. To evaluate the proposed fault-tolerance methodology, we have performed two kind of analysis. First, we analyze the number of network failures that can be tolerated. Remember that a fault-tolerant routing algorithm is able to tolerate f failures if it can provide a valid path between every source-destination pair for any combination of f failures. Notice that there are situations where the failures physically disconnect the network. We consider these situations as combinations where there is no path for all source-destination pairs and therefore the combination is not tolerated.

Second, we evaluate the network performance degradation in presence of faults when using the proposed methodology. To do this, we have simulated different tolerated network configurations with a varying number of faulty links under uniform traffic. For each number of faults, we have tested 50 random fault combinations to obtain the average network throughput and latency. In those experiments, the combinations where some nodes are physically disconnected are discarded and not simulated since only tolerated combinations are simulated.

12.4.3.1 Fault Analysis

The number of possible fault combinations exponentially increases with the number of faults. For this reason, it is not possible to explore all possible fault combinations for a given number of faults in a reasonable amount of time. Therefore, we have used statistical analysis as a tool. Specifically, we have analyzed a subset of the fault combinations, where the faults are randomly chosen. This subset is large enough to obtain results with a confidence level of 99% and an error lower than 1%.

Figures 12.13 and 12.14 show the percentage of tolerated fault combinations for different number of faults using one or two intermediate nodes for a 2-D network with 1,024 nodes and a 3-D network with 1,000 nodes, respectively. The results are shown for a number of link faults

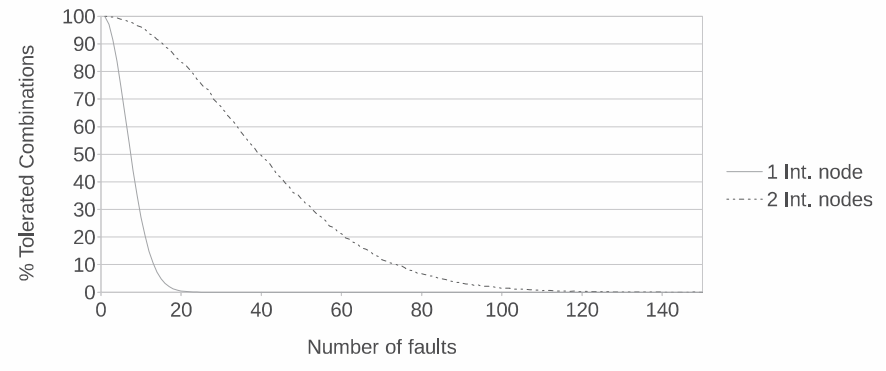


FIGURE 12.13: Fault combinations tolerated by the methodology when using one or two intermediate nodes in a 2-D network with 1,024 nodes.

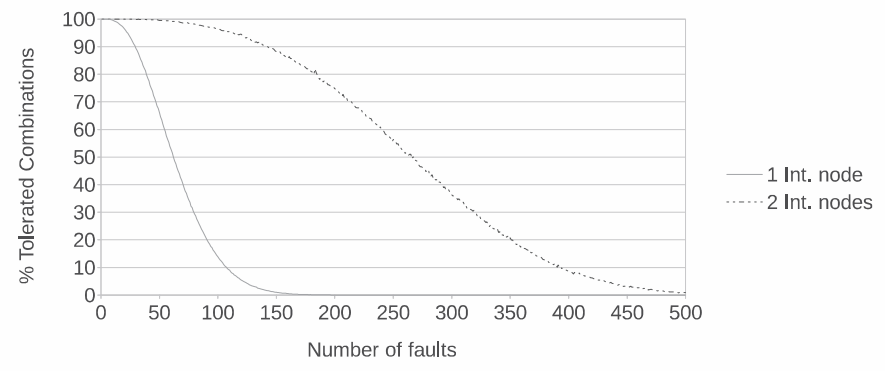


FIGURE 12.14: Fault combinations tolerated by the methodology when using one or two intermediate nodes in a 3-D network with 1,000 nodes.

up to 150 for 2-D networks and 500 for 3-D networks. First, as expected, the percentage of tolerated combinations of faults strongly increases when using two intermediate nodes instead of only one because there is more control over the path followed by the packet. That is, using two intermediate nodes instead of only one provides more alternative paths or, what is the same, we have more options to configure the final path to the destination node, avoiding the faults. On the other hand, in the 2-D network, the percentage of tolerated combinations of faults is considerably lower than in the 3-D one because more alternative paths are available in the latter for each source-destination pair. In the case of the 3-D network, the methodology is able to tolerate more than 99.5% of the 10-fault combinations with only one intermediate node and more than 99.98% for the configurations of 15 faults with 2 intermediate nodes. The fact of having more dimensions gives more probability to route the packet through different paths that do not share resources, being able to avoid more faults. However, the 3-D network has more resources. There are 3,000 links in the 3-D network versus 2,048 links in the 2-D network. This is why, for 23 faults in the 2-D network even with two intermediate nodes, the percentage of tolerated

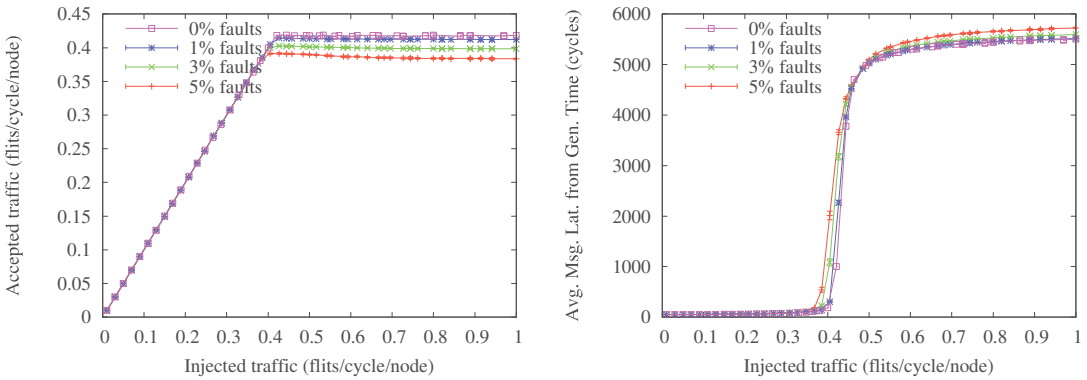


FIGURE 12.15: 32-ary 2-direct 1-indirect under uniform traffic: (a) Accepted traffic and (b) average latency versus injected traffic.

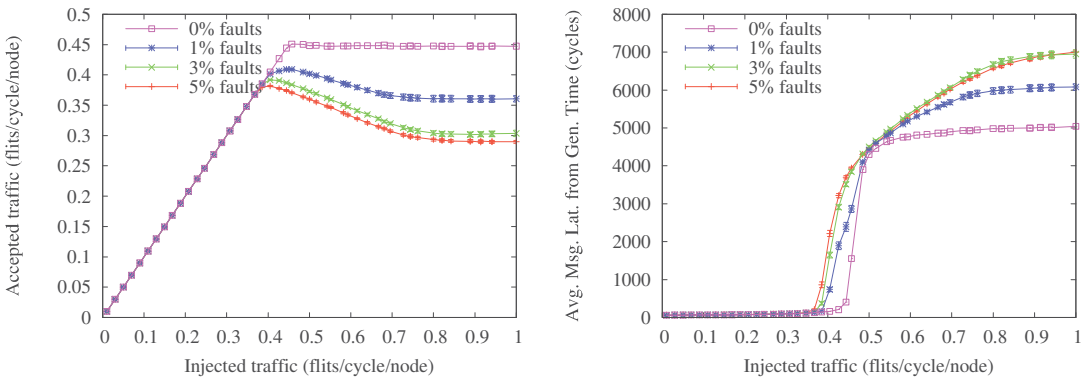


FIGURE 12.16: 10-ary 3-direct 1-indirect under uniform traffic: (a) Accepted traffic and (b) average latency versus injected traffic.

combinations is lower than 80%. However, in the 3-D network, more than 100 faults are needed to reach a percentage of tolerated combinations lower than 80%.

12.4.3.2 Performance Analysis

In this Section we analyze the performance degradation suffered by the network when applying the fault-tolerant routing methodology in presence of faults. To do this, we have simulated several network scenarios with 1% faulty links, 3% faulty links and 5% faulty links. For each fault scenario, we have generated 50 random fault combinations, all of them tolerated by the methodology. That is, all source-destination pairs are able to communicate. Up to 2 intermediate nodes can be used, since this allows to test fault combinations with a higher number of faults.

However, the fact of using two rather than only one intermediate node does not strongly impact network performance. This is because the methodology is able to avoid the fault using only one intermediate node in most of the cases, although it is able to use two.

In order to measure the performance degradation, we obtained the accepted traffic and average latency form generation time versus injected traffic. We consider average latency form generation time to consider the time spent by the packet at the injection queue. Accepted traffic is measured as the amount of data per node and per time that the network can accept (flits/cycle/node). Network throughput is the peak value of accepted traffic. Average latency form generation time is measured as the mean of the elapsed time from message generation at the source node until its ejection at the destination node. For network load, source nodes inject traffic following a uniform traffic pattern (i.e., randomly selecting the destination node).

Figures 12.15.(a) and 12.15.(b) show results for a 32-ary 2-direct 1-indirect network under uniform traffic. In this case, the network suffers a performance degradation of about 1% in throughput with 1% faulty links (21 links) compared to the same network without faults. For 3% and 5% faulty links, performance degradation increases to 3,8% and 6,5%, respectively. Latency is affected as well, increasing the average value with respect to the fault-free case. In particular, at saturation, latency is increased by 67% with 1% faulty links.

For the 10-ary 3-direct 1-indirect network (Figures 12.16.(a) and 12.16.(b)) the performance degradation, for the same percentage of faulty links, is higher when compared to the 32-ary 2-direct 1-indirect network. In particular, network throughput degrades by about 9% with 1% faulty links (30 faults) compared to the fault-free network, and by 13% and 15% with 3% and 5% faulty links, respectively. The increase in latency for the 1% link faults case, is 1.8 times at the saturation point. Notice that the increase in latencies is because the network with faults saturates before than the one without faults. If we focus on the base latency or the latency before saturation, we can see that there is almost no impact. This is because the use of intermediate nodes is only required for a low percentage of source–destination pairs. Although both networks have roughly the same number of nodes, they have a different number of links and, for the same number of relative link faults, the 3-D network involves more faulty links. Therefore, more paths are affected. On the other hand, as shown in Figures 12.13 and 12.14, the fact of having a higher number of dimensions with the same number of nodes improves the probability of avoiding a given fault combination. Therefore, although throughput is degraded by 3.8% with 3% faulty links in the 2-D network versus a performance degradation of 13% with 3% faulty links in the

3-D network, the probability of supporting a combination with this number of faults in the 3-D network is about 97%, against only 16% in the 2-D network.

12.4.4 Reducing the HoL-blocking effect in Direct Topologies

In this section, we evaluate by simulation the HoL-blocking reduction routing algorithms proposed in this thesis (IODET, XORDET and XORADAP) comparing them with previously proposed ones. First, we will compare the deterministic routing algorithms, IODET and XORDET, with other HoL-blocking reduction deterministic algorithms like DBBM, BBQ, VOQnet and VOQsw. We will also consider a fully adaptive routing algorithm and a DOR deterministic routing which allows packets to use all the VCs of the selected dimension, that is a deterministic routing algorithm without destination node classification. Notice that this latter algorithm is actually partially adaptive (as it allows several routing options) and does not guarantee in-order delivery of packets. For this reason, we will refer to it as Out of Order DETerministic routing (OODET). To guarantee deadlock-freedom in tori, the bubble flow control mechanism was used (either in all the VCs for deterministic routing or in the escape VC for fully adaptive routing).

After the evaluation of IODET and XORDET, we will evaluate XORADAP to analyze how it behaves under different traffic patterns and we will show how a hybrid approach is able to combine the best of two worlds and obtain good performance results for any traffic pattern.

Regarding the number of VCs per physical channel, it must be a power of two in XORDET. In XORADAP, the number of groups of VCs must be a power of two and also an escape channel is required. To perform a fair comparison, for traditional fully adaptive routing, we will use the same number of VCs as the one used in XORADAP. Source nodes implement VOQnet in the injection. This means that messages with different destinations do not harm the injection of each other.

12.4.4.1 Deterministic routing algorithms evaluation

First, we will analyze the behavior of IODET and XORDET versus the other HoL-blocking reduction deterministic routing algorithms. Figure 12.17 shows the obtained results for a 2D torus with 256 nodes and uniform random traffic pattern. With only a few number of VCs (4 or 8), any HoL-blocking algorithm is able to reach nearly the same performance as VOQnet, which is the upper bound. The exception is DBBM that, due to its poor destination classification in the

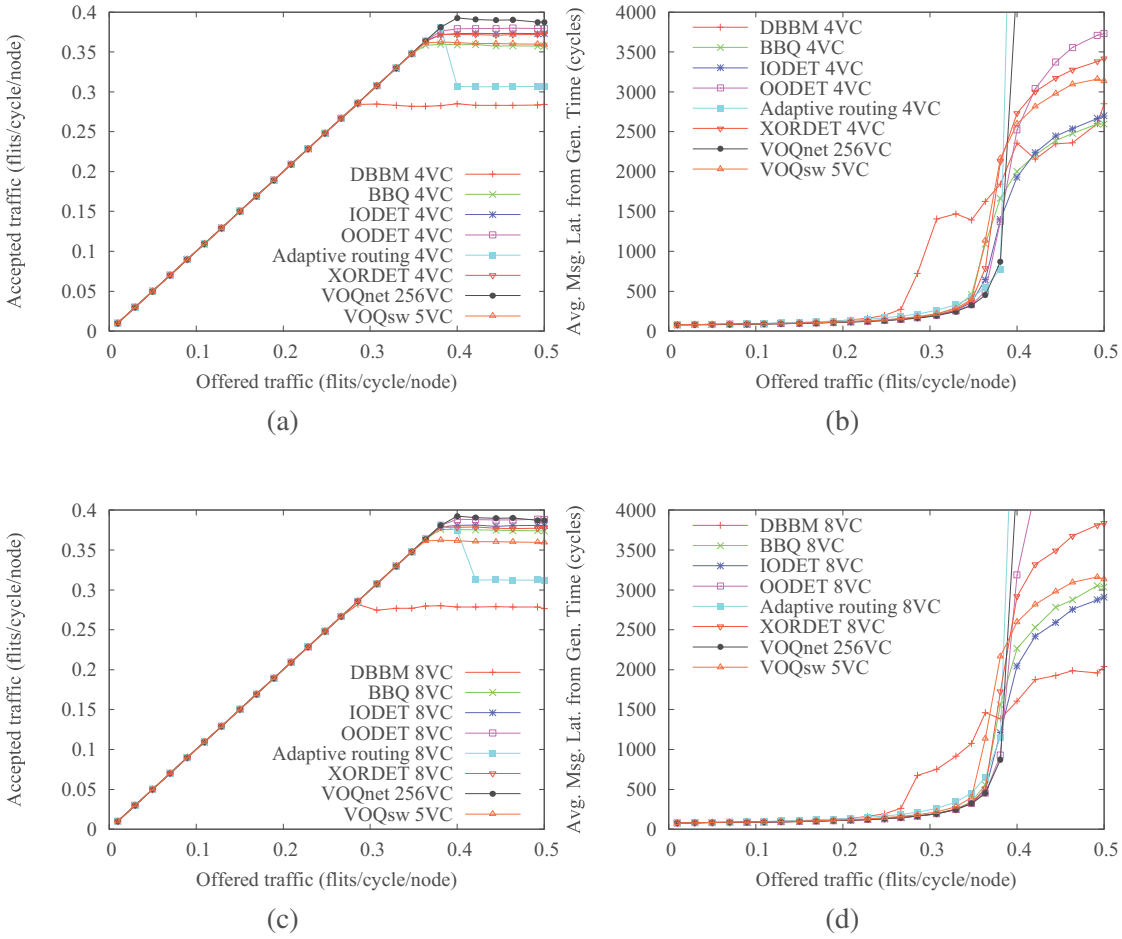


FIGURE 12.17: Average packet latency and accepted traffic vs offered load. 256-node 2D-torus. Uniform random traffic pattern. (a,b) 4 VCs and (c,d) 8 VCs.

last dimension, it obtains a worse performance. Notice the importance of the dimension ordering followed by the routing algorithm. Unexpectedly, BBQ, which also considers a subset of the node destination identifier bits to classify packets, works quite well. The difference between DBBM and BBQ is that the former consider the least significant bits while the latter considers the most significant ones. As IODET and XORDET considers all the node destination bits to classify packets, it should not be affected by changes in the dimension ordering followed by the routing algorithm. On the other hand, fully adaptive routing suffers the typical performance rollback after network saturation [85].

In Figure 12.17, the traditional DOR routing following XY order was used. However, using other deterministic routing algorithms could be interesting. For instance, in [65], X+Y+Z+X-Y-Z- direction-order routing was proposed instead of dimension order routing for fault tolerance purposes. Direction order routing allows packets to be routed in both directions of a dimension

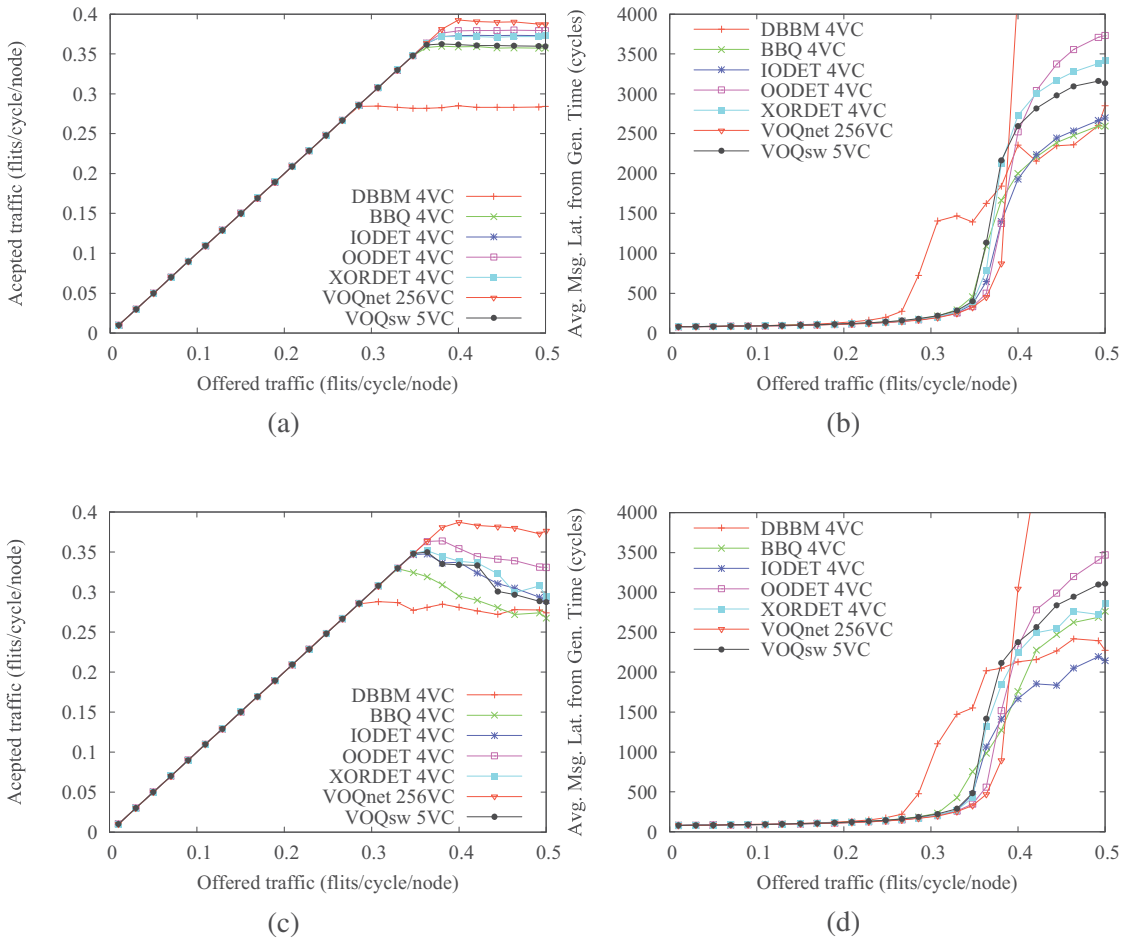
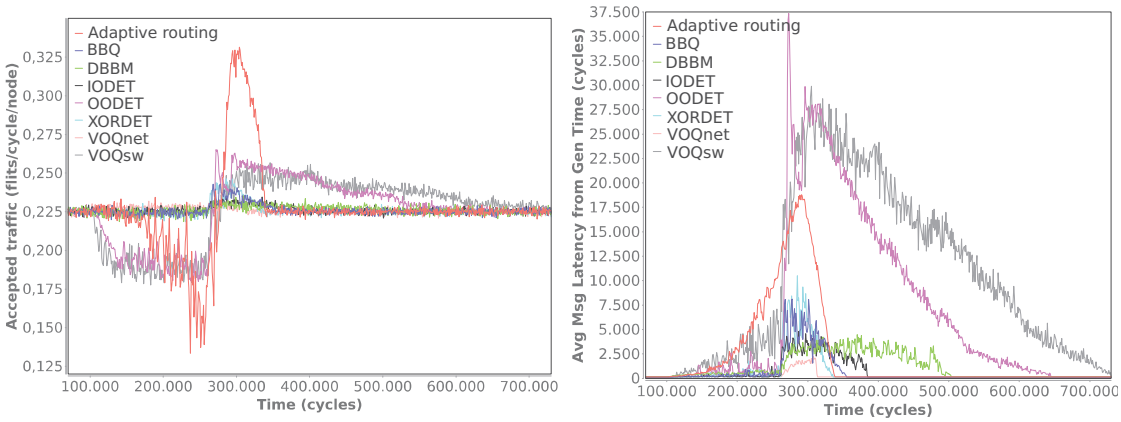


FIGURE 12.18: Average packet latency and accepted traffic vs offered load. 256-node 2D-torus. Uniform random traffic pattern. (a,b) XY routing and (c,d) X+Y+X-Y- routing. 4 VCs.

and, therefore, offers greater flexibility to avoid faults. Furthermore, direction order routing allows routing through non-minimal paths. For 2D networks, the X+Y+X-Y- direction order routing counterpart would be used. Packets are routed following an ascending dimension order, but taking first the positive dimension directions, and then the negative ones. Figure 12.18 shows evaluation results for the different HOL-blocking reduction mechanisms but using X+Y+X-Y- with minimal paths as the baseline deterministic routing. We can see how the fact of traversing dimensions in a different order changes the behavior of some algorithms like BBQ, which drives down its performance significantly. This effect is similar to the one produced by DBBM before and is due to the fact of using a subset of the destination node identifier bits to select the VC to use. The dimension ordering followed by the routing algorithm may generate an unfair use of the VCs, overloading some of them while others are barely used. However, XORDET or IODET, which consider all the destination node identifier bits, are less affected by the change



(a) (b)

FIGURE 12.19: Results for hot-spot. 256-node 2D-torus and 8 VCs.

in the routing algorithm and obtain roughly the same performance as the one obtained with XY routing.

After analyzing the behavior of the different algorithms under uniform random traffic pattern, next we analyze them under other traffic patterns. We will analyze a scenario where the HoL-blocking reduction ability of the routing algorithm may have a great impact. Assume that we have uniform random traffic pattern in the network, but we also introduce a hot-spot node: 25% of network nodes send packets only to one node (the hot-spot node) during some period of time. Traffic injection rate to the hot-spot is computed in such a way that it does not exceed the node ejection bandwidth (1 flit/cycle). The hot-spot traffic starts at clock cycle 100,000 and is active until a number of packets (10,000 in our experiment) have been delivered. This corresponds to clock cycle 260,000. In addition, the remaining nodes (75%) continue generating traffic following a uniform random traffic pattern, that is, sending packets to all the destinations except the hot-spot node. Therefore, during this period of time, the network has two traffic flows: 75% of nodes generate packets with an uniform random traffic pattern and 25% generate packets destined to the hot-spot node. In such a situation, a HoL-blocking reduction routing algorithm should be able to isolate the traffic destined to the hot-spot (i.e. hot flows), thus avoiding interfering the other flows (i.e. cold flows). On the other hand, a fully adaptive algorithm mixes the different flows, spreading the possible congestion to the whole network. To perform this experiment, we have implemented large injection queues at source nodes so that they always can queue a packet if the packet cannot be injected into the network.

This scenario is evaluated in Figure 12.19 for a 256-node 2D torus. We can see a completely

different behavior of the analyzed routing algorithms. On the one hand, fully adaptive routing, OODET and VOQsw rapidly spread congestion as packets destined to the hot-spot node interferes other packets, leading to a high reduction in the delivered traffic rate (Figure 12.19.(a)) and strongly increasing latency (Figure 12.19.(b)). Only when the hot-spot traffic disappears and after a high number of cycles, the network recovers. Notice that, after the hot-spot traffic is removed, accepted traffic increases for some cycles, due to the high number of messages that have been queued at the injection nodes.

On the other hand, the HoL-blocking reduction deterministic routing algorithms show a much better behavior, close to the one of VOQnet (which requires 256 VCs) without impacting the network throughput and latency in spite of the hot-spot traffic. The exception is DBBM, which requires a higher number of cycles to recover from the hot-spot traffic. This is because the injected uniform random traffic pattern is on the edge of saturation in DBBM.

To summarize, XORDET and IODET were able to reach (with only a few VCs) the same performance as fully adaptive algorithm for uniform random traffic pattern (see Figure 12.17). Contrary to DBBM and BBQ, they are less affected by changes in the routing algorithm (i.e. the order in which dimensions are crossed, see Figure 12.18) and they are able to efficiently isolate the hot-spot traffic (see Figure 12.19). The advantage of XORDET versus IODET is that it is simpler to implement at the internal switch. Remember that XORDET uses virtual networks, but IODET performs VCs changes in the network, which requires additional internal switch connections.

12.4.4.2 XORADAP evaluation

We will first analyze XORADAP with uniform random traffic pattern. In this case, we only show results of XORDET as representative of deterministic routing algorithms with the HoL-blocking reduction ability to offer greater clarity, since the other routing algorithms work worse or equal to XORDET. Figure 12.20 shows the results (8 VCs for fully adaptive routing and 1 VC for the escape path). For XORADAP, we selected three different configurations with 9 VCs: two groups with 4 VCs each, 4 groups with 2 VCs and 8 groups with only one VC per group. Remember that more groups of VCs leads to a better packet classification but a lower routing flexibility. Regarding fully adaptive routing, we used the same number of VCs as XORADAP for the sake of fairness. As the number of VCs in XORDET must be a power of two, we evaluated it by using both 8 and 16 VCs.

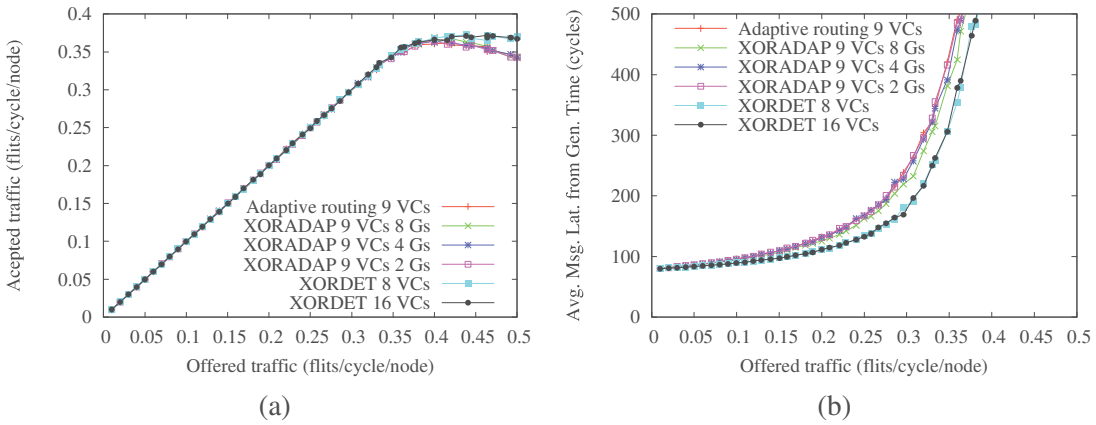


FIGURE 12.20: 256-node 2D-torus. Uniform random traffic pattern. (a) Accepted traffic and (b) average packet latency vs. offered traffic.

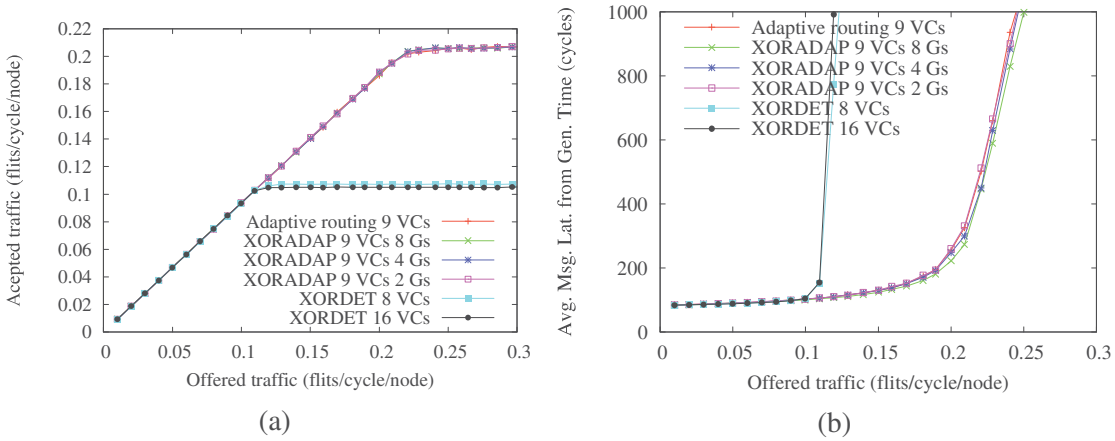


FIGURE 12.21: 256-node 2D-torus. Matrix transpose traffic. (a) Accepted traffic and (b) average packet latency vs. offered traffic.

We can see that all the routing algorithms evaluated obtain roughly the same throughput. Notice, though, that the fully adaptive algorithms suffer a performance degradation after its saturation point [85]. Regarding latency, (Figure 12.20.(b)), for medium to high traffic rates (i.e. 0.3 flits/cycle/node), a higher routing flexibility (i.e., fully adaptive routing or XORADAP with less number of groups of VCs) leads to higher latency values due to the HoL-blocking effect generated by interfering traffic flows. We can see how the XORADAP routing algorithm with more groups of VCs, less adaptive behavior, obtains a slightly lower latency. Both configurations of XORDET obtain the lowest latency values, with almost no differences between them.

As we mentioned in Section 12.3.2.2, there are some adversarial traffic patterns that significantly impact the performance of the network with deterministic routing algorithms. Nevertheless,

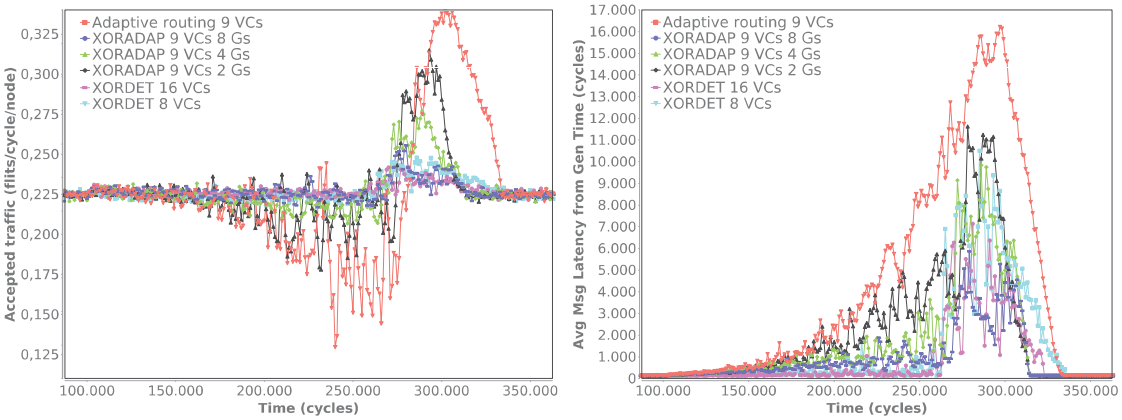


FIGURE 12.22: 256-node 2D-torus. Uniform random traffic pattern with hot-spot. Accepted traffic (a) and average packet latency (b) vs. simulation time.

XORADAP obtains good performance results not only with uniform random traffic pattern, but it is also able to obtain good results with those adversarial traffic patterns.

To illustrate this behavior, we have conducted some experiments with the matrix-transpose traffic pattern. In Figure 12.21, we compare the behavior of XORDET, fully adaptive routing and the different configurations of XORADAP for the matrix transpose traffic pattern in a 256-node 2D torus. 9 VCs (8 adaptive channels and 1 escape channel) were used in fully adaptive and XORADAP routing algorithms, and 8 VCs and 16 VCs in XORDET. In XORADAP, the three aforementioned configurations were tested: two groups with 4 VCs each, 4 groups with 2VCs and 8 groups with only one per group.

As expected, XORDET obtains a significantly lower throughput than any adaptive algorithm, in spite of using more VCs. In particular, fully adaptive routing more than doubles XORDET performance. This is the weakest point of deterministic routing. It is not able to efficiently cope with adversarial traffic patterns. The poor behavior of XORDET, and, in general, of any deterministic routing, is due to the unbalanced distribution of traffic for this pattern, which leads to overutilization of some links while other are unused [86]. Concerning the hybrid routing algorithm proposed in this thesis, XORADAP, it obtains roughly the same results as fully adaptive routing, since it takes advantage of its flexibility making a better use of the links.

Considering the results presented up to now, we can confirm that XORADAP achieves its first design goals. It is as good as fully adaptive routing for adversarial traffic patterns, thus improving XORDET and deterministic routing in general.

Next, we will analyze XORADAP behavior in the hot-spot scenario, where the HoL-blocking reduction is very important. Figure 12.22 shows the results for the same experiment performed in Section 12.4.4.1. Remember that the hot-spot traffic starts at clock cycle 100,000 and it is active until clock cycle 260,000. As expected, XORADAP helps to achieve a better behavior than fully adaptive routing. In particular, XORADAP configurations with more groups of VCs can better isolate the hot-spot traffic flows, obtaining a more stable value of accepted traffic (in fact very close to XORDET in the best case -XORADAP with 8 groups of VCs-) and a smaller average packet latency. On the other hand, if we use a XORADAP configuration with a few number of groups of VCs, two for example, we obtain a result more close to fully adaptive routing, but with smaller impact on the variability of delivered traffic and reducing packet latency with respect to fully adaptive routing.

The analysis shown before demonstrates that XORADAP also achieves its second design goal. It can be as good as a HoL-blocking reduction deterministic routing algorithm to classify and isolate traffic, outperforming fully adaptive routing under hot-spot traffic. To sum up, XORADAP routing algorithm combines the flexibility of adaptive routing with HoL-blocking reduction, being able to efficiently cope with varying networks loads, including uniform random traffic, adversarial or hot-spot traffic. Indeed, for a given number of VCs, several configurations are possible.

Chapter 13

Conclusions

This thesis has proposed a new family of hybrid topologies, the KNS, for large–scale interconnection networks. It keeps the dimensional organization from direct topologies, but connecting the nodes of a dimension in a different way which is able to provide as good performance and scalability as indirect topologies do, but with a low hardware cost.

In addition, this dissertation has presented a routing algorithm for this family of topologies. This routing algorithm a deterministic routing algorithm, Hybrid-DOR, and it is based on the DOR algorithm for direct topologies. This algorithm has a good behavior, obtaining good performance with a low latency.

On the other hand, this thesis proposes a new fault-tolerant mechanism for this new family of topologies. It is based on Valiant mechanism. In this case, a packet which have to pass across a path with one or more faults could avoid these faults by using intermediate nodes which allows using another path which, for some fault combinations, could be longer but it allows packets to achieve their destination.

Finally, three different routing algorithms have been presented in this thesis in order to reduce the HoL-blocking effect in direct topologies. The first one, IODET, takes advantage of the dimensionality of direct topologies, classifying packets to the different virtual channels depending on the destination identifier component in the dimension where the packet is traveling. The second one, XORDET could be implemented at a lower cost and easier implementation than other proposals because the packet should not change the virtual channel to reach its destination. It uses the XOR operation, xoring the bits from the destination identifier to select a virtual channel. And the last one, XORADAP, uses the same XOR operation with an adaptive algorithm, using

several adaptive channels and one escape channel at least. The adaptive channels allow packets to cross the network following any dimensional order. But in this case, the XOR operation is not used to choose a virtual channel. The adaptive channels are divided in different groups, with one or more virtual channels. Packets are confined to these groups depending on the XOR operation.

The deterministic algorithms, IODET and XORDET, work as expected regarding isolation of congested traffic flows (e.g. hotspot nodes.) Despite this, the performance drops when there are some adversarial traffics like bit-reversal or transpose patterns. However, XORADAP, thanks to its routing flexibility, it is able to deal with these adversarial traffic patterns, without losing the feature of isolating congested traffic flows.

13.1 Future Directions

As for future work, first, we plan to improve the fault-tolerant mechanism for KNS topologies. In this thesis, a mechanism is presented, and we describe how the algorithm works with a given fault information. However, we have not studied yet how to distribute this information to the different nodes and switches and how it could affect to the network performance. It can be done statically or dynamically. The static fault model is easier because we only need to stop the network and give the information to the different network components. But, to do this, we need to be able to do checkpoints to save the status of the network and restore it when the fault information is delivered. On the other hand, in the dynamic fault model, the fault information is sent while the network is working. Therefore, the time to reconfigure the network is very important. The idea could be to develop both of them, having different options to solve the problem in different scenarios.

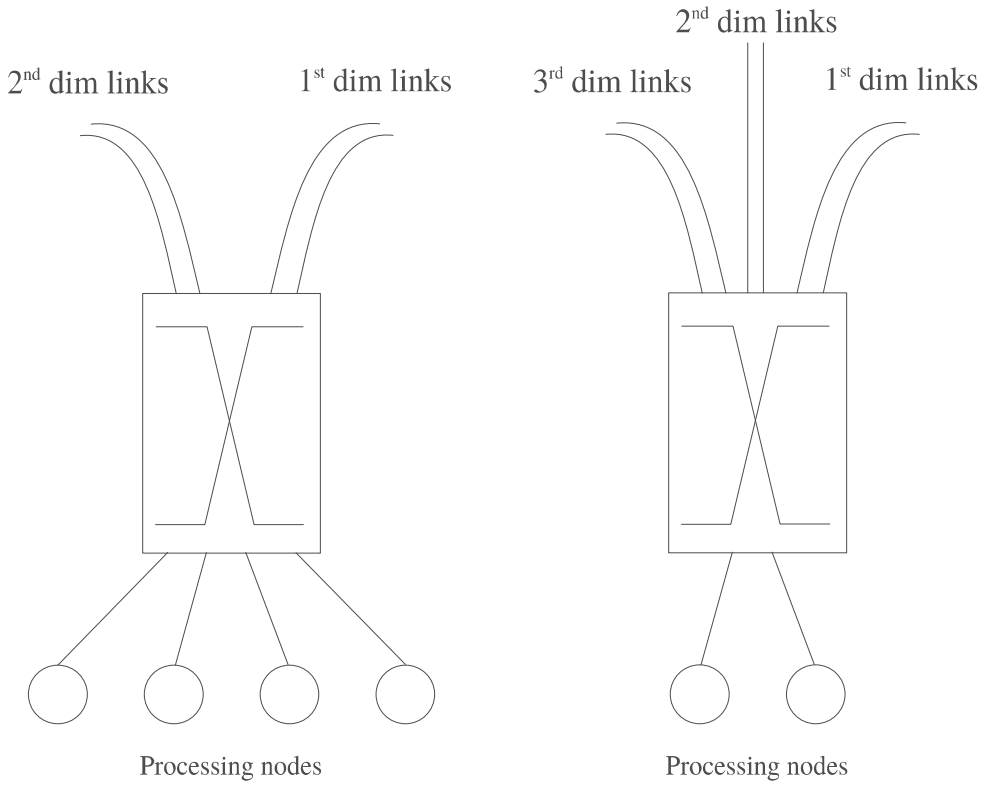


FIGURE 13.1: Different configurations for 8-port switches.

There are other aspects of KNS that can be studied, like attaching several processing nodes to the same router or using parallel links. For example, if we have routers with 8 ports, we could attach 4 processing nodes and configure a 2-dimensional network, using 2 links for each

dimension, or attach 2 processing nodes and configure a 3-dimensional network, using 2 links for each dimension too (see Figure 13.1).

On the other hand, this thesis presents new routing algorithms to avoid the HoL-blocking effect in direct topologies. These algorithms are able to work in KNS topologies. However, they are not specifically developed for these topologies. So, it would be interesting to study its behavior in these topologies in more detail.

In addition, we can join the proposed HoL-blocking reduction routing algorithms with injection control techniques. Some related work has been recently published focusing on this technique. These mechanisms help to reduce the traffic flows which induce to generate congestion. Therefore, if we design a new injection control technique and we use it together to our HoL-blocking reduction routing algorithms, we can obtain better results.

13.2 Other Publications

In addiction to the papers shown above, other related papers have been published in domestic conferences:

- R. Peñaranda, C. Gómez, M.E. Gómez, P. López, and J. Duato. KNS: Familia de Topologías Hbridas para la Interconexión de Redes de Gran Escala. In *Actas de las XXIV Jornadas de Paralelismo* (JP), pages 109-114, Madrid, Spain, 2013.
- R. Peñaranda, C. Gómez, M.E. Gómez, P. López, and J. Duato. Deterministic versus adaptive routing in direct topologies. In *Actas de las XXIV Jornadas de Paralelismo* (JP), pages 115-120, Madrid, Spain, 2013.
- R. Peñaranda, C. Gómez, M.E. Gómez, P. López, and J. Duato. On the Reduction of HoL-blocking in Direct Topologies with Deterministic Routing. In *Actas de las XXV Jornadas de Paralelismo* (JP), pages 423-432, Valladolid, Spain, 2014.
- R. Peñaranda, C. Gómez, M.E. Gómez and P. López. Reducción del HoL-blocking con encaminamiento adaptativo. In *Actas de las XXVI Jornadas de Paralelismo* (JP), pages 48-52, Córdoba, Spain, 2015.
- R. Peñaranda, E. Gunnar Gran, T. Skeie, M.E. Gómez and P. López. Una nueva metodología para encaminamiento tolerante a fallos en topologías KNS. In *Actas de las XXVII Jornadas de Paralelismo* (JP), pages 335-343, Salamanca, Spain, 2016.

13.3 Funding Acknowledgments

This thesis was partially supported by:

- the Spanish Ministerio de Economía y Competitividad (MINECO), by FEDER funds under Grant TIN2009-14475-C04-01.
- the Spanish Ministerio de Economía y Competitividad (MINECO), by FEDER funds under Grant TIN2012-38341-C04-01.
- the Spanish Ministerio de Economía y Competitividad (MINECO), by FEDER funds under Grant TIN2015-66972-C5-1-R.
- the Programa de Ayudas de Investigación y Desarrollo (PAID), from Universitat Politècnica de València.
- the financial support of the FP7 HiPEAC Network of Excellence, under grant agreement 287759.
- the Ayudas para Primeros Proyectos de Investigación from Universitat Politècnica de València, under grant ref. 2370.

All works listed above are exclusively related with this thesis. The specific contributions of the Ph.D. candidate reside mostly in the implementation of the proposed techniques, the setup and execution of most simulation experiments, the writing of the paper drafts describing the work as well as the presentation in the conferences. Along these processes, the co-authors have repeatedly provided useful hints and advice, which the Ph.D. candidate has then applied to make the work evolve into its final version.

References

[1] TOP500 Supercomputer Site. `http://www.top500.org` Accessed 3 Feb 2016.

[2] W.J. Dally and B. Towles. *Principles and practices of interconnection networks*. Morgan Kaufmann, 2004.

[3] J. Duato, S. Yalamanchili, and N. Lionel. *Interconnection Networks: An Engineering Approach*. Morgan Kaufmann Publishers Inc., USA, 2002. ISBN 1558608524.

[4] Charles E. Leiserson. Fat-trees: universal networks for hardware-efficient supercomputing. *IEEE Trans. Comput.*, 34(10):892–901, October 1985. ISSN 0018-9340.

[5] C. Gómez, F. Gilabert, M.E. Gómez, P. López, and J. Duato. RUFT: Simplifying the Fat-Tree Topology. In *Parallel and Distributed Systems, 2008. ICPADS '08. 14th IEEE International Conference on*, pages 153–160, dec. 2008. doi: 10.1109/ICPADS.2008.44.

[6] D Bermudez Garzon, Maria Eugenia Gomez, Pierre Lopez, Jose Duato, and Christopher Gomez. FT-RUFT: A Performance and Fault-Tolerant Efficient Indirect Topology. In *Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on*, pages 405–409. IEEE, 2014.

[7] Mellanox Technology. `http://www.mellanox.com`.

[8] Myricom. `http://www.myri.com`.

[9] G. Della Vecchia and C. Sanges. Recursively Scalable Networks for Message Passing Architectures. *Proceedings of International Conference on Parallel Processing and Applications*, pages 33–10, 1987.

[10] D. Rahmati, A.E. Kiasari, S. Hessabi, and H. Sarbazi-Azad. A Performance and Power Analysis of WK-Recursive and Mesh Networks for Network-on-Chips. In *Computer Design, 2006. ICCD 2006. International Conference on*, pages 142–147, oct. 2006. doi: 10.1109/ICCD.2006.4380807.

[11] J. Kim, W.J. Dally, and D. Abts. Flattened butterfly: a cost-efficient topology for high-radix networks. In *Proceedings of the 34th annual international symposium on Computer architecture*, ISCA '07, pages 126–137, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-706-3. doi: 10.1145/1250662.1250679.

[12] R. Das, S. Eachempati, A.K. Mishra, V. Narayanan, and C.R. Das. Design and evaluation of a hierarchical on-chip interconnect for next-generation CMPs. In *High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on*, pages 175–186, feb. 2009. doi: 10.1109/HPCA.2009.4798252.

[13] A.K. Gupta and W.J. Dally. Topology optimization of interconnection networks. *Computer Architecture Letters*, 5(1):10–13, jan.-june 2006. ISSN 1556-6056. doi: 10.1109/L-CA.2006.8.

[14] J. Kim, W.J. Dally, S. Scott, and D. Abts. Technology-Driven, Highly-Scalable Dragonfly Topology. In *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ISCA '08, pages 77–88, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-0-7695-3174-8. doi: 10.1109/ISCA.2008.19.

[15] Marina García, Enrique Vallejo 0001, Ramón Beivide, Cristobal Camarero, Mateo Valero, Germán Rodríguez, and Cyriel Minkenberg. On-the-fly adaptive routing for dragonfly interconnection networks. *The Journal of Supercomputing*, 71(3):1116–1142, 2015.

[16] Yulu Yang, A. Funahashi, A. Jouraku, H. Nishi, H. Amano, and T. Sueyoshi. Recursive diagonal torus: an interconnection network for massively parallel computers. *Parallel and Distributed Systems, IEEE Transactions on*, 12(7):701–715, jul 2001. ISSN 1045-9219. doi: 10.1109/71.940745.

[17] H. Matsutani, M. Koibuchi, and H. Amano. Performance, Cost, and Energy Evaluation of Fat H-Tree: A Cost-Efficient Tree-Based On-Chip Network. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–10, march 2007. doi: 10.1109/IPDPS.2007.370271.

[18] F.T. Leighton. *Introduction to parallel algorithms and architectures: arrays, trees, hypercubes*. Number v. 1. M. Kaufmann Publishers, 1992. ISBN 9781558601178.

[19] A.O. Balkan, Gang Qu, and U. Vishkin. Mesh-of-Trees and Alternative Interconnection Networks for Single-Chip Parallelism. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 17(10):1419–1432, oct. 2009. ISSN 1063-8210. doi: 10.1109/TVLSI.2008.2003999.

[20] Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu. BCube: a high performance, server-centric network architecture for modular data centers. In *SIGCOMM '09: Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, pages 63–74, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-594-9. doi: 10.1145/1592568.1592577. URL `http://www.bibsonomy.org/bibtex/23a5da89fbf099e3c70f4559ab38082c5/chesteve`.

[21] Taisuke Boku, Kisaburo Nakazawa, Hiroshi Nakamura, Takeshi Sone, Takeshi Mishima, and Ken'ichi Itakura. Adaptive routing technique on hypercrossbar network and its evaluation. *Systems and Computers in Japan*, 27(4):55–64, 1996.

[22] C. Gómez, F. Gilabert, M.E. Gómez, P. López, and J. Duato. Deterministic versus Adaptive Routing in Fat-Trees. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–8, march 2007. doi: 10.1109/IPDPS.2007.370482.

[23] P. Carvey L. Dennison W.J. Dally. Architecture of the avici terabit switch/router. *in:Proceedings of Hot Interconnects*, 6, August 1998.

[24] Thomas E. Anderson, Susan S. Owicki, James B. Saxe, and Charles P. Thacker. High-speed switch scheduling for local-area networks. *ACM Trans. Comput. Syst.*, 11(4):319–352, 1993. ISSN 0734-2071. doi: 10.1145/161541.161736.

[25] T. Nachiondo, J. Flich, and J. Duato. Buffer Management Strategies to Reduce HoL Blocking. *IEEE Trans. on Paral. and Distributed Systems*, 21:739–753, 2010. ISSN 1045-9219. doi: 10.1109/TPDS.2009.63.

[26] Pedro Yebenes, Jesus Escudero-Sahuquillo, Crispin Gomez, Pedro Javier Garcia, Francisco J Quiles, and Jose Duato. BBQ: a straightforward queuing scheme to reduce hol-blocking in high-performance hybrid networks. In *Euro-Par 2013*, pages 699–712. Springer, 2013.

[27] Ruben Casado, Aurelio Bermúdez, Jose Duato, Francisco J Quiles, and José L Sánchez. A protocol for deadlock-free dynamic reconfiguration in high-speed local area networks. *Parallel and Distributed Systems, IEEE Transactions on*, 12(2):115–132, 2001.

[28] Timothy Mark Pinkston, Ruoming Pang, and José Duato. Deadlock-free dynamic reconfiguration schemes for increased network dependability. *Parallel and Distributed Systems, IEEE Transactions on*, 14(8):780–794, 2003. URL `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1225057`.

[29] Olav Lysne, Timothy Mark Pinkston, and Jose Duato. A methodology for developing dynamic network reconfiguration processes. In *Parallel Processing, 2003. Proceedings. 2003 International Conference on*, pages 77–86. IEEE, 2003. URL `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1240568`.

[30] Olav Lysne, José Miguel Montañana, Timothy Mark Pinkston, José Duato, Tor Skeie, and José Flich. Simple deadlock-free dynamic network reconfiguration. In *High Performance Computing-HiPC 2004*, pages 504–515. Springer, 2005. URL `http://link.springer.com/chapter/10.1007/978-3-540-30474-6_53;http://ceng.usc.edu/smart/people/publications/archives/HiPC04tpink.pdf`.

[31] Valentin Puente, José A. Gregorio, Fernando Vallejo, and Ramón Beivide. Immunet: A Cheap and Robust Fault-Tolerant Packet Routing Mechanism. In *ISCA*, pages 198–211. IEEE Computer Society, 2004. ISBN 0-7695-2143-6. URL `http://dblp.uni-trier.de/db/conf/isca/isca2004.html#PuenteGVB04;http://dl.acm.org/citation.cfm?id=1006718;http://www.bibsonomy.org/bibtex/2c62c803fc6736322fb95f595c76502fa/dblp`.

[32] Christopher J. Glass and Lionel M. Ni. Fault-Tolerant Wormhole Routing in Meshes without Virtual Channels. *IEEE Trans. Parallel Distrib. Syst.*, 7(6):620–636, 1996.

[33] Olav Lysne, Tor Skeie, and Thomas Waadeland. One-fault tolerance arid beyond in wormhole routed meshes. *Microprocessors and Microsystems*, 21(7):471–480, 1998.

[34] Nils Agne Nordbotten and Tor Skeie. A routing methodology for dynamic fault tolerance in meshes and tori. In *High Performance Computing–HiPC 2007*, pages 514–527. Springer, 2007.

[35] William J. Dally and Hiromichi Aoki. Deadlock-Free Adaptive Routing in Multicomputer Networks Using Virtual Channels. *IEEE Trans. Parallel Distrib. Syst.*, 4(4):466–475, 1993.

[36] Daniel H. Linder and James C. Harden. An Adaptive and Fault Tolerant Wormhole Routing Strategy for k-Ary n-Cubes. *IEEE Trans. Computers*, 40(1):2–12, 1991.

[37] Rajendra V Boppana and Suresh Chalasani. Fault-tolerant wormhole routing algorithms for mesh networks. *Computers, IEEE Transactions on*, 44(7):848–864, 1995.

[38] Andrew A. Chien and Jae H. Kim. Planar-Adaptive Routing: Low-cost Adaptive Networks for Multiprocessors. In Allan Gottlieb, editor, *ISCA*, pages 268–277. ACM, 1992. ISBN 0-89791-509-7.

[39] Suresh Chalasani and Rajendra V Boppana. Communication in multicomputers with non-convex faults. *Computers, IEEE Transactions on*, 46(5):616–622, 1997.

[40] Chun-Lung Chen and Ge-Ming Chiu. A Fault-Tolerant Routing Scheme for Meshes with Nonconvex Faults. *IEEE Trans. Parallel Distrib. Syst.*, 12(5):467–475, 2001.

[41] Suresh Chalasani and Rajendra V Boppana. Fault-tolerant wormhole routing in tori. In *Proceedings of the 8th International Conference on Supercomputing*, pages 146–155. ACM, 1994.

[42] Chris M. Cunningham and Dimiter R. Avresky. Fault-Tolerant Adaptive Routing for Two-Dimensional Meshes. In *HPCA*, pages 122–131. IEEE Computer Society, 1995. ISBN 0-8186-6445-2.

[43] Christopher J Glass and Lionel M Ni. The turn model for adaptive routing. In *ACM SIGARCH Computer Architecture News*, volume 20, pages 278–287. ACM, 1992.

[44] José Duato. A Thory of Fault-Tolerant routing in Wormhole Networks. In Lionel M. Ni, editor, *ICPADS*, pages 600–607. IEEE Computer Society, 1994. ISBN 0-8186-6555-6.

[45] Leslie G. Valiant. A Scheme for Fast Parallel Communication. *SIAM J. Comput.*, 11(2): 350–361, 1982.

[46] María Engracia Gómez, José Duato, Jose Flich, Pedro López, Antonio Robles, Nils Agne Nordbotten, Olav Lysne, and Tor Skeie. An Efficient Fault-Tolerant Routing Methodology for Meshes and Tori. *Computer Architecture Letters*, 3, 2004.

[47] Connect-IB. http://www.mellanox.com/related-docs/prod_adapter_cards/PB_Connect-IB.pdf.

[48] Quadrics homepage. http://www.quadrics.com.

[49] H. Litz, H. Froning, M. Nuessle, and U. Bruning. HyperTransport NIC for Ultra-low Latency Message Transfer. feb 2008.

[50] S. Scott, D. Abts, J. Kim, and W.J. Dally. The BlackWidow High-Radix Clos Network. *SIGARCH Comput. Archit. News*, 34(2):16–28, May 2006. ISSN 0163-5964. doi: 10.1145/1150019.1136488.

[51] N. Binkert, Al Davis, N.P. Jouppi, M. McLaren, N. Muralimanohar, R. Schreiber, and Jung Ho Ahn. The role of optics in future high radix switch design. *SIGARCH Comput. Archit. News*, 39(3):437–448, June 2011. ISSN 0163-5964. doi: 10.1145/2024723.2000116.

[52] J. Flich, M.P. Malumbres, P. López, and J. Duato. Improving Routing Performance in Myrinet Networks. *Parallel and Distributed Processing Symposium, International*, page 27, 2000. ISSN 1530-2075. doi: 10.1109/IPDPS.2000.845961.

[53] J. C. Martínez, J. Flich, A. Robles, P. López, and J. Duato. In-Order Packet Delivery in Interconnection Networks using Adaptive Routing. In *IEEE International Parallel and Distributed Processing Symp*, 2005.

[54] W.J. Dally and C.L. Seitz. Deadlock-Free Message Routing in Multiprocessor Interconnection Networks. *Computers, IEEE Transactions on*, C-36(5):547–553, may 1987. ISSN 0018-9340. doi: 10.1109/TC.1987.1676939.

[55] V. Puente, R. Beivide, J.A. Gregorio, J.M. Prellezo, J. Duato, and C. Izu. Adaptive bubble router: a design to improve performance in torus networks. In *Parallel Processing, 1999. Proceedings. 1999 International Conference on*, pages 58–67, 1999. doi: 10.1109/ICPP.1999.797388.

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

[57] A.A. Chien. A cost and speed model for k-ary n-cube wormhole routers. *Parallel and Distributed Systems, IEEE Transactions on*, 9(2):150–162, feb 1998. ISSN 1045-9219. doi: 10.1109/71.663877.

[58] J. Duato and P. López. Performance evaluation of adaptive routing algorithms for k-ary n-cubes. In Kevin Bolding and Lawrence Snyder, editors, *Parallel Computer Routing and Communication*, volume 853 of *Lecture Notes in Computer Science*, pages 45–59. Springer Berlin, Heidelberg, 1994. ISBN 978-3-540-58429-2.

[59] Xuan-Yi Lin, Yeh-Ching Chung, and Tai-Yi Huang. A Multiple LID Routing Scheme for Fat-Tree-Based InfiniBand Networks. In *IPDPS*, 2004.

[60] M. Karol, M. Hluchyj, and S. Morgan. Input vs. Output Queueing on a Space-Division Packet Switch. *Communications, IEEE Trans. on*, 35(12):1347–1356, 1987. ISSN 0090-6778. doi: 10.1109/TCOM.1987.1096719.

[61] J.C.R. Bennett, C. Partridge, and N. Shectman. Packet reordering is not pathological network behavior. *Networking, IEEE/ACM Trans. on*, 7(6):789–798, dec 1999. ISSN 1063-6692. doi: 10.1109/90.811445.

[62] N. McKeown, V. Anantharam, and J. Walrand. Achieving 100% throughput in an input-queued switch. In *INFOCOM '96. Fifteenth Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation. Proceedings IEEE*, volume 1, pages 296–302 vol.1, mar 1996. doi: 10.1109/INFCOM.1996.497906.

[63] W.J. Dally. Virtual-channel flow control. *Parallel and Distributed Systems, IEEE Transactions on*, 3(2):194–205, mar 1992. ISSN 1045-9219. doi: 10.1109/71.127260.

[64] Roberto Peñaranda, Crispín Gómez, María Engracia Gómez, Pedro López, and José Duato. IODET: A HoL-blocking-aware Deterministic Routing Algorithm for Direct Topologies. In *ICCS*, pages 702–703. IEEE Computer Society, 2012. ISBN 978-1-4673-4565-1.

[65] María Engracia Gómez, José Duato, Jose Flich, Pedro López, Antonio Robles, Nils Agne Nordbotten, Tor Skeie, and Olav Lysne. A New Adaptive Fault-Tolerant Routing Methodology for Direct Networks. In Luc Bougé and Viktor K. Prasanna, editors, *HiPC*, volume 3296 of *Lecture Notes in Computer Science*, pages 462–473. Springer, 2004. ISBN 3-540-24129-9.

[66] Roberto Peñaranda, Crispín Gómez, María Engracia Gómez, Pedro López, and José Duato. HoL-blocking Avoidance Routing Algorithms in Direct Topologies. In *HPCC*, pages 11–18. IEEE Computer Society, 2014. ISBN 978-1-4799-6123-8.

[67] Roberto Peñaranda, Crispín Gómez Requena, María Engracia Gómez, Pedro López, and José Duato. A New Family of Hybrid Topologies for Large-Scale Interconnection Networks. In *NCA*, pages 220–227. IEEE Computer Society, 2012. ISBN 978-1-4673-2214-0.

[68] Mellanox Store. http://www.mellanoxstore.com.

[69] Brian Towles and William J. Dally. Worst-case traffic for oblivious routing functions. In *Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures*, SPAA '02, pages 1–8, New York, NY, USA, 2002. ACM. ISBN 1-58113-529-7. doi: 10.1145/564870.564872. URL http://doi.acm.org/10.1145/564870.564872.

[70] Suchendra M. Bhandarkar and Hamid R. Arabnia. The Hough Transform on a Reconfigurable Multi-Ring Network. *J. Parallel Distrib. Comput.*, 24(1):107–114, 1995.

[71] Roberto Peñaranda, Ernst Gunnar Gran, Tor Skeie, María Engracia Gómez, and Pedro López. A New Fault-Tolerant Routing Methodology for KNS Topologies. In *2016 2nd IEEE International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB)*, pages 1–8. IEEE, 2016. doi: 10.1109/HIPINEB.2016.9. URL http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7457761.

[72] Jose Flich, Tor Skeie, Andres Mejia, Olav Lysne, Pedro Lopez, Antonio Robles, Jose Duato, Michihiro Koibuchi, Tomas Rokicki, and Jose Carlos Sancho. A Survey and Evaluation of Topology-Agnostic Deterministic Routing Algorithms. *IEEE Transactions on Parallel and Distributed Systems*, 23(3):405–425, 2012. ISSN 1045-9219. doi: 10.1109/TPDS.2011.190. URL http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5953590.

[73] Chien-chun Su and Kang G. Shin. Adaptive Fault-Tolerant Deadlock-Free Routing in Meshes and Hypercubes. *IEEE Transactions on Computers*, 45:672–683, 1995.

[74] Mithuna Thottethodi, Alvin R. Lebeck, and Shubhendu S. Mukherjee. BLAM: A High-Performance Routing Algorithm for Virtual Cut-Through Networks. In *Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, IPDPS '03, pages 45.2–, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-1926-1. URL http://dl.acm.org/citation.cfm?id=838237.838513.

[75] Andrew Chien. A Cost and Speed Model for k-ary n-cube Wormhole Routers. In *Hot Interconnects '93*, 1993.

[76] Li-Shiuan Peh and William J. Dally. A Delay Model and Speculative Architecture for Pipelined Routers. In *Proceedings of the 7th International Symposium on High-Performance Computer Architecture*, HPCA '01, pages 255–, Washington, DC, USA, 2001. IEEE Computer Society. ISBN 0-7695-1019-1. URL http://dl.acm.org/citation.cfm?id=580550.876446.

[77] Roberto Peñaranda, Crispín Gómez Requena, María Engracia Gómez, and Pedro López. XORAdap: A HoL-Blocking Aware Adaptive Routing Algorithm. In Masoud Daneshtalab, Marco Aldinucci, Ville Leppänen, Johan Lilius, and Mats Brorsson, editors, *PDP*, pages 48–52. IEEE Computer Society, 2015. ISBN 978-1-4799-8491-6. URL http://dblp.uni-trier.de/db/conf/pdp/pdp2015.html#PenarandaRGL15;http://doi.ieeecomputersociety.org/10.1109/PDP.2015.50;http://www.bibsonomy.org/bibtex/2b8839a14d1a16c9734cb94d893feb51d/dblp.

[78] Carmen Carrión, Ramón Beivide, José-Ángel Gregorio, and Fernando Vallejo. A flow control mechanism to avoid message deadlock in k-ary n-cube networks. In *HiPC*, pages 322–329. IEEE Computer Society, 1997. ISBN 0-8186-8067-9. URL http://dblp.uni-trier.de/db/conf/hipc/hipc1997.html#CarrionBGV97;http://doi.ieeecomputersociety.org/10.1109/HIPC.1997.634510;http://www.bibsonomy.org/bibtex/22f7390b3349bc42bab7a64234feaf9a5/dblp.

[79] Luis Gravano, Gustavo D. Pifarré, Pablo E. Berman, and Jorge L. C. Sanz. Adaptive Deadlock- and Livelock-Free Routing with All Minimal Paths in Torus Networks. *IEEE Trans. Parallel Distrib. Syst.*, 5(12):1233–1251, 1994. URL http://dblp.uni-trier.de/db/journals/tpds/tpds5.html#GravanoPBS94;http://doi.ieeecomputersociety.org/10.1109/71.334898;http://www.bibsonomy.org/bibtex/2806d2f295c635a86b3e788c3019e9b2e/dblp.

[80] Keith D. Underwood and Eric Borch. A Unified Algorithm for Both Randomized Deterministic and Adaptive Routing in Torus Networks. In *IPDPS Workshops*, pages 723–732. IEEE, 2011. ISBN 978-1-61284-425-1. URL http://dblp.uni-trier.de/db/conf/ipps/ipdps2011w.html#UnderwoodB11;http://doi.

ieeecomputersociety.org/10.1109/IPDPS.2011.214;http://www.bibsonomy.org/bibtex/292473ee8b95cc7c2a8526816c237db52/dblp.

[81] Arjun Singh, William J. Dally, Amit K. Gupta, and Brian Towles. GOAL: A Load-Balanced Adaptive Routing Algorithm for Torus Networks. In Allan Gottlieb and Kai Li, editors, *ISCA*, pages 194–205. IEEE Computer Society, 2003. ISBN 0-7695-1945-8. URL http://dblp.uni-trier.de/db/conf/isca/isca2003.html#SinghDGT03;http://doi.acm.org/10.1145/859618.859641;http://www.bibsonomy.org/bibtex/28a8bbcf9d0a4b3d4a1b053b59130ec43/dblp.

[82] Leslie G. Valiant and Gordon J. Brebner. Universal Schemes for Parallel Communication. In *STOC*, pages 263–277. ACM, 1981. URL http://dblp.uni-trier.de/db/conf/stoc/stoc81.html#ValiantB81;http://doi.acm.org/10.1145/800076.802479;http://www.bibsonomy.org/bibtex/262620662b32e54ddb1c3db1bfd8d4954/dblp.

[83] José Duato, Jose Flich, and Teresa Nachiondo Frinós. A Cost-Effective Technique to Reduce HOL Blocking in Single-Stage and Multistage Switch Fabrics. In *PDP*, pages 48–53. IEEE Computer Society, 2004. ISBN 0-7695-2083-9. URL http://dblp.uni-trier.de/db/conf/pdp/pdp2004.html#DuatoFN04;http://doi.ieeecomputersociety.org/10.1109/EMPDP.2004.1271426;http://www.bibsonomy.org/bibtex/26f79cc4f262b291644f218ce7b5e66eb/dblp.

[84] Tor Skeie, Olav Lysne, and Ingebjørg Theiss. Layered Shortest Path (LASH) Routing in Irregular System Area Networks. In *16th International Parallel and Distributed Processing Symposium (IPDPS 2002), 15-19 April 2002, Fort Lauderdale, FL, USA, CD-ROM/Abstracts Proceedings*, 2002. doi: 10.1109/IPDPS.2002.1016559. URL http://dx.doi.org/10.1109/IPDPS.2002.1016559.

[85] P.Duato J. López. Deadlock-Free Adaptive Routing Algorithms for the 3D-Torus: Limitations and Solutions. In Arndt Bode, Mike Reeve, and Gottfried Wolf, editors, *PARLE'93, Parallel Architectures and Languages Europe*, volume 694 of *Lecture Notes in Computer Science*, pages 684–687. Springer Berlin, Heidelberg, 1993. ISBN 3-540-56891-3.

[86] Torsten Hoefler, Timo Schneider, and Andrew Lumsdaine. Multistage switches are not crossbars: Effects of static routing in high-performance networks. In *CLUSTER*, pages 116–125. IEEE Computer Society, 2008. ISBN 978-1-4244-2640-9. URL http://dblp.uni-trier.de/db/conf/cluster/cluster2008.html#HoeflerSL08;http://doi.ieeecomputersociety.org/10.1109/CLUSTR.2008.4663762;http://www.bibsonomy.org/bibtex/2265214a6500976bbc7b96062ee81fbdb/dblp.