COST-EFFECTIVE FAULT TOLERANT ROUTING IN NETWORKS ON CHIP

VENERA ADANOVA

SEPTEMBER 2008

COST-EFFECTIVE FAULT TOLERANT ROUTING IN NETWORKS ON CHIP

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

VENERA ADANOVA

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

SEPTEMBER 2008

Approval of the thesis

COST-EFFECTIVE FAULT TOLERANT ROUTING IN NETWORKS ON CHIP

submitted by **Venera Adanova** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Science** _____

Prof. Dr. Volkan Atalay
Head of Department, **Computer Engineering** _____

Assoc.Prof. Dr. Ali Hikmet Dogru
Supervisor, **Computer Engineering Department** _____

Examining Committee Members

Prof.Dr.Müslim Bozyigit
Computer Engineering Dept., METU _____

Assoc.Prof. Dr. Ali Hikmet Doğru
Computer Engineering Dept., METU _____

Prof.Dr.Adnan Yazıcı
Computer Engineering Dept., METU _____

Assoc.Prof.Dr.Ahmet Cosar
Computer Engineering Dept., METU _____

Dr.Sevgi Özkan
Graduate School of Informatics, METU _____

Date:

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name: Venera Adanova

Signature :

ABSTRACT

COST-EFFECTIVE FAULT TOLERANT ROUTING IN NETWORKS ON CHIPS

Adanova, Venera
M.Sc., Department of Computer Engineering
Supervisor: Assoc.Prof. Dr. Ali Hikmet Doğru

September 2008, 87 pages

Growing complexity of Systems on Chip (SoC) introduces interconnection problems. As a solution for communication bottleneck the new paradigm, Networks on Chip (NoC), has been proposed. Along with high performance and reliability, NoC brings in area and energy constraints. In this thesis we mainly concentrate on keeping communication-centric design environment fault-tolerant while considering area overhead. The previous researches suggest the adoption solution for fault-tolerance from multiprocessor architectures. However, multiprocessor architectures have excessive reliance on buffering leading to costly solutions. We propose to reconsider general router model by introducing central buffers which reduces buffer size. Besides, we offer a new fault-tolerant routing algorithm which effectively utilizes buffers at hand without additional buffers out of detriment to performance.

Keywords: Networks on Chip, Fault-Tolerant Routing, Wormhole Routing

ÖZ

YONGALARDA MALİYETİ UYGUN HATAYA DAYANIKLI YÖNLENDİRME AĞI

Adanova, Venera
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Doç. Dr. Ali Hikmet Doğru

Eylül 2008, 87 sayfa

Yonga Üstü Sistem'in (YÜS) artan karmaşıklığı bağlantı sorunlarına yol açmaktadır.İletişimdeki bu darboğaza çözüm olarak yeni bir model olan Yonga Üstü İletişim Ağ (YÜİA) ileri sürülmüştür. Yüksek performans ve güvenilirliği yanında, YÜİA alan ve enerji kısıtlaması kazandırmaktadır. Bu tezde esas olarak, alanı sabit tutmayı gözetirken, iletişim merkezli tasarım çevresini hataya dayanıklı olarak tutmaya odaklandık. Önceki araştırmalar, çokişlemci mimarilerden hataya dayanıklı çözümler uyarlamayı öneriyorlar. Fakat, çoğu zaman çok işlemci mimariler pahalı çözümlere yol açan tampon belleklemeye dayanırlar. Tampon bellek boyutunu küçülten merkezi tampon bellekleri tanıtarak genel yönlendirici modeli yeniden ele almayı öneriyoruz. Bunun yanında, performansı düşürmeyen ek tampon belleksiz, eldeki tampon belleklerden etkili biçimde yararlanan hataya dayanıklı yeni bir yönlendirici algoritma sunuyoruz.

Anahtar Kelimeler: Yonga Üstü İletişim Ağı, Hataya Dayanıklı Yönlendirme, Solucan Yönlendirme

Апама арналат

Dedicated to my mother

ACKNOWLEDGEMENTS

I am deeply grateful to my thesis supervisor, Assoc. Prof. Dr. Ali Hikmet Dogru, for his guidance, support and encouragement throughout my thesis work.

I would like to thank my friend Rita Ismailova for her invaluable support and assistance both in my thesis work and in my life.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURES

LIST OF TABLES

TABLES

LIST OF ABBREVIATIONS

2D	Two Dimensional
BFT	Butterfly Fat Tree
CLICHÉ	Chip-Level Integration of Communicating Heterogeneous Elements
DIM	Dimension
DOR	Dimension Ordered Routing
DSP	Digital Signal Processor
EW	East-West
f-chain	Faulty chain
FPGA	Field-Programmable Gate Array
f-ring	Faulty ring
IC	Input Controller
IP	Intellectual Property
LAN	Local Area Network
MIN	Multistage Interconnection Networks
MTBF	Mean Time Between Failures
MTTR	Mean Time To Repair
NAM	Network AniMator
NoC	Network on Chip
NS	North-South
OTcl	Object Tool command line
PE	Processing Element
ROMM	Randomized, Oblivious, Multi-Phase, Minimal
SAF	Store-and-Forward
SB	Shared Buffer

SN	South-North
SoC	System on Chip
SPIN	Scalable Programmable Integrated Network
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VC	Virtual Channel
VCID	Virtual Channel Identifier
VCT	Virtual Cut-Through
WAN	Wide Area Network
WE	West-East

CHAPTER 1

INTRODUCTION

The rapid advance in semiconductor technology has opened new advantages which allow placing many resources into single chip. This approach is known as Systems on Chip (SoC). It is predicted [1] [2] that further improvements in silicon technology will allow tens or even hundreds of processing elements (PEs) to fit into single chip. A PE can be a processor core, a DSP core, an FPGA block, memory block etc. However, as the number of PEs in SoC increases, complexity of interconnection architecture of the SoC also increases [3]. The up to now solutions that are used as communication infrastructures (point-to-point, shared-medium architectures) considered to be not enough for data exchange among PEs because of growing system size and non-scalable wire delay [4]. Although point-to-point communication maximizes chip performance, its design complexity takes high development cost [3]. Widely used shared communication architectures have limited scalability and poor performance when the number of PEs becomes more than ten. Besides, general buses have high energy consumption because of its broadcast data transfer nature. This means that data need great energy in order to reach each possible receiver [5].

Because of limitations introduced by communication infrastructure, the on chip design methodology became communication-centric rather than computation-centric [6]. This led to a new design methodology called Network on Chip (NoC). NoC efficiently decouples communication infrastructure from computation. Decoupling gives more structured architecture and modularity. The performance

increases due to increase in bandwidth and support of concurrent communication [4].

Models and techniques used in NoC are generally borrowed from computer networks and parallel processing. Scalability of computer networks and its steady success over many years motivate researchers to adopt switch-based (router) networks and packet-based communication. NoC provides PEs with communication infrastructure, where PEs are connected to the network via intelligent switches (routers). The communication is done by message-passing between PEs. The message may take multiple hops through intermediate routers in order to reach its destination. Thus, PEs are independent from each other and do not require global arbiter to access the network enabling highly utilized of network bandwidth.

NoC is much like interconnection networks for high performance parallel computers in which each processor is an individual chip [3]. However, NoC have a number of characteristics that make their design quit different than the inter-chip (inter-board) networks. Firstly, wires are more abundant [7]. Second, they have energy and area constraints. This limits the buffer usage since they significantly impact the area overhead.

The design of NoC requires the consideration over network topology, switching technique and routing algorithm to be used [8].

The right topology selection is important since it defines the ability of the network to efficiently disseminate information. Its importance extends when considering the network latency, throughput, area, fault-tolerance, power consumption, and designing the routing strategy and mapping cores to the network nodes. It is desirable to have rich set of predefined topologies that can be reused. This significantly reduces the time required for system design [9] [10].

While the topology defines only the static aspects of network-based communication, routing algorithms and switching technique governs the actual movement of messages along the network. Routing algorithm greatly influences the network performance and power consumption, since more complicated

algorithms require larger design. The switching technique defines when the routing decision is made and how the packets are transferred between switches. It introduces trade-offs between area overhead that results from larger buffer sizes and performance [11].

Along with the described requirements, it is important to keep communication-centric design environment fault-tolerant and reliable. It must at least guarantee that the particular fault will not cause the entire chip to fail. The solutions for fault-tolerant can be adopted from multiprocessor architectures. However, they should be evaluated in terms of throughput, latency, area overhead, and energy dissipation [4].

Our research is motivated mainly by the possible prospects of NoC, since it is considered to be a solution for communication bottleneck. NoC is still in its infancy. Despite lots of researches conducted in this area, few problems have been faced and still more left for the future. We mainly focus on fault-tolerance, since even less researches were conducted on this area. It is not of less importance to keep system working under various fault patterns. Possibly with degraded performance. Fault-tolerance requires redundancy in terms of buffers. Since faulty components introduce additional dependencies between channels while routing messages around faulty regions, the number of required virtual channels increases. However, NoC constraints the area used by buffers. Thus, it is desirable to have fault-tolerant algorithms with less buffer requirements. This situation again leads to trade-offs between area and performance.

Our main concern is to decrease the number of used virtual channels while tolerating faults. Since NoC restricts buffer size it is important to be able to take advantage of buffers that are in hand rather than using additional buffers. This requires dynamic allocation of these buffers that are idle. Besides, we make some changes to router organization, i.e. to the location of buffers.

Chapter 2 gives an idea about the main aspects of NoC. Proposed traffic models are also described briefly in this chapter.

Chapter 3 describes the general-purpose simulator, ns-2, and our router model. The various simulation results on parameterizable aspects of NoC are given and analyzed.

Chapter 4 deals with our fault-tolerant algorithm and compares the simulation results with fCube4 algorithm.

Conclusion and future work is presented in Chapter 5.

CHAPTER 2

BACKGROUND

NoC consists of routers, buffers and links. While considering the design of a system it is important to choose the right topology, so that the applications match onto topology in such a manner that the performance will be the best. Since buffer usage is limited the right switching technique that requires the limited buffers size must be chosen. It depends on application if the messages are sent packet by packet or divided into smaller units called *flits*. The right buffer utilization has big impact on the system performance. The complexity of a router is determined by the routing algorithm. Although deterministic algorithms are simple and fast, they do not have the flexibility of adaptive algorithms.

This chapter describes the proposed topologies, switching techniques and routing algorithms. The differences between them and their comparisons are presented. We also give a general view of faults and how the literature proposes to handle them.

2.1 Network Topology

The topology of a network is a graph that defines how the nodes are interconnected by channels. In general, NoC adopts the interconnection architecture of high performance parallel computing, where Intellectual Property (IP) blocks are connected to the network via routers, thus providing a communication between functional IP blocks. However, SoC design paradigm introduces some constraints, as compared to high performance parallel computing. Along with the desirable characteristics, such as high throughput and low latency, SoC also considers energy consumption and the area overhead of proposed interconnection architecture. It is also desirable for interconnection architecture to be scalable, making the design of SoC even more difficult since the latency of a message should still be kept low as the number of IP blocks increase [12]. Since, the system performance is highly dependent on communication, the right topology selection is important.

[13] argues that the topology should be application specific since the system will consist of heterogeneous nodes. However, predefined, regular topologies are generally suggested [14] [1] [15], since they reduce the system design time.

Below we present some of the proposed architectures. Here, functional IP blocks are shown as white squares and the routers are shown as black squares.

2.1.1 Octagon

The octagon architecture was proposed by Karim et al. [16] for network processor SoCs. It is a special case of more general class of networks called Spidergon [17]. In this architecture each node is associated with a processing element (PE or otherwise IP) and a switch. The basic octagon consists of eight nodes and 12 bidirectional links (Fig.1), and the communication between any two nodes takes at most two hops within it. As the number of nodes increase the octagon is extended to a multidimensional space.

Unlike the traditional shared buses and crossabars [18] the octagon architecture has much higher throughput, is easy in implementation, and has less wiring complexity. It implements simple shortest path algorithm. However, as the number of nodes increase the implementation complexity also increases. It also requires an implementation of network arbiter and a good scheduler, since nodes share the single communication point at the center of an octagon.

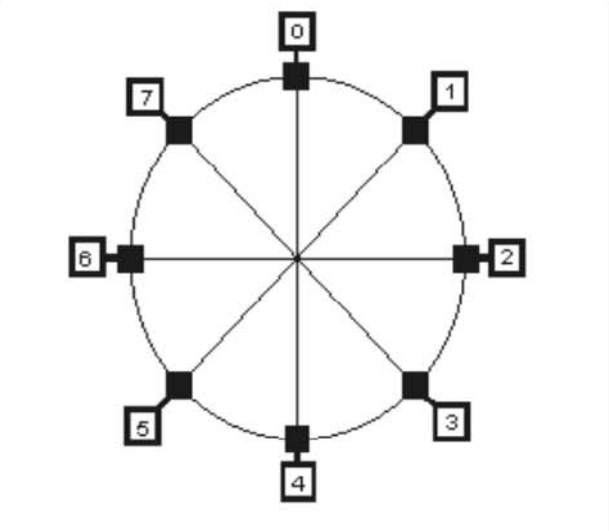


Figure 1 Octagon [22]

2.1.2 CLICHÉ

The CLICHÉ (Chip-Level Integration of Communicating Heterogeneous Elements) was proposed by Kumar et al. [14]. It is a simple 2D-mesh [18], where each IP is connected to a switch and each switch, except the edge switches, is connected to four other switches (Fig. 2). Here, the IP to switch and switch to switch interconnection is done via bidirectional links.

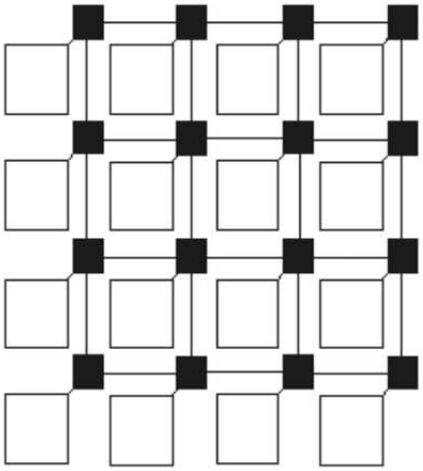


Figure 2 CLICHÉ [22]

CLICHÉ was proposed as a basic topology for NoC, since it is simple from layout perspective and scalable. It allows addition of resources, thus increasing the network, without reintroducing any changes in communication protocols, while increasing the network bandwidth.

2.1.3 2D Torus

Dally and Towels [7] proposed 2D torus, which is also mesh-based architecture. The only difference from 2D mesh is that the edge switches are connected via wraparound channels. Thus, every switch is connected to an IP block and four neighboring switches (Fig. 3). The number of switches is the same as for CLICHÉ and equal to the number of IP blocks.

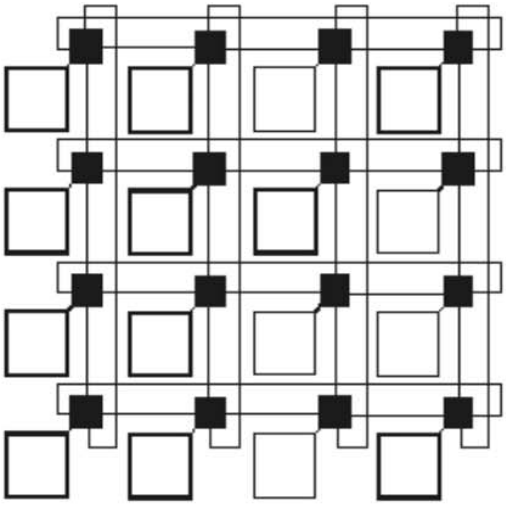


Figure 3 2D Torus [22]

The wraparound channels are very long and still increases with the number of IP blocks, thus the link delay increases. Long channels require repeaters to be placed within a link, so that the packet propagation fits within one clock cycle. As a solution, folded torus was proposed [19]. In a folded torus the wraparound channels are divided into approximately fixed length by shifting the position of nodes (Fig. 4). Hence, the channel length between any switches is now equal and considerably shorter.

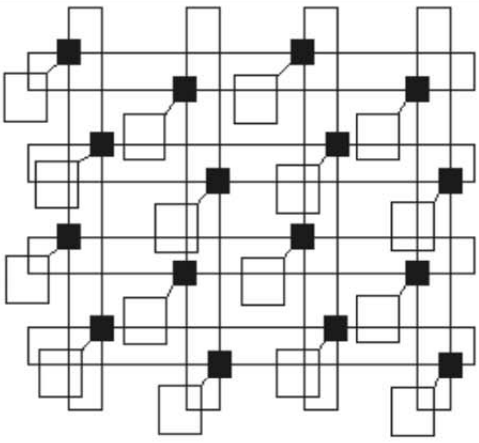


Figure 4 Folded Torus [22]

2.1.4 Tree Architectures

The tree-based architectures are derived from multistage interconnection networks (MIN) [18], where IP blocks communicate with each other by sending a message through a number of switch stages. Hence, IP block can communicate simultaneously without contention.

Guerrier and Greiner [20] proposed a SPIN (Scalable, Programmable, Integrated Network), which has a fat-tree architecture. In this architecture, every node has four children and the parent is replicated four times in every level (Fig. 5). The IP blocks are placed at the leaves of the tree and switches are at the vertices.

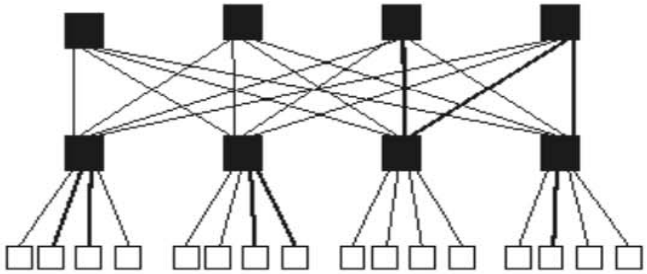


Figure 5 SPIN [22]

The size of a network grows as (NlogN)/8, where N is the number of IP blocks. Since, the SPIN architecture provides multi-path between any pair of IP blocks the wiring complexity increases with the number of IPs.

Another tree-based architecture, BFT (Butterfly Fat-Tree), was proposed by Pande et al. [21]. Here, again IP blocks are at the leaves and switches are at the vertices (Fig. 6). Each node is labeled by a pair of coordinates, (l, p), where l indicates a node's level and p indicates its position in that level. As the IP blocks reside at the lowest level of tree, their addresses are indicated as (0, N), where N ranges from 0 to (N-1). Each switch has four child ports and two parent ports, and IPs are connected to N/4 switches at the first level. At the j^{th} level of a tree the number of

switches is $N/2^{j+1}$, and the total number of switches approaches to N/2 as N increases. Thus, there is one switch for every two IP blocks.

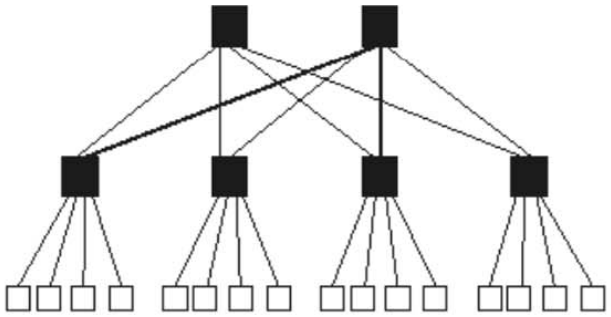


Figure 6 BFT [22]

2.1.5 Comparison of Network Topologies

Pande et al. [22] compared all these proposed interconnect architectures relative to throughput, latency, energy and area overhead. Under uniform traffic SPIN and Octagon show good performance in terms of throughput and latency. This is due to the fact that both of them have more links between two pairs of nodes than do the others. However, it was show that SPIN and Octagon have higher energy dissipation because of the higher degree of connectivity they provide.

When considering area overhead the Folded Torus and CLICHÉ are favored because of their simplicity. The inter-switch wires are of equal sizes and fit within one clock cycle in both of them. When the wires do not have equal sizes these with longer length require repeaters in order to be able to work at the same speed with others or force others to become longer. In BFT and SPIN wire length between switches depends on the levels of the switches. In Octagon, the inter-switch wires connecting IPs from disjoin Octagon units also require repeaters.

2.2 Switching Techniques

The network performance is highly dependent on a switching technique that is being used. Switching techniques determine when and how internal switches are set to connect router inputs to router outputs [18]. It does not decide which channel should be used by a message, but performs the actual mechanism that removes data from an input channel and puts it on the output channel. Various kind of switching techniques were proposed. These techniques are coupled with *flow control* mechanism in order to move messages through the network [23]. Since the network consists of channels and buffers, the flow control mechanism deals with the allocation of channels and buffers. The ideal flow control mechanism reduces the channel and buffer congestion while decreasing message latency.

2.2.1 Circuit Switching

In a circuit switching the path from source to destination is reserved before the message is injected into the network. The reservation is done by a routing probe that is send along the network. Routing probe contains the destination address and some control information. It progresses toward the destination reserving physical links. Thus, when routing probe reaches the destination the complete path is set up. The acknowledgment is sent to the source and a message is injected into the network. Since, the path is already set the message is transmitted at the full bandwidth of a path and does not required to be stored at an intermediate routers. The path is released either by destination node or by a few last bits of a message itself.

However, this technique is very conservative. Since physical path is reserved for the entire duration of a message and blocks other message. If a routing probe is blocked itself the physical path that was reserved by it up to now cannot be used by other messages, thus channels remain idle. This leads to a channel underutilization.

The circuit switching is advantageous when the message size is long compared to a set up time and infrequent.

2.2.2 Packet Switching

The packet switching technique was borrowed from computer networks [12]. In this technique the message is divided into fixed-length packets and every packet is sent to the network individually. First few bytes of a packet are allocated for packet header which contains routing information. This technique is also called store-and-forward (SAF) switching technique, since the packet should be entirely stored at an intermediate router before being forwarded to the next router.

In a networks that use packet switching the message latency is directly proportional to the distance between source and destination, and can be represented as $(L/B)*D$, where L is a message length , B is a channel bandwidth and D is a distance between source and destination. Every packet is independent of each other and needs to be routed at every intermediate router. Besides, packets can arrive out of order to the destination, thus producing additional overhead.

The packet switching is advantageous when the messages are short and frequent. In compare to circuit switching the resources are fully utilized and packets may be injected into the network simultaneously. However, if the packet size becomes large routers should provide buffers in order to be able store the entire packet.

2.2.3 Virtual Cut-Through

The packet switching requires entire packet to be stored in a router before routing decision is made. Since the packet size is usually bigger than the link bandwidth, it takes several cycles to transfer entire packet through the link. However, the header of a packet is available after a few cycles. Virtual cut-through (VCT) switching technique was introduced in order to decrease the message latency. The routing is

done as soon as the packet header arrives and if the next router has available buffer, cut-through to the input of a next router before the entire packet arrives. The message is send after the header in a pipelined manner. When there is no congestion in the network, the latency of a header is the routing latency, and the propagation delay through the router and physical channel. However, if a header blocks then the entire packet is stored at a router. Thus, each router still will have to be able to store the entire packet. At high network loads VCT acts like SAF switching technique. The latency is defined as $(L_h/B)*D+L/B$, where L_h is the length of a header. When, $L_h<<L$ the distance, D, will not have big effect on a latency.

2.2.4 Wormhole Switching

In a SoC environment the routers should consume small silicon area compared to IP blocks [22]. However, SAF and VCT switching techniques require the large buffers in order to be able to store entire packet, making it difficult in construction of small, fast and compact routers. As a solution wormhole switching technique was proposed [24]. The message is divided into small units called flits. The size of a flit depends on system parameters and generally is equal to *phit*, where phit is a unit of information that can be transferred across the channel in a single cycle. The message contains three types of flit: header, data and tail. The header flit contains the routing information and reserves resources toward the destination. Data flits and tail flit does not contain routing information and simply follow the header flit in a pipelined manner. Resources are released by tail flit, which indicates the end of a message. In compare to VCT where the blocked packet is stored in one router completely removing a message from network channels, wormhole technique blocks flits in place. Thus, the blocked message occupies buffers in several routers. This enables routers to have smaller buffer size, one or two flits.

Since the data flits do not contain routing information and simply use these resources that are reserved by header flit, different messages cannot interleave. Thus, if a buffer or channel was allocated for one message it is held for the entire

message duration. The latency of a message is defined as ((L_f/B)*D+L/B), where L_f is a length of a flit. When L_f <<L the distance between the source and destination, D, does not have much impact on a latency, unless D is very large. [12] indicates that in the absence of contention a message that is wormhole routed is shown to be nearly independent of the distance between source and destination.

2.2.5 Virtual Flow Control

There are two resource types in a network: buffers and channels. Usually, one channel is related to one buffer. If a packet A has allocated a buffer b_i, the associated channel c_i remains idle until A releases b_i. In networks that use flit-level flow control, where channels are reserved for the duration of a packet, if a packet is blocked then all the channels held by this packet remain idle. This situation leads to an inefficient use of physical channels and may lead to deadlocks (see Section 2.3) as well. In order to cope with this problem virtual channels (VC) [25] are adopted.

A virtual channel consists of a private buffer that can hold one or more flits [25]. Several virtual channels may share the bandwidth of a single physical channel. This property provides each channel in the network with multiple buffers. If a blocked packet holds one buffer associated with channel c_i, another buffer is available allowing other packets to pass.

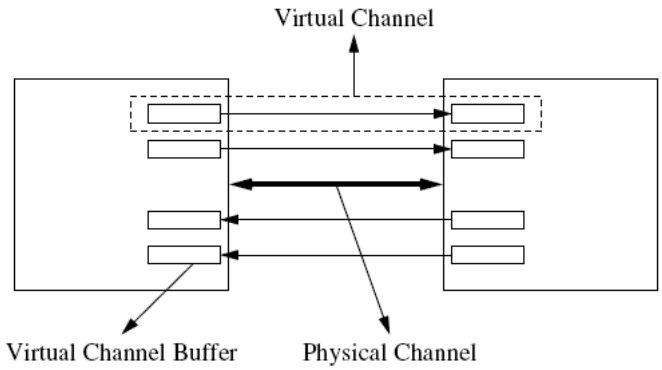


Figure 7 Virtual channels

Figure 7 illustrates two unidirectional virtual channels in each direction. This can be seen as if virtual channels are using two distinct physical channels at half the speed.

Virtual channel assignment is made at the packet level, meaning that, if a packet is assigned to a particular virtual channel than this assignment is fixed for the duration of the packet. It assures that different packets will not interleave, as data flits does not contain routing information. However, physical channel is allocated at the flit level.

Virtual channels were initially introduced for deadlock avoidance in [19]. However, lately they are suggested to be used also for performance improvement purposes.

Adding virtual channels increase network throughput, since physical channel utilization increases. However, virtual channel usage is limited. With every new VC the probability of a message being granted physical channel decreases, thus increasing message latency. Large number of VCs also has a big impact on router performance. Routers become complex, arbitrating between VCs and multiplexing them over one physical channel, thus increasing the hardware cycle time.

2.2.6 Comparison of Switching Techniques

There is no unique solution to all problems. Although wormhole switching is the most popular switching technique and significantly improves network performance, individual messages may block other messages, and lead to resource underutilization. SAF and VCT switching techniques consume network bandwidth proportional to network load and fully utilizes the network bandwidth. However, they require large buffer sizes to store packets. Circuit switching is very conservative and allocates the entire physical path, thus blocking other messages.

Wormhole technique can be improved by adding virtual channels. But the latency of individual messages may vary, because of contention for small buffers, in compare to SAF, which has predictable latency characteristics. At low loads wormhole routing performs better, but at high loads SAF is much better. VCT achieves the performance of wormhole switching at low loads and of SAF at high loads.

The SAF allows error detection and retransmission at link-by-link basis. It if flexible and easily avoids faulty components. Pipelining reduces buffer size, but introduces unique challenges in deadlock and fault avoidance, since the message is pipelined over several routers and form dependence between these buffers.

2.3 Routing Algorithms

The routing algorithm determines the path that must be taken by a message in order to reach its destination. Various kinds of algorithms were introduced in the literature. These are differentiated according to the place where the decision is taken about the path that should be traversed by a message and the routing flexibility to different network conditions. The decision of an entire path can be done by source node prior to message injection. This is called source routing. In this case, the packet carries the whole routing information, thus increasing the

packet size. Furthermore, the path is fixed and cannot be changed after the packet injection into the network. Alternatively, the path can be determined at every intermediate router using distributed routing. In this case, the decision is distributed through the network. Every router, upon receiving the packet, decides whether to deliver it to the local node or forward it to the neighboring node. The decision of a right neighboring node, where the packet should be delivered, is also implemented in an intermediate router. The routing algorithm can be implemented in adaptive way so that the path traversed by a message is dependent on the network traffic, or can be oblivious. In latter case, the path by a message is independent of network traffic.

Routing algorithm also has a big impact on network performance. The routing decision must be fast thus decreasing the message latency and be easily implemented in hardware. Complex routing algorithms increase the routing complexity and may require additional hardware. Besides, as it was indicated before, routers must be small and compact. That is the reason why the knowledge of network's global state information is not favored, since it requires additional storage space in each router. Among these requirements, routing algorithms should also be able to preserve deadlock, livelock and starvation freedom.

2.3.1 Deadlock, Livelock and Starvation

Unrestricted routing algorithms may lead to situations where a message can never reach its destination. These are caused by deadlock, starvation or livelock.

Livelock happens when a packet wonders across the network never reaching its destination. It happens when the resources requested by a packet are occupied by other packets. Adaptive algorithms are usually livelock-prone, since they allow messages to go away from their destinations. Hot potato [] is one example for livelock-prone algorithms. In this algorithm, if a channel required by a packet is busy, it is send through any other available channel, although that channel moves a message to the different path. Thus, livelock enforces some restrictions to routing

algorithm. This may be achieved by several techniques, such as minimal path, restricted non-minimal path, probabilistic avoidance [18].

The packet may also block permanently because resources requested are never granted to it and lead to starvation. The avoidance of starvation requires correct resource assignment scheme.

The most difficult problem in routing algorithm implementation is to preserve deadlock freedom. It is a situation when several packets are blocked permanently in the network. Deadlock occurs when the packets are allowed to allocate resources while holding others [12]. In VCT and SAF resources are buffers and in a wormhole and circuit switching resources are channels. The most deadlock-prone network is a network that uses wormhole switching technique since the packet holds several router buffers simultaneously.

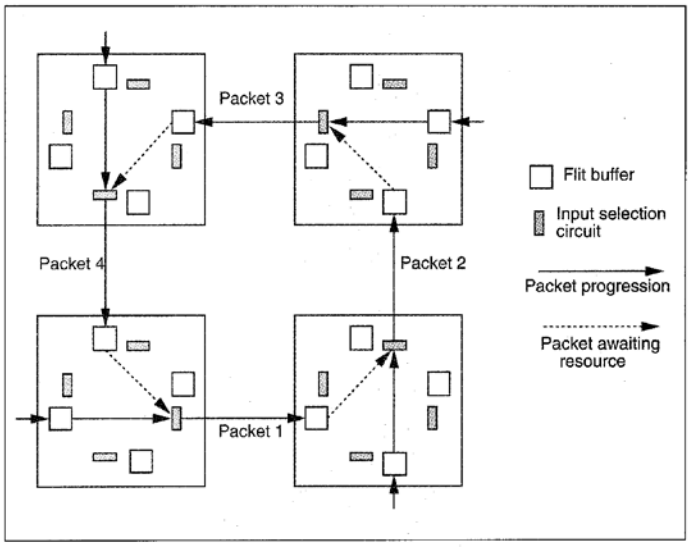


Figure 8 An example of deadlock [12]

Figure 8 represents a deadlocked situation where four routers and four packets are blocked. Here, packets are requesting for a channel while holding a channel requested by other message.

The most common technique is deadlock avoidance that is implemented by routing algorithm. Channels are granted in strictly monotonic order so that the packet is granted a resource only if the resulting global state is safe. The channel dependency graph [18] is used in implementation of deadlock-free routing algorithm. In this graph the vertices are unidirectional channels and the edges are connected channels. Two unidirectional channels are connected if the routing algorithm allows the usage of second channel (output) after the first (input). If all these connections form a cycle then the algorithm is deadlock-prone. Hence, some restriction should be embedded into the routing algorithm. Generally, it is done by disallowing some turns in a packet's path.

2.3.2 Oblivious Routing Algorithms

An oblivious routing does not consider the network state. The path taken by messages is only dependent on the source and destination of the messages. It may contain routing table that includes several options for an output channel based on the destination address. This may give some level of adaptivity, however the routing algorithm is independent of network state. They are used for general-purpose routing as they are easy to implement and considerably perform well. Oblivious algorithms can be deterministic or randomized according to the variations of options of output channels supplied at any intermediate router.

2.3.2.1 Deterministic Routing Algorithms

Deterministic algorithms always supply the message from particular source to particular destination with the same path. This means that every intermediate router always grants the same output channel for the same destination.

The deterministic routing became popular with the wormhole switching technique. As wormhole switching technique is heavily pipelined, it is extremely important that all the stages work at the same speed in order for it to be efficient. This

requires the routing algorithm to be implemented in hardware. Deterministic algorithm is usually preferable due to its simplicity [18].

The most popular deterministic routing algorithm is Dimension Ordered Routing (DOR), which is also known as XY (YX) routing when used in 2D-meshes and e-cube routing when used in hypercube. The DOR algorithm routes packets first in one dimension. When it arrives to the proper coordinate of the other dimension, proceeds in that dimension.

In XY-routing the message first travels on x-dimension. When the offset of x-dimension becomes zero, i.e. the message reaches the router that is on the same x coordinate with destination, it starts moving on y-dimension. Messages always are routed along the minimal path.

DOR algorithm is deadlock-free when used in 2D-mesh and does not require additional virtual channels. It preserves its deadlock freedom by allocating channels in strictly monotonic order. This restricts some kind of turns that could lead to deadlock.

2.3.2.2 Randomized Routing Algorithms

DOR algorithm is simple and fast, however when the system is heavily loaded it may act poorly. Because of its poor worst case behavior, a few algorithms using randomization were proposed. Randomization helps using the network bandwidth efficiently when the minimality constraints for routing algorithm are relaxed to some extend.

The oblivious algorithm, named Valiant, was introduced by Valiant and Brebner [27] and it is the best known randomized algorithm. This algorithm works in two phases. In both first and second phases, it uses dimension-order routing, at first phase - to route a packet from its source to randomly selected node, at the second phase it routes the packet from that random node to its destination. As Valiant is non-minimal, it tries to avoid congestion in the network. Packets with the same source and destination have different intermediate nodes, which are selected

randomly, thus they cannot take the same path. In order to avoid deadlocks buffers are needed, and as in a mesh topology, it requires two sets of independent buffers per communication link, so, this algorithm requires the buffers of O(n) depth in each node and does not preserve data locality. If livelocks are considered, the packet in the algorithm reaches its destination, even it may take longer path than actually required and so, in Valiant routing algorithm livelock cannot occur. The negative aspect of this algorithm is the path taken, because it increases depending on the topology being used. For example, in a mesh, the packet path may be doubled, resulting in longer routing times of Valiant routing, and so, doubling the demand on network bandwidth.

Randomized, Oblivious, Multi-phase, Minimal (ROMM) [28] algorithm is another routing algorithm which uses randomization. From dimension-order routing algorithm it inherits properties of minimality, which is to preserve data locality and to assure that the path taken by a packet will be minimal, and from Valiant – randomization, which is to assure that packets with same source and destination will not take the same path. These properties are combined to avoid congestion. Since the algorithm is oblivious, it is easy to implement it, in contrast to adaptive algorithms, which are more complex. As ROMM algorithms use the minimal path, it is constrained in randomization compared to Valiant that uses full randomization.

ROMM algorithms select random nodes within the range of a minimal path which a message is required to follow in order to reach the destination. A packet is routed in *p*-phases. In *p*-phase ROMM algorithm there are *p-1* randomly selected nodes $Z_1, Z_2, \ldots, Z_{p-1}$ between source and destination, and all Z_i 's (i=1,..,p-1) must be on minimal path from source to destination.

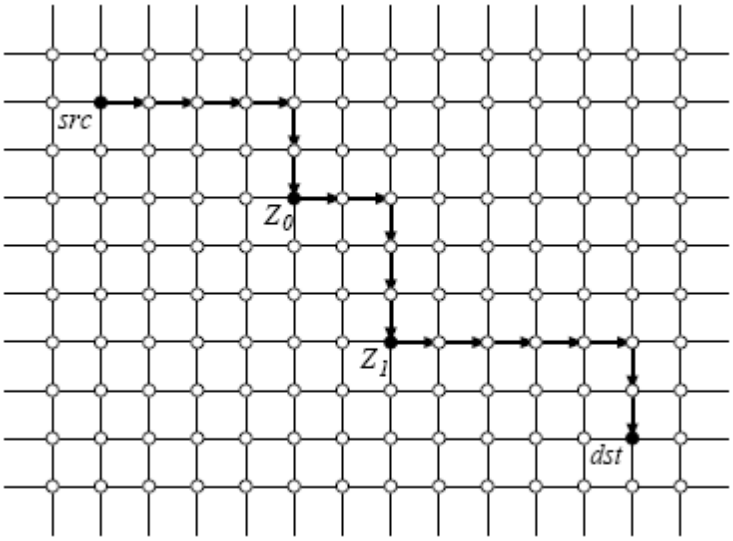


Figure 9 A possible path from source to destination using 3-phase ROMM on a 2-dimensional mesh.

Fig.9 shows a possible path from source to destination in a 3-phase ROMM. In this example source node establishes the path taken by message by means of dimension-order routing and chooses a random node that lies within that path, which is Z_0 in our case. Next, message is routed from *src* to Z_0 using dimension-order routing. In the same way, a new source node, which is now Z_0, sends a message using dimension-order routing having chosen another random node that lies along the minimal path from Z_0 to *dist*, in this case Z_1. Z_1 in its turn routes a message to *dist* again using dimension-order routing.

In order to avoid deadlock in *p*-phases ROMM algorithms *p*-virtual channels for wormhole routed mesh network is needed. Also, due to wrap-around links, it requires 2*p* virtual channels for wormhole routed torus network. As *p* increases the algorithm becomes very costly, since it increases memory usage adding virtual channels and complicates routing with additional logics required to manage virtual channels.

2.3.3 Adaptive Routing Algorithms

Another option in implementing a routing algorithm is to consider the network state. Every intermediate router supplies a message with several output channels based on the current node or buffer and the destination node. One of these output channels are then selected according to its status at the current node. Thus, a path of a message from source to destination is adaptive, allowing avoid busy channels. Adaptivity makes the routing more flexible however the hardware becomes more slow and complex.

Adaptive algorithm can be minimal or non-minimal. Minimal algorithms supply a message with a set of channels that always bring it closer to destination. Non-minimal algorithms may misroute a message by sending it away from its destination. It relies on the optimistic assumption that misrouting can bring a message to another set of minimal channels.

The further classification of adaptive algorithms is: fully adaptive and partially adaptive. The fully adaptive algorithms can use all the physical paths in its class, while partially adaptive algorithms can use only subset of them.

2.3.3.1 Planar-Adaptive Routing

The aim of planar adaptive routing algorithms, proposed by Chien and Kim [29] for *n*-dimensional meshes and hyper cubes, is to provide adaptivity in only two dimensions at a time and is considered to be minimal. While being adaptive they limit the routing direction in a series of 2-D planes, and as the packet moves toward its destination the routing dimensions change so that the packet is routed in all dimensions in its way to destination. By this the network cost is reduced while deadlock-freedom is maintained.

In order to avoid deadlocks, in a *k*-ary *n*-cube with no loop paths, planar-adaptive routing requires three virtual channels for each physical channel in meshes.

Let A_0 to A_{n-2} denote n–1 adaptive planes, each of them defined as the combination of several sets of virtual channels.

$$A_i = d_{i,2} + d_{i+1,0} + d_{i+1,1} \quad (1)$$

Each adaptive plane involves only two dimensions. To support the $n - 1$ adaptive planes three virtual channels in each dimension are needed.

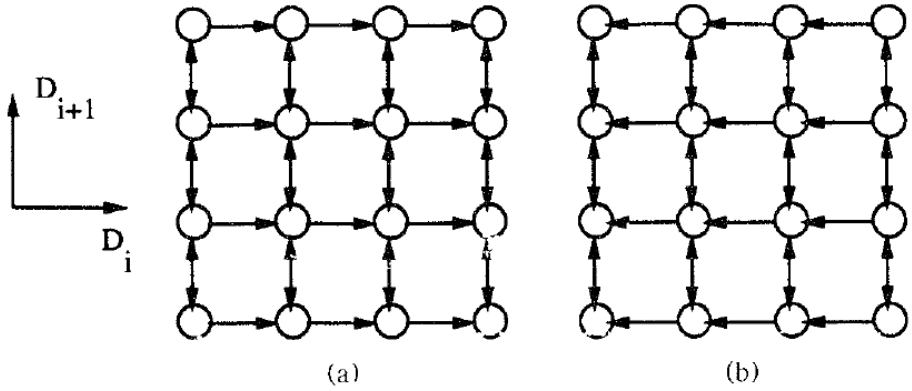


Figure 10 The channels for an adaptive plane A_i.
The increasing (a) and decreasing (b) networks are logically decoupled as they contain disjoint sets of virtual channels [29]

The algorithm conditionally can be divided into high level (between adaptive planes) and low-level (within adaptive plane A_i). In high-level each routing in adaptive plane A_i reduces the distance in d_i to zero. The packet reaches its destination when it is routed in all of the adaptive planes. Since for VC d_{n-1}, there is no adaptivity left for a minimal router, the packet is routed directly to its destination. In low-level routing, within each adaptive plane multiple paths can be chosen and the packet completes its routing in at least one dimension. If in plane A_i, the d_{i+1} the offset is reduced to zero first, routing continues in d_i until the d_i distance is reduced to zero.

The overlapping adaptive planes require only three virtual channels per physical channel for an *n*-dimensional network. VCs $d_{i,2}$, $d_{i+1,0}$ and $d_{i+1,1}$ belong to adaptive plane A_i and inside the plane, a packet is routed adaptively with respect to dimensions d_i and d_{i+1} by choosing the channels which directs close to the destination. In order to prevent deadlock, the traffic is divided into increasing and decreasing virtual networks which are completely disjoint: increasing network is

formed by VCs $d_{i,2}+$ and $d_{i+1,0}$ with packets which need to increase their d_i address while decreasing network formed by VCs $d_{i,2}-$, $d_{i+1,1}$, where packets cross dimension d_i in negative direction.

If decompose the network into the adaptive planes, it can be seen that routing in each plane in deadlock-free, and that cycles cannot form between planes. Consequently, the planar-adaptive routing algorithm is free from deadlocks.

2.3.3.2 Dimension-Reversal

In non-minimal adaptive routing packets can temporarily move away from their destination, but eventually achieve their destination. Due to misrouting, the distance a packet travels may not be minimal. This technique is fully adaptive and requires two VCs per physical channel. Two non-minimal routing algorithms for mesh networks were proposed by Dally et al. [30]. In the first of them, named static dimension reversal routing algorithm, every two adjacent nodes are connected by *r* pairs of channels. Thus, the network is divided into *r* sub-networks and all the i^{th} pair channels are contained in the i^{th} sub-network. Each packet header stores additional value *c*, the numbers to break the dependency cycle, which is initially set to zero. The packet with $c<r$-1 can move in any direction in its own sub-network, except when the packet moves from high dimensional channel to low dimensional channel the value of *c* is increased by one. If *c* reaches the value *r*-1 then the packet must switch into the deterministic dimension-ordered routing algorithm. The packets can be also misrouted, but the parameter *r* restricts the number of times it can happen.

The second algorithm is called dynamic dimension reversal routing algorithm. In this algorithm the channels are divided into two classes: adaptive and deterministic. At first, packets are sent through adaptive channels and can move in any direction. However, when a packet reaches a node, it must switch to the deterministic channels as at node all output channels are busy by packets with values of *c* smaller or equal to its own. Once a packet enters the deterministic channels, it cannot return to adaptive channels. The algorithm is deadlock-free.

2.3.3.3 Turn Model

Turn model, proposed by Glass and Ni [31], is partially adaptive algorithm, both minimal and nonminimal. It is based on the idea that deadlocks occur because the packet routes contain turns that form cycles. Deadlock cannot occur if there is no cyclic dependency between channels [18]. The turn in the packet route occurs when it is routed from one dimension to another. If packet changes its direction without moving to another dimension it is considered to have 180-degree turn. When physical channel is split into several virtual channels, moving from one virtual channel to another in the same dimension and direction is considered as 0-degree turn. Different turns when combined form cycles. The main concept of the proposed algorithm is to prohibit smallest number of turns so that the cycles are prevented. In order to develop a maximally adaptive routing algorithm in *n*-dimensional meshes and *k*-ary *n*-cubes following six steps are presented:

1- Classify channels according to the directions in which they route packets.
2- Identify the turns that occur between one direction and another.
3- Identify simple turns that may occur.
4- Prohibit one turn in each cycle.
5- Incorporate as many turns as possible that involve wraparound channels, without reintroducing cycles (in the case of *k*-ary *n*-cubes).
6- Add 0-degree and 180-degree turns without reintroducing cycles. 180-degree turns are needed when there are multiple channels in the same direction and for nonminimal routing.

The Figure 11(a) shows eight possible turns and two possible abstract cycles that may occur in 2-D mesh.

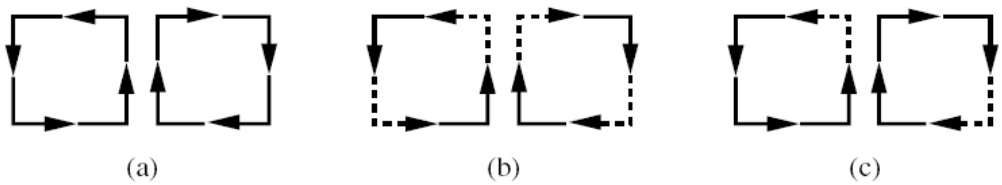


Figure 11 (a) 8 possible turns, (b) 4 turns allowed by XY-routing, (c) 2 prohibited turns in West-First routing [18]

XY- routing allows only four turns as shown in Figure 11(b), thus the deadlock cannot occur. Figure 11(c) presents turns that are prohibited in the algorithm presented by [31]. It is West-First routing, where two turns to the west are prohibited, thus in order to travel west packet must begin in that direction and then adaptively south, east and north.

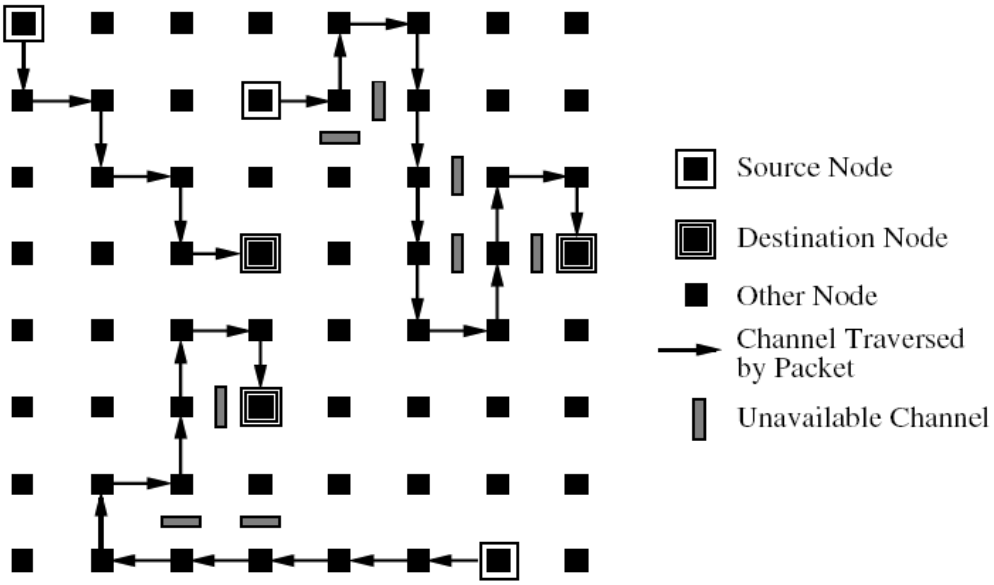


Figure 12 West-First routing [18]

Figure 12 illustrates how this algorithm works. Here, unavailable channel is either blocked or faulty channel.

Also some other algorithms can be derived from turn model, such as north-last and negative-first routing. The north-last routing algorithm does not allow turns from north to east or from north to west. The negative-first routing algorithm does not allow turns from north to west or from east to south.

2.4 Fault Tolerant Routing

The design of NoC is not limited by performance improvement issues. The second dominant issue is fault tolerance, which is the ability of the network to function in the presence of faulty components. The implementation of fault tolerance into the system may degrade its performance considerably. Moreover, the faulty network may lead the communication into deadlock and livelock, thus making routing algorithms implemented for fault free network ineffective.

From first glance it might appear that the adaptive algorithms can tolerate faults and move messages along faulty components. However, even the single link failure can demolish the deadlock-freedom property of adaptive algorithm. If adaptive algorithm provides several channels to a message and all these channels are occupied by other messages that have encountered a fault and cannot make a progress, this message blocks even if its destination does not lie along the faulty component. This means that the deadlock might occur even though cyclic dependencies between resources do not exist [18]. Since, these messages that encounter faulty component cannot make a progress and are holding buffer and channel resources, the other messages that do not require to traverse the faulty component, but need these resources that are held by blocked messages are also blocked. This leads to a *wait chain* that might block the whole system [18].

The deadlock might also occur due to dynamic faults, where a message in progress is interrupted by fault. In this case, if data flits lose their header flit they cannot make a progress without routing information and hold resources indefinitely. The recovery mechanism should be able to resolve wait chains, clean the network from interrupted messages, and recover corrupted messages or notify the source node about loss so that a new copy of a message is sent.

2.4.1 Fault Models

The fault tolerant algorithm is implemented according to faulty component patterns introduced, and the ability of a system to diagnose these faults. Diagnose can be conducted on different levels. Generally, only two classes of faults are diagnosed: either, the entire PE and its associated router fails (*node failure*), or any channel fails (*link failure*).

Faults may be *static* or *dynamic*, where static faults already exist when the system is powered on while dynamic faults occur on run time of a system. Both of them are considered as permanent faults and remain in the system until repaired. Permanent fault avoidance is much easier than transient fault avoidance. Transient faults cannot be reproduced and are not amenable to prediction.

The fault tolerance technique can be implemented in the system according to the fault occurrence rate. If the mean time between failures (MTBF) is much bigger than the mean time to repair (MTTR) then the system can utilize lower-cost fault tolerance approaches. However, if MTTR is much bigger than MTBF, as it can be seen in space-borne systems, then the fault tolerance approach should be more expensive so that the system will work, possibly with degraded performance, until the repair is conducted.

The fault model of a system should also define the behavior of a faulty nodes and the fault information that is available at a node. The faulty node should stop sending or receiving messages in order to avoid deadlock in a system. Faults are generally considered to be non-malicious, thus making fault tolerance much easier

in implementation. The fault information available at a node may be global or local. In global fault information, every node knows the status of every other node, thus making the best decision in routing. However, this technique leads to synchronization problems in conducting global information updates, and increase storage requirements at a node. In local fault information, every node knows the status of only neighboring node. This technique may force messages to take longer path, but the network implementation becomes easier.

2.4.2 Fault-Tolerant Routing Using SAF and VCT Techniques

In SAF and VCT networks the deadlock is avoided by right buffer management, since packets form deadlock only when there are no available buffer at an adjacent router. Fault-tolerant routing assumes packet misrouting, so the main issue is to avoid livelock. Usually SAF and VCT techniques are used in high-dimensional networks because in these techniques message latency depends on distance [12].

Chien and Shin [32] introduced a fault-tolerant routing algorithm that is used in binary hypercube (k=2). In binary hypercube messages traverse only one link in each dimension, so if the destination address differs in m bits from source address the shortest path will take m dimensions. They presented a new concept called as spare dimension, which is the dimension that is not on the shortest path. When all the dimensions that lie on a shortest path are blocked the message is misrouted and sent to spare dimension. Since in n-dimensional binary hypercube there is n disjoint path between any pair of nodes, it is assumed that n-1 faults cannot disconnect the network. So, the algorithm can tolerate up to (n-1) faults. By this assumption where a network can have only n-1 faults, Chien and Shin [32] asserts that there is always one spare dimension that can be used by blocked messages.

Another approach in order to avoid faulty components in a network is to use randomization. Some routers use this technique to avoid deadlocks [33]. If a message holds an input buffer for a long time, it is removed from input buffer and stored in a local buffer in a node. They are re-injected into the network later when

the output buffers become available. As the local buffer size is also limited and implemented as a queue, when the queue becomes full and there is some message that should also be stored, the random message is selected and misrouted. Thus, every message has a nonzero probability to avoid misrouting.

2.4.3 Fault-Tolerant Routing Using Wormhole Switching Technique

In compare the networks that use VCT and SAF switching techniques, wormhole switching introduces some unique challenges. The wormhole switching is deadlock-prone and even in fault-free networks requires a deep consideration in routing algorithm implementation. This is because the message is divided into flits and the blocked message occupies several routers, thus blocking other messages that are also competing for the same virtual channel. It is suggested to implement good routing algorithm rather than recovery mechanism, since recovery is expensive in terms of time and resources [18]. The general solution is to route messages around faulty regions as with VCT and SAF without introducing cyclic dependency between channels. The fault-tolerant routing algorithm is determined by the shape of a fault region and the base routing algorithm upon which it was implemented. Fault-tolerance requires additional virtual channels and the routing restrictions among their usage.

2.4.3.1 Fault Regions

The concept of *regions* was firstly defined by S.Kumar et al. [14], where the region *G* is defined as an area inside the NoC, that is insulated from the network and may have different internal topology and communication mechanism. A region may be some resource that is of larger size than the atomic slots in the mesh. They are connected to the rest of the network with special IO wrappers. The concept of region adds several advantages to NoC:

- A set of resources can be dedicated to a specific task.

- The communication between the resources inside the region can have different communication capacity.
- The resources inside the region are insulated from external traffic.
- NoC can accommodate a specific technology inside these regions.

However, these regions are constrained to have convex boundaries.

The concept of regions was further adopted by Chien et al. [29], where faulty components were insulated into region and messages are routed around that region. The fault region is also constrained to have convex boundaries. This constrained implies that the faulty components must form a block fault, so that every non-faulty component that is on the boundary of a fault region can have a faulty link only from one side. Since concave faults may also occur, they are transferred to convex faults by deactivating some non-faulty components.

The algorithm intoduced in [29] is a planar-adaptive routing (see Section 2.3), where the network is divided into two virtual netwoks: decreasing and increasing. There is one additional virtual channel in vertical direction. These messages that are traveling in increasing (decreasing) *x*-coordinates use increasing (decreasing) virtual network. When a message, that is traveling in increasing (respectively in decreasing) network, encounters a fault it is misrouted in *y*-coordinate using virtual channel 0 (respactively 1). Thus, cyclic dependency is prevented by using totally three virtual channels. The algorithm is adaptive and a message can use either dimension adaptively. The message is only misrouted when it is on the same row or column with destination and encounters a fault.

Planar-adaptive routing marks non-faulty components as faulty, thus decreasing the number of good links and nodes. If a faulty node is on the boundary of a network the message cannot be routed around the fault region and have to traverse backward. This bring new channel dependency between increasing and decreasing networks. As a solution the entire row or column is marked as faulty. The same situation can be seen for concave regions, where a message will need to traverse back, thus message are no longer monothonically increasing or decreasing. The convex faulty region simplifies the routing algorithm and deadlock avoidance, thus

any kind of region is desired to have convex boundaries. However, this constrain leads to unefficient usage of non-faulty nodes.

Chalasani et. al. [34] improved the concept of faulty region presented by [29]. In their fault-tolerant routing algorithm non-faulty nodes are not marked as faulty in order to obtain convex fault. They introduced a new concept of *fault rings* (f-ring).

F set contains all faulty links and nodes. The faulty region is considered to be rectangular faulty block (convex) if all the fault-free nodes around the faulty region connect and form rectangle and all the nodes lying in the rectangle are faulty. The block-fault-model assumes that each fault belongs to exactly one faulty block. If all non-faulty nodes around the f-region connect and form a ring this is called *f-ring*. When a fault region touch boundaries of a network non faulty nodes cannot form ring so this is called *f-chain* (F2 in Fig. 13). The nodes at which f-chain touches the boundaries of a network are called *end-nodes*. F-rings are said to be overlapped if two or more f-rings share one or more links (F3 and F4 in Fig.13).

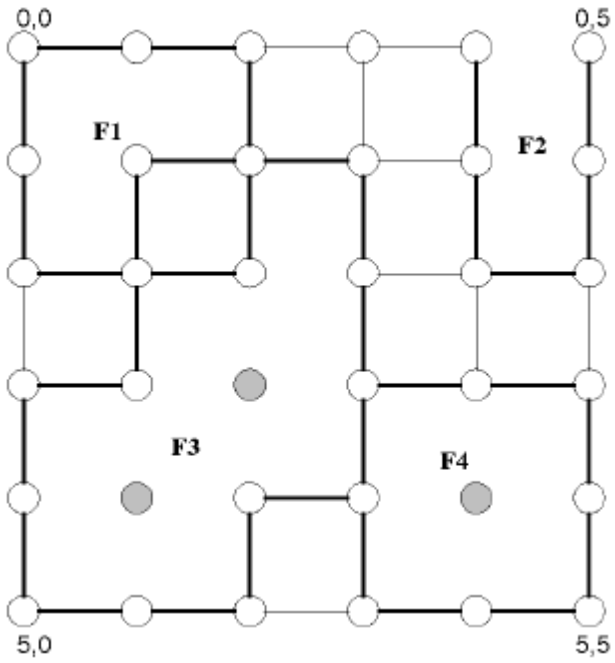


Figure 13 f-rings, f-chain and overlapping f-rings

[34] implemented their fault-tolerant algorithm upon the DOR algorithm, which is also called e-cube algorithm. Since DOR is deterministic algorithm it cannot tolerate even one faulty component. They divided the messages into column and row messages. The column and row messages are further divided into West-East (WE), East-West (EW), North-South (NS), and South-North (SN) messages, where WE and EW messages are row messages and NS and SN messages are column messages. A message that is still taking row hops is called row message, and a message that only requires column hops in order to reach its destination is called column messages. The row messages may change their status into column, but column messages never change their status into row messages. Two algorithms were presented: f-Cube2 and f-Cube4.

f-Cube algorithm uses 2 VC's and considers only non-overlapping block-faults. Row messages use c_0 VC and column messages use only c_1 VC. A message may have normal or misrouted status. The message's status is changed to misrouted if its next DOR hop is not available. Each time a message status is changed its direction is set, and its direction will not change throughout that f-ring. A misrouted NS (SN) message's direction is set to clockwise (counter clockwise). The EW message's direction is set to counter-clockwise (clockwise) if a destination is a row above (row below) than the current node. The same for WE messages.

The algorithm above proposed for non-overlapping fault-blocks is proved to be deadlock-free, since WE messages use only west boundaries and EW messages only east boundaries of f-ring. But when fault-blocks overlap, they share common columns or rows. Hence, the channels used by WE and EW messages are no longer disjoint. The similar issue arises in routing messages along f-chain, since messages that reach end-nodes are either stuck or make u-turn making channel utilization disjoint. In order to avoid deadlock f-Cube2 algorithm was modified to f-cube4 algorithm which uses 4 VCs. The algorithm divides the network into 4 disjoint networks WE, EW, NS and SN messages use c_0, c_1, c_2, and c_3 VC's respectively. Here, again the row messages can become column messages but not vice versa.

Many other proposed fault-tolerant algorithms [35] [36] [37] [38] are generally derivatives of f-Cube4 with various attempt to decrease the number of virtual channels to be used in routing around f-regions. These fault tolerant algorithms are recommended to be used as a backup algorithm in adaptive networks. Since, adaptive networks lose their adaptivity in last dimension left and messages are not guaranteed to be delivered if they encounter faults, with additional four VC the adaptive algorithm can tolerate any number of faults.

2.4.3.2 Fault-Tolerant Routing Without VCs

In order to make wormhole routed meshes fault tolerant, usually additional virtual channels are used. However, virtual channels come at expanse as they require extra control lines, buffer space and switching hardware. Ni et al. [39] uses turn model in their one-fault-tolerant routing algorithm which does not require additional virtual channels. However, this algorithm is (*n*-1) fault-tolerant, which means that in 2D mesh it can tolerant only one faulty component. The authors argue that misrouting should be used only to increase fault-tolerance degree and never just for increasing adaptiveness, since misrouting increase communication latency

One-fault-tolerant algorithm is based on the negative-first routing algorithm, where packets are first routed in the negative directions (west and south) of dimensions 0 and 1 then in the positive directions (east and south) of dimensions 0 and 1. The result of the modifications of the negative-first routing algorithm:

1. Route the packet west and south to the destination or farther west and south than the destination. Avoid routing the packet to a negative edge for as long as possible. If a packet encounters a fault along the negative edge, route the packet one hop perpendicular to the edge.
2. Route the packet east and north to the destination, avoiding routing the packet as far east or north as the destination for as long as possible. If a faulty node on a negative edge of the mesh blocks the path to a destination on the edge, route the packet one hop perpendicular to the edge, two hops toward the destination and one hop back to the edge.

Another algorithm that does not require additional VCs was presented by Duato et al. [40]. It is software based fault tolerant routing. In fault-free cases the messages are routed using DOR algorithm. A packet that encounters a fault is ejected from the network and passed to messaging layer of the local node's operating system. The messaging layer further decides whether to send the packet along a non-minimal path or send it via intermediate node. Thus, the packet is re-injected into the network with modified header. However, the message reinjection complicates the messaging layer of a node, since it has to make right decisions upon message reinjection. As the messaging layer should know what paths has the message taken, the header flit size increases because of routing information. Along with introduced overhead in messaging layer, reinjection also increases message latency.

Both algorithms are considered for environments where fault rates are relatively low. The goal is to keep the system functioning, with possibly degraded performance. They try to keep the feature of deterministic router designs, i.e. compactness and speed, thus avoiding additional virtual channels. So, these messages that do not encounter faults are minimally impacted.

2.4.4 Dynamic Recovery

As it can be seen from the previous chapters, wormhole routing is more difficult in avoiding deadlock. Since messages are sent in pipelined fashion, flits of one message occupy several routers. The destination information is stored only in header flit, so the data flits are heavily dependent on it. However, dynamic faults that occur on runtime may corrupt a link separating data flits from header flit. Since data flits cannot be routed without routing information, they remain in the network holding resources and thus leading to deadlock. The recovery from these faults should be held on run time.

2.4.4.1 Flit-Level Recovery

For a packet switching networks recovery can be easily constructed, so that the exactly-one injection is guaranteed. They are amenable to a link-level error detection and retransmission. However, the wormhole switching is not amenable to a link-level retransmission, since flits are spread over several routers. One solution for a wormhole switching technique that guarantees end-to-end delivery was presented by [41]. It implements link-level monitoring of flits. The copy of every flit is kept in a router until it receives acknowledgment that it was successfully received, which means that at least to copies of a flit remains in the network. Thus, if a flit encounters a fault while passing a link its copy can be retrieved from the last router it has leaved. The message is constructed from header, data, tail flits and a token. When a link fault occurs and the data flits lost their header flit a new header is generated for them. The message token is kept the same that was appended to original message. Thus, every router keeps the copy of header flit until token releases it. Those data flits that are on the other side of corrupted link with header flit will continue on the same way after the new token is appended to them. At the destination header flits are distinguished according to tag which indicates if the message is original or restart and the token according to replica or unique type.

2.4.4.2 Message-Level Recovery

Message-level recovery suggests discard the entire message from the network and retransmit it from source.

[42] suggested the message-level recovery algorithm which is applicable in network where the path taken by message from source to destination is released by destination node. When a message encounters a fault and is divided into two parts two type of control flits are send: forward and reverse flits. Forward flit moves toward destination and releases all the buffers and channels allocated by these flits that are closer to destination. Reverse flit moves toward source and releases resources allocated by the other part of a message that was separated from header flit. When the reverse flit reaches source the message is retransmitted and when the

forward flit reaches the destination the partially received message is discarded. As long as there is a path between source and destination the control flits are guaranteed to notify about the lost. However, this type of recovery requires additional virtual channels for recovery traffic.

Another recovery algorithm was suggested by [43], which is called *compassionless* routing. Here, the time of a message arrival at destination node is approximately calculated at source node. If it is clear that the message will not arrive at destination node after the tail flit is injected into the network the message is padded with *pad flits* thus the entire path is kept throughout message delivery. Every router distinguishes between data flits and pad flits. Pad flits are also used to release the path reserved by header flit. If the fault model is static after the approximate time calculation expires in source node, it sends an FKILL signal toward the destination thus releasing the path kept for a message. If the fault model is dynamic then an FKILL and BKILL signal are sent from the detecting router. FKILL signal moves toward the destination while BKILL signal moves toward source node. Since the path from source to destination is kept by pad flits, these signals do not require additional buffer at intermediate routers. But, in this case pad flits are consuming bandwidth, thus decreasing the network utilization.

2.5 Traffic Generator Models

In order to understand and unravel network power/performance related issues network traffic modeling are considered. Traffic patterns for network evaluation are divided into two following types [44]:

1. Application-driven traffic, which models the network and its clients simultaneously. In other words it is full system simulation.

2. Synthetic traffic, which captures the application-driven workload (easier to design and manipulate, therefore it is used more often) – uniform, Poisson, permutation traffics etc.

General prior research in the area of classic networks such as the Internet, Ethernet, and wireless LANs which transport TCP/IP, HTTP, and FTP traffic among others, demonstrated how application-driven traffic models and model-based synthetic traffic generators can facilitate understanding of traffic characteristics and drive early-stage simulation to explore a large network design space.

The class of NoCs has substantially different traffic behavior as compared to the traditional network fabrics. Thus, determining network architecture is one of fundamental aspect of Network-on-Chip design. Since there exists an enormous network design space with respect to topology, routing and flow control schemes etc, and the network design should be customized for applications or a class of applications, determining network architecture becomes difficult. For on-chip networks no complete traffic model exists. Designers frequently rely on simple synthetic traffic patterns such as uniform random, bit-permutation and tornado traffic to stress-test a network design.

Varatkar et al. [45] first proposed a traffic model for on-chip networks, but the model captures pair-wise traffic rather than traffic over the entire network, and modeled a single application. They present a traffic analysis approach to characterize the on-chip communication traffic pattern of different multimedia applications based on self-similar or long-range dependent (which was denoted as LRD^2) stochastic processes. The degree of self-similarity of the on-chip traffic using techniques based on the Hurst parameter was illustrated by analyzing the statistical properties of the arrival process at different points in a generic architecture for standard MPEG-2 applications. Having obtained the Hurst parameter, the designer can choose the minimal buffer size for the router at each tile, which ensures a certain quality of service (QoS) for running the multimedia application. That is, the Hurst parameter which characterizes the traffic pattern for a particular application helps in finding the optimal buffer length distribution which turns out to be the critical issue for the routers at each node in the on-chip

communication network. Also, they describe a method of generating synthetic traces with statistical properties similar to the original ones.

Application oriented traffic described by Jantsch et al. [46] reflects the communication distribution of the applications and allows designers to adjust the locality of traffic so as to analyze the network behavior under locality network. They introduce traffic characteristic parameters by mean of traffic configuration tree, which characterizes network traffic by considering distributions of messages along the three dimensions: spatial distribution, temporal characteristics, and message size specification. The spatial distribution describes the communication partnership between sources and destinations and classified into two categories, namely traffic pattern and channel-by-channel traffic. The temporal characteristics give details of the message generation probability over time such as constant rate, random rate, and normal rate. The size distribution classifies the length of communicated messages into uniform, random and normal. They introduced notion of distribution coefficient and locality factor and incorporated the case when locality factor is equal to zero. Thus they achieve the traffic to have uniform distribution to nodes with a different distance, i.e. distribution coefficient becomes independent of distance.

Since locality factor may be set individually with distance, the amount of traffic distributed to a certain distance can be controlled.

Peh et al. [47] proposed (H, p, σ) 3-tuple network traffic modeling. The 3-tuple (H, p, σ) answer three questions: (1) How often are bursts of packets injected at a router, and how large are these bursts? (2) How far do packets travel in the network from a source node towards their destination node? And (3), what proportion of total traffic does each router inject into the network? These three components are orthogonal or independent from each other.

The level of burstiness or packets per unit time is not constant and varies from one cycle to the next and this level of burstiness can be parameterized using single parameter, the Hurst exponent *H*. The quantity $H = 1 - \beta/2$, known as the Hurst exponent or “fractal dimension,” shows the presence and degree of scale-invariant memory. A value of 0.5 is a sign of no long-term memory effect (white noise), while values closer to 1 show increasing presence of the long-term memory.

The second parameter p is one of the two spatial components of traffic and abstracts the hop distance of traffic and captures its distribution. It can be equally determined by three factors: (1) the communication pattern of the application, (2) the computation power (hardware resources such as processing unit, memory etc.) of each router node, and (3) the mapping of the application onto the nodes. The first two factors decide the best possibility which network traffic mapping can achieve, in other words, they define a tight lower bound of the network traffic's average hop count. To derive this lower bound, first define the traffic acceptance probability $0 \leq p \leq 1$ such that when one source node is to send a packet, any other node in the network that is used by the same application will receive (i.e. in-transit node) or consume (i.e. destination node) that packet with acceptance probability p. A node with a relatively higher p has an increased capability of receiving or consuming a packet. Therefore a relatively high p denotes short-distance network traffic (source-destination nodes are close, i.e. localized traffic) while a relatively small p denotes longer distance (global) traffic as each node across the source-destination path possesses a smaller capability of consuming the in-transit packets.

Lastly σ represents the second spatial component of the 3-tuple model, and characterizes the distribution of the ratio of total network traffic that each router injects into the network. The traffic injection distribution shows the percentage of traffic, or the average probability of packet injection, that comes from each node. It therefore captures spatial characteristics of traffic. A greater σ presents a flatter, more evened-out distribution (a relatively large σ can model uniform distribution), while a smaller σ models more concentrated, uneven injected traffic, where a few routers inject most of the traffic forming hotspots in the network.

CHAPTER 3

NETWORK SIMULATOR AND SIMULATION ANALYSIS

The best way to analyze the system behavior is to make a simulation where all of its aspects are captured. We chose the general-purpose simulator, ns-2, to simulate NoC. Here, we present the simulation results solely considering the non-faulty system. The main goal of this chapter is to present the behavior of NoC under XY-routing. The different parameters that may influence the network performance are analyzed.

3.1 Network Simulator

To simulate NoC we used the general-purpose simulator, ns-2, which was developed by the Network Research Group at the Lawrence Berkley National Laboratory (LBNL). Ns-2 is an object-oriented, discrete-event driven network simulator implemented in C++ and OTcl (Object Tool Command Line) [48]. Users can set up and run a simulation by writing a Tcl script, where the topology, routing algorithm, protocols and applications can be defined. It provides basic TCP and UDP as the network transmission protocols, four routing strategies, such as Static, Session, Dynamic and Manual and many mechanisms for modelling traffic generation. Ns-2 also provides users with a built-in graphical animator tool, NAM (Network AniMator), which visualizes the flow of messages and the whole system simulated. These protocols that are not supported by ns-2 can be implemented in

C++ and added into the ns-2 library, which then can be used in simulation via OTcl script.

Since, ns-2 was designed for regular computer network simulation, like Local Area Networks (LANs), Wide Area Networks (WANs) and for satellite networks, radio propagation and wireless systems, it does not contain NoC protocols. Even though, in [49] [50] [51] the authors used ns-2 in simulation of NoC they did not implement its protocols but rather used built-in options.

3.1.1 Embedded Protocols

We had to implement our own router model presented in [52] in ns-2, because it does not consider router delay. Each time a packet arrives, ns-2 finds the appropriate slot that the packet should use and sends it to the output buffers immediately. Thus, packets latency is computed considering only buffer latency and link latency. Since, ns-2 pre computes all possible outputs before simulation begins using Dijkstra's Shortest Path algorithm, every node contains tables which contain slot numbers that should be used in order to reach certain destination. In NoC the routing algorithm is distributed, so the decision of output channel that should be taken is made in every intermediate node, so the routing algorithm is also implemented.

Ns-2 does not support flit level communication and drops random packets in links when the buffer overflows. In flit level communication the drop of header flit cause the data flits moving behind also to be dropped. Besides, dropping packets reduces system performance (i.e. multimedia applications). In our router model drops never occur and packets spend several cycles in the router passing different stages in order to allocate an output channel.

3.2 Router Model

Routers are the building blocks of NoC. Thus, the performance of interconnection networks depends on the performance of the routers and the optimization of a router significantly increases network throughput [53].

Generally, proposed router models assumed that the clock cycle time depends only on router latency, thus assuming that the router latency fits within one clock cycle. Peh et al. [52] suggested a more realistic router delay model, where the clock cycle time was fixed and the router consists of several pipeline stages that might vary and each stage takes one cycle.

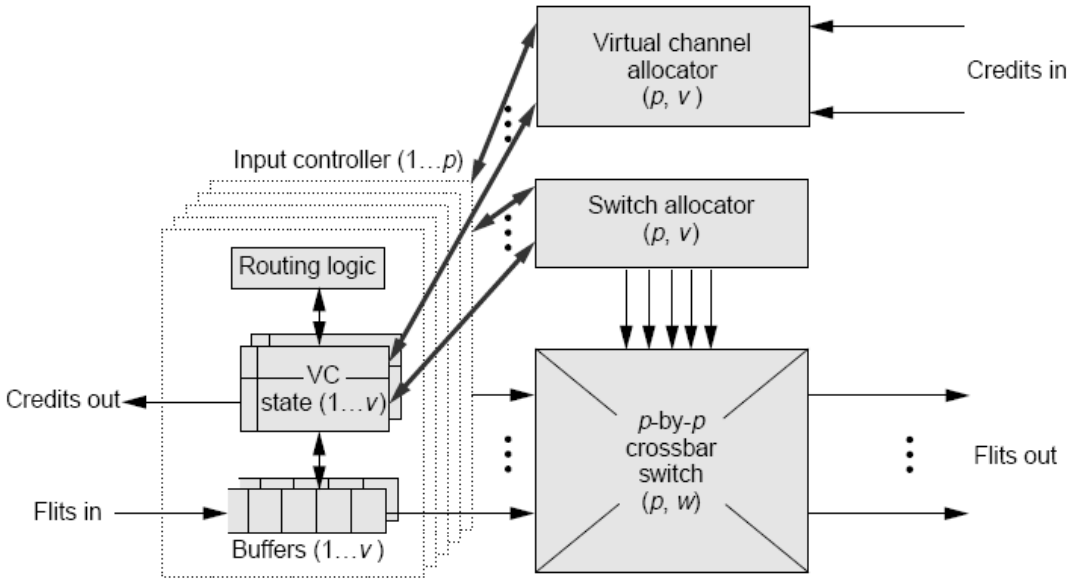


Figure 14 Canonical virtual channel router architecture [52]

Figure 14 presents a canonical virtual channel router architecture, where *p* is the number of physical channels, *v* is the number of virtual channels per physical channel, and *w* is the channel width in bits. Crossbar ports are shared between virtual channels of a physical channel, allocated on flit-by-flit basis.

Each virtual channel has a separate buffer and a state. The packet passes through routing, virtual channel allocation, switch allocation and crossbar traversal stages. Each flit has a VC-identifier (VCID) field, which defines the virtual channels where a flit is destined to. When a header flit arrives input controller decodes its VCID and injects the packet into virtual channel with associated VCID, where it is buffered. Virtual-channel enters a routing state and sends a flit's destination field to routing logic which returns the output virtual channel for a packet. Input controller then sets the virtual-channels state to virtual-channel allocation. Virtual-channel sends a request for those output virtual-channels to global virtual-channel allocator, which in turn returns the available output virtual-channels and updates the states of those output virtual-channels to unavailable. Once an output virtual-channel is allocated the header flit sends a request to global switch allocator for output port. When a request is granted the header flit leaves for the next node, with its VCID field changed to the recently allocated virtual-channel's VCID.

When data flits arrive they inherit the output virtual-channel reserved by their header flit so they can immediately send a request to global switch allocator for output port. The tail flit signals the virtual-channel allocator to release the output virtual-channels reserved by its header flit after gaining access to crossbar passage.

Each time a flit leaves, the buffer count for its output virtual channel is decremented in the current router, and the credit containing the input virtual channel is sent to the previous router, prompting it to increment its buffer count. Thus, a flit is never sent to next router until it has available buffers to store entire flit. This prevents the network from packet drops.

3.3 Network Simulation

In this section simulations are conducted in order to see the behavior of NoC under different conditions. These conditions are parameterizable values that can be

changed, like packet size, buffer depth at each input controller, number of VCs per physical channel, and network size. The evaluation is done in terms of message average latency and throughput.

Latency: Transport latency is defined as the time (in clock cycles) that elapses from the occurrence of message header injection into the network at the source node and the occurrence of a tail flit reception at the destination node. The average message latency is calculated as a sum of latencies of every received message divided by total received messages.

Throughput: Throughput is defined as maximum traffic accepted by the network. In our case it as a ratio of total received messages to total generated messages. Thus, if the number of received messages equal to the number of generated messages, then the network is working at 100% capacity. In all our simulation the throughput is given as the percentage of total received messages out of total generated messages.

Load normalization: Since injection rate depends on network size and packet size, it has to be normalized. In *k*x*k* network every node under maximum load (that is 1) is considered to inject into the network 4/*k* flits every cycle when the *k* is even. In our case, if we use 16x16 mesh the maximum load is 4/16=0.25 (flits/node/cycle).

We implemented router model described earlier in ns-2. The routing algorithm is XY-routing, where a packet moves first in x- dimension until it reaches the same position with the destination and then moves along y-dimension. Used switching technique is wormhole. The destinations are uniformly distributed across an eight-to-eight mesh network. All simulations ran under Poisson traffic. It is noteworthy that Poisson traffic model and uniformly destination distribution is considered to be non-realistic. However, there are no other realistic traffic models proposed for NoC, since it is not known yet how future SoCs will behave [44].

Each simulation ran for a 3000 cycles for a warm-up period and other 30 000 cycles for performance data collection. The latency of a packet begins when the message header is injected into the network at the source node and continues until

the last unit of information is received at the destination node. The communication load is given in terms of a percentage of the network capacity.

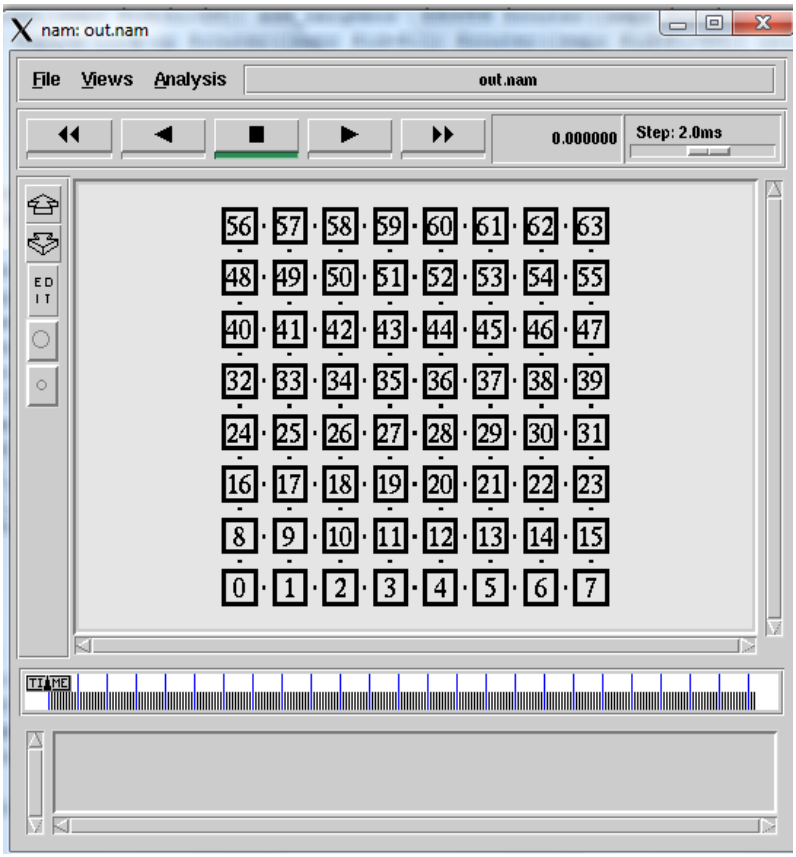


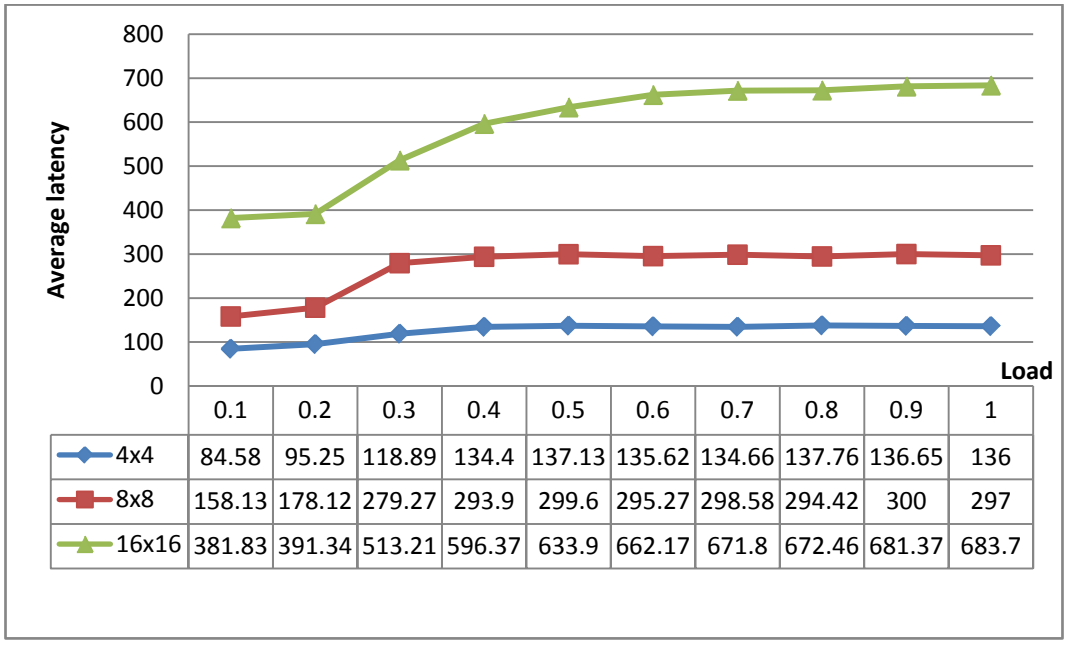
Figure 15 Visual graphic of a network in ns-2

Figure 15 shows the connection of routers on NAM. Since, traffic generators are attached to routers and they do not have graphical representation in ns-2 we decided not to show source nodes. Actually every node has one source, where packets are generated. In NoC router and source are connected via links and similar to every link in simulator they have one cycle propagation delay. So we simulated a link between a router and a source such that every packet sent by source reaches the router after one cycle.

These packets that reach their destinations are ejected from network immediately. Thus, we do not consider buffer size in source nodes and while the buffer size in routers is restricted the source nodes are considered to have unlimited buffer size.

3.3.1 Effect of Network Size

Figures 16 and 17 illustrate the effect of network size on average message latency and throughput respectively. The message size is 32 flits and the buffer depth is 4 flits per input channel. The performance is evaluated for 4x4, 8x8 and 16x16 networks.



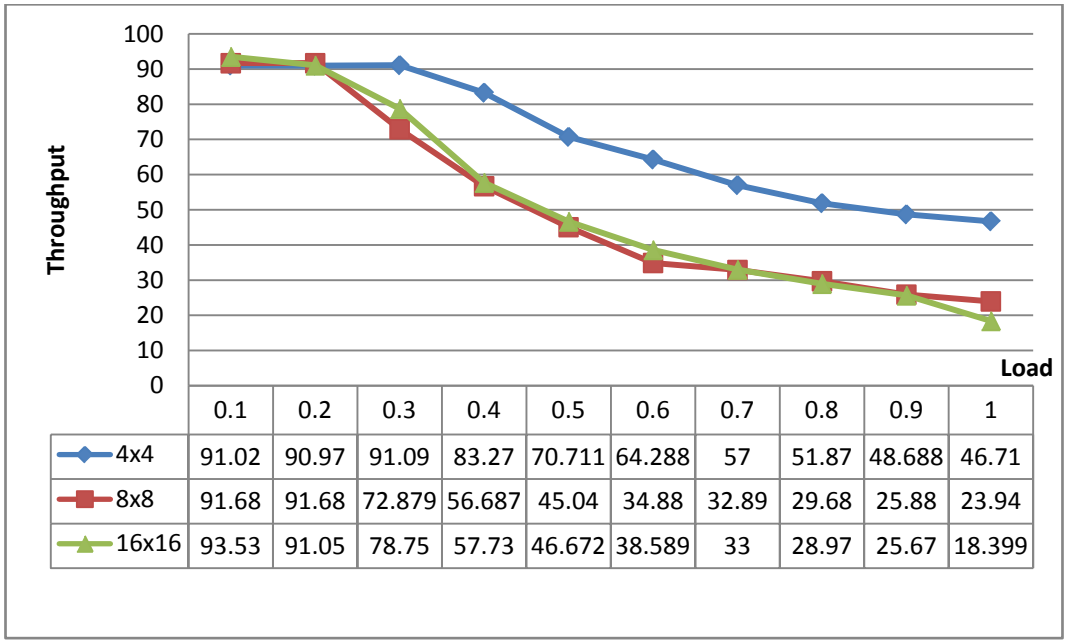
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
4x4	84.58	95.25	118.89	134.4	137.13	135.62	134.66	137.76	136.65	136
8x8	158.13	178.12	279.27	293.9	299.6	295.27	298.58	294.42	300	297
16x16	381.83	391.34	513.21	596.37	633.9	662.17	671.8	672.46	681.37	683.7

Figure 16 Average latency under different network sizes

As the network size increases the average message latency of messages also increases (Fig. 16). As destinations are uniformly distributed which means that every node has equal probability to become destination, with the increase of network size the distance between nodes increases. Thus, the messages have to spend more cycles in a network until they reach their destinations.

The throughput of a network with bigger sizes saturates earlier than the 4x4 network (Fig. 17). The increase of network sizes causes more messages to be injected into the network. From the other side message will spend more time in a

network thus blocking the resources. In 4x4 networks the nodes take advantage of their proximity which allows messages to reach their destinations quickly and thus free the allocated resources.

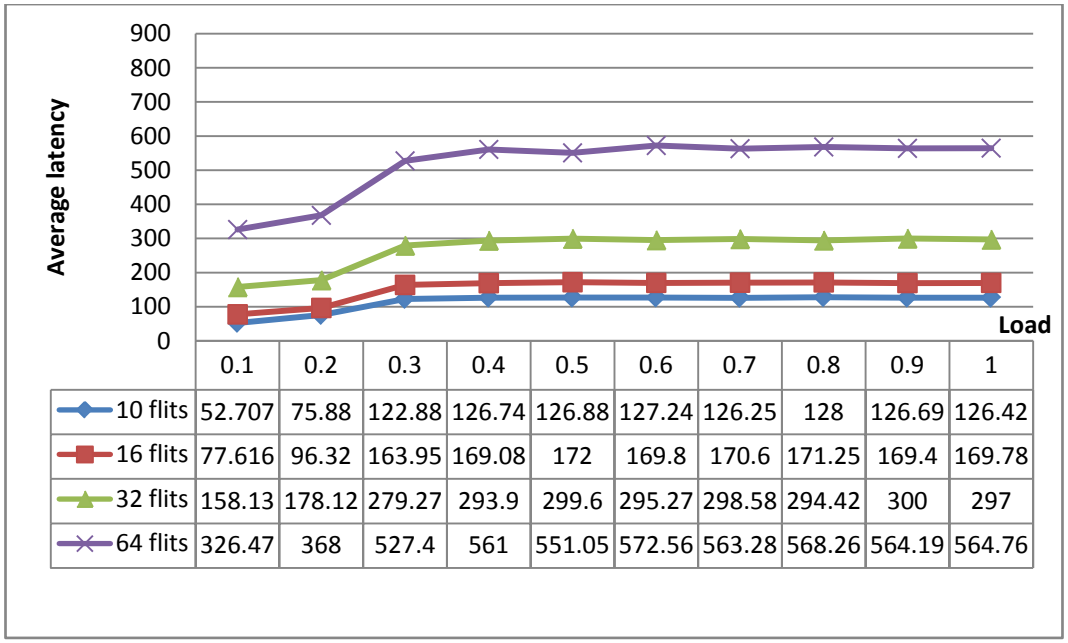


	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
4x4	91.02	90.97	91.09	83.27	70.711	64.288	57	51.87	48.688	46.71
8x8	91.68	91.68	72.879	56.687	45.04	34.88	32.89	29.68	25.88	23.94
16x16	93.53	91.05	78.75	57.73	46.672	38.589	33	28.97	25.67	18.399

Figure 17 Throughput under different network sizes

3.3.2 Effect of Packet Size

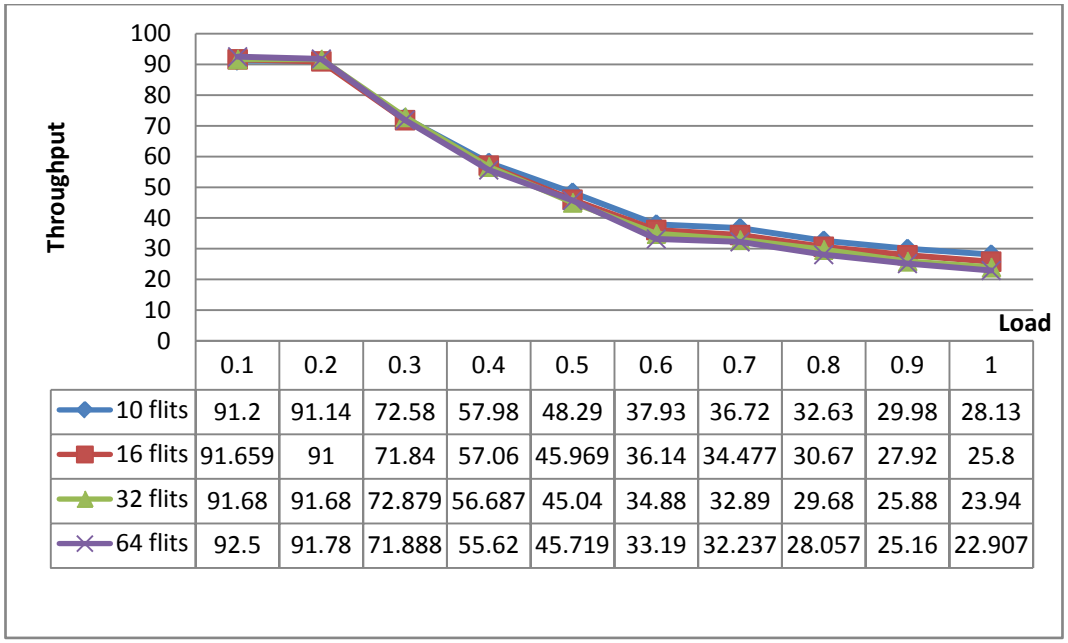
The effect of packet size on network performance is presented in Figures 18 and 19. Here, the network size is 8x8, and the buffer depth per input channel is 4 flits. The number of VCs per physical channel is one. The performance is evaluated for message sizes that are 10, 16, 32 and 64 flits long.



	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
10 flits	52.707	75.88	122.88	126.74	126.88	127.24	126.25	128	126.69	126.42
16 flits	77.616	96.32	163.95	169.08	172	169.8	170.6	171.25	169.4	169.78
32 flits	158.13	178.12	279.27	293.9	299.6	295.27	298.58	294.42	300	297
64 flits	326.47	368	527.4	561	551.05	572.56	563.28	568.26	564.19	564.76

Figure 18 Average latency under different packet sizes

The average latency of messages increases as the message size increase (Fig.15). This is natural, since the messages with bigger sizes spend more cycles in a network. However, the throughput does not change with the message size (Fig.16). As it was indicated before the injection rate also depends on message size. Thus, when the message size increases the injection rate decreases. This causes every node to send the same amount of flits independent of message size. We consider that the network throughput does not depend on message size but rather on the number of flits that are injected.

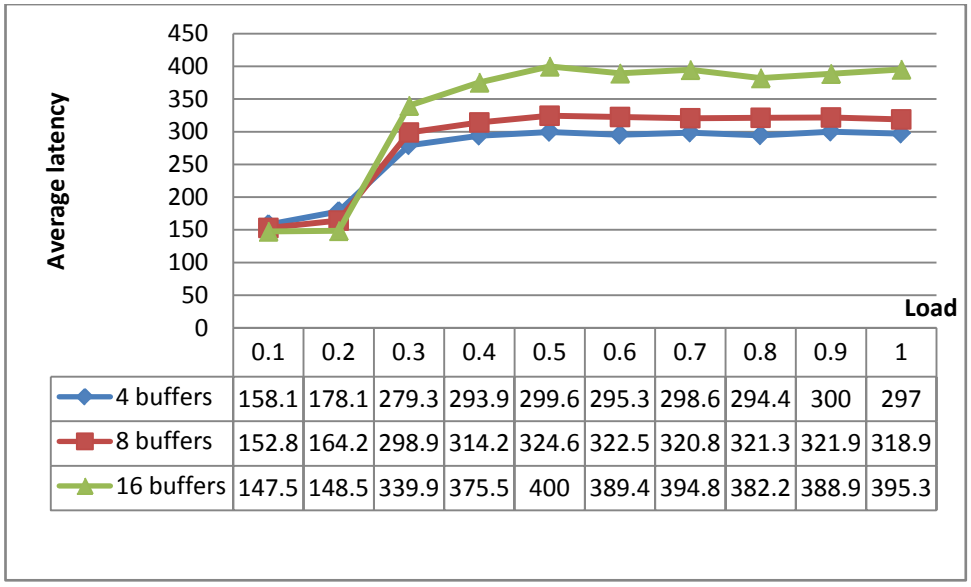


	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
10 flits	91.2	91.14	72.58	57.98	48.29	37.93	36.72	32.63	29.98	28.13
16 flits	91.659	91	71.84	57.06	45.969	36.14	34.477	30.67	27.92	25.8
32 flits	91.68	91.68	72.879	56.687	45.04	34.88	32.89	29.68	25.88	23.94
64 flits	92.5	91.78	71.888	55.62	45.719	33.19	32.237	28.057	25.16	22.907

Figure 19 Throughput under different packet sizes

3.3.3 Effect of Buffer Depth

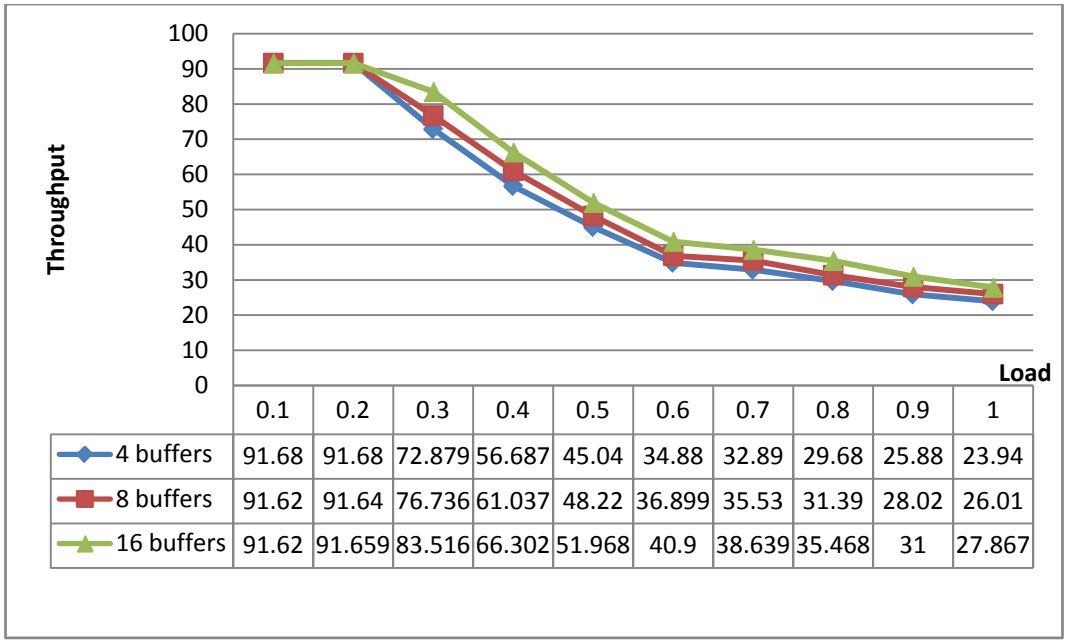
Figures 20 and 21 represent the effect of buffer depth on average latency and throughput respectively. The network size is 8x8 and the message size is 32 flits. The buffer depth changes from 4 to 8 and 16.



	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
4 buffers	158.1	178.1	279.3	293.9	299.6	295.3	298.6	294.4	300	297
8 buffers	152.8	164.2	298.9	314.2	324.6	322.5	320.8	321.3	321.9	318.9
16 buffers	147.5	148.5	339.9	375.5	400	389.4	394.8	382.2	388.9	395.3

Figure 20 Average latency under different buffer depth

As it can be seen from Figure 20 increase in buffer depth increases the message latency after the saturation point. It was indicated that the message latency begins when the header flit is injected into the network. The source node sends a flit to the router as soon as it finds out that the there is a free buffer available in uploading buffers. At that time the message's latency begins. As the buffer size increases, more flits can be injected into the network. However, there is a contention for resources in the router and these flits will have to wait longer in the router after being injected into the network. The bigger the buffer depth, the earlier the message is injected into the network and thus, the earlier its latency begins.



	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
4 buffers	91.68	91.68	72.879	56.687	45.04	34.88	32.89	29.68	25.88	23.94
8 buffers	91.62	91.64	76.736	61.037	48.22	36.899	35.53	31.39	28.02	26.01
16 buffers	91.62	91.659	83.516	66.302	51.968	40.9	38.639	35.468	31	27.867

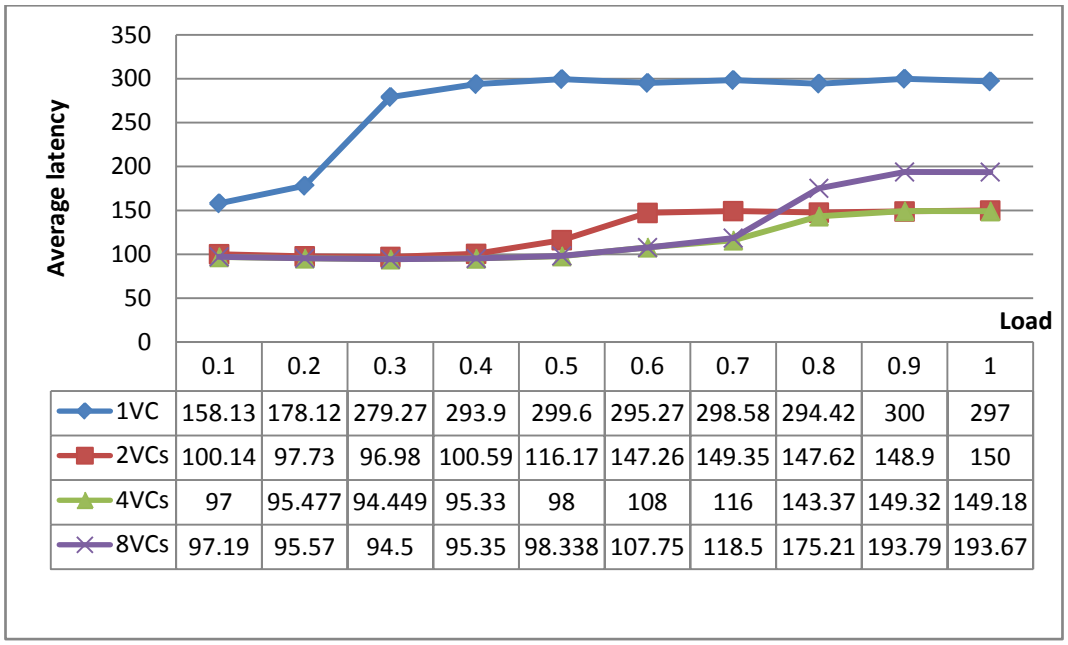
Figure 21 Throughput under different buffer depth

However, increasing the buffer depth slightly increases the throughput. Although flits that are newly injected into the network wait longer thus increasing the message latency, once they get the grant for the channel requested they path quickly to the next router. This releases the resources that are held by these messages. On the other hand the increase in throughput is not significant. In [2] it was also indicated that if the message size is small and wormhole switching is used, the increase in performance through increased buffer size is insignificant.

We consider that trying to achieve better performance by increasing the buffer depth will not give any advantages, since the improvement is small at the cost of bigger buffers. Buffer depth seriously influences the overall area. In [11] it was suggested avoid big buffers, since the increase of the buffer size at each input channel from 2 to 3 words, increases the router area of 4x4 NoC by 30%.

3.3.4 Effect of VC Size

Figures 22 and 23 show the results of the simulation, where different VC sizes per physical channel were used. Here, the packet size is 32 flits and the buffer depth per virtual channel is 4 flits. The performance is evaluated under VC sizes of 1, 2, 4 and 8.

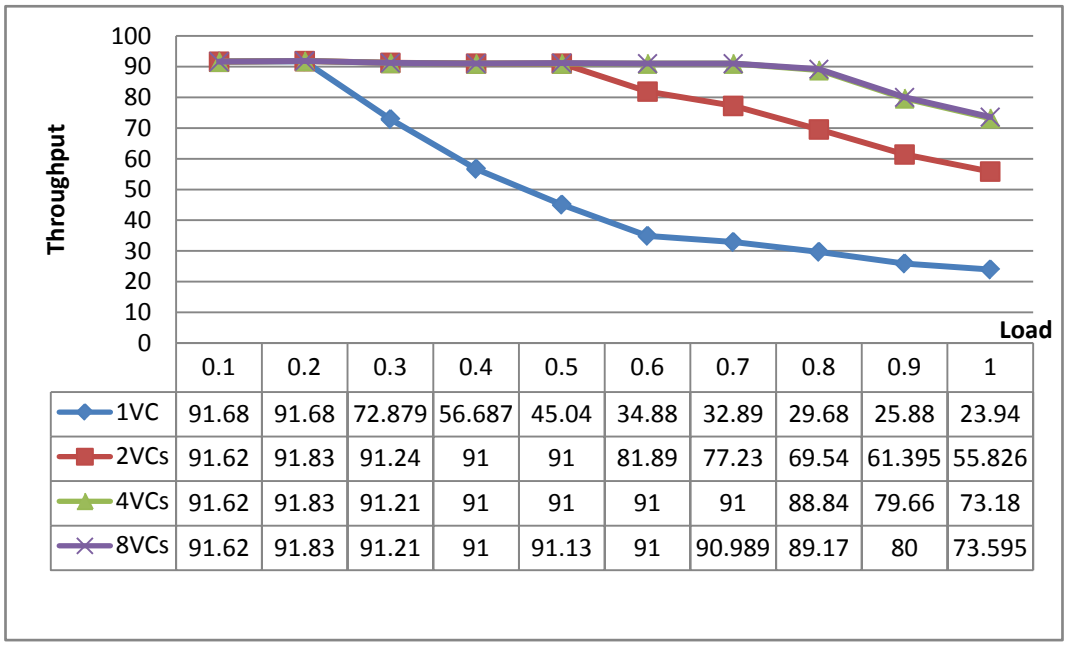


	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
1VC	158.13	178.12	279.27	293.9	299.6	295.27	298.58	294.42	300	297
2VCs	100.14	97.73	96.98	100.59	116.17	147.26	149.35	147.62	148.9	150
4VCs	97	95.477	94.449	95.33	98	108	116	143.37	149.32	149.18
8VCs	97.19	95.57	94.5	95.35	98.338	107.75	118.5	175.21	193.79	193.67

Figure 22 Average latency under different VC sizes

The average latency of packets is high when we use solely a channel without virtual channels (1 VC). The throughput rapidly decreases as the communication load increases. This is due to inefficient usage of channels, since a flit reserves it for entire packet duration. The best results are obtained for 4 VCs per physical channel, where both average latency and throughput present high performance. This result is consistent with researches in [25] [22], where four VCs are showed to be the optimal size. 8 VCs per physical channel yield good throughput but the latency increases after saturation point. This confirms the results presented in [52], where the author asserts that the latency increases after 8 VCs per physical channel. This rapid increase in latency may be due to complexity in the router that

is introduced by large VC numbers, as the router will have to arbitrate among many requestors competing for one physical channel. Thus, number of VCs is constrained to four, since it yields a balance between high throughput, low latency, and conservation of silicon area [25]. Since we consider that the message latency starts upon its injection into the network the graphs show the latency to be constant after saturation point. However, if we would consider also the time the message spends in source buffers before being injected into the network, the graphs would show that the latency is going upward. The increase in message latency is due to the fact that as the injection load increases till the accepted traffic limit, there will be more message contention.



	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
1VC	91.68	91.68	72.879	56.687	45.04	34.88	32.89	29.68	25.88	23.94
2VCs	91.62	91.83	91.24	91	91	81.89	77.23	69.54	61.395	55.826
4VCs	91.62	91.83	91.21	91	91	91	91	88.84	79.66	73.18
8VCs	91.62	91.83	91.21	91	91.13	91	90.989	89.17	80	73.595

Figure 23 Throughput under different VC sizes

As it can be seen from Figure 23 while the router that does not make use of any VCs enjoys the throughput of 30% capacity, the router with 2VCs and 4VCs increase the throughput up to 50% and 70% respectively.

CHAPTER 4

FAULT-TOLERANT ROUTING ALGORITHMS AND SIMULATION RESULTS

In this chapter we present the simulation results of the f-Cube4 fault-tolerant routing algorithm and our own, Shared Buffer (SB), algorithm. Some comparison between these algorithms is presented. Various kinds of fault situations such as non-overlapping fault-rings, fault-chains and overlapping fault-rings are induced and algorithm performances are evaluated. Besides, the faults are categorized according to their localization in order to see the performance of a network under random and localized faults.

4.1 Fault Model

We consider node and link faults. A node failure makes all links incident on it faulty. If all links of a node is faulty then this node is marked as faulty. However, if a node has faulty output link it marks its neighbor to which the link is incident, as faulty, even if the node is not faulty. Only non-faulty nodes generate messages. It is also considered that the messages are destined to non-faulty nodes. These are general assumptions used in literature.

Presented fault-tolerant algorithms work with local fault information. Thus, every node knows only the status of its neighboring nodes. As it was indicated in Chapter 2, global fault information requires additional storage for routing tables and leads to synchronization problems.

Faulty nodes and links form fault regions. We use the block fault model (rectangular faults), which requires a fault to belong to only one fault block. Under this fault model a fault-free node has faulty links incident on it in at most one dimension. Since fault-tolerant algorithms are applicable in only connected networks, where every non-faulty node has a path to every other non-faulty node, the block faults that touch both row boundaries or column boundaries of a network, thus disconnecting the network, are not considered. However, the algorithm can be applied to each connected sub network.

The concept of fault ring (f-ring) that was introduces by [34] is used. Here, non-faulty nodes around the region are connected to form a ring. If the faults are on the boundary of a network, non-faulty components cannot form a ring, thus non-faulty components connect to form an f-chain. The nodes at which f-chain touches the boundaries of the network are end-nodes. We require f-ring to be of rectangular shape. The fault-free node is in the f-ring if it is at most two hops away from a faulty node. Since network may contain several f-rings and if these f-rings share physical channels they are called overlapping f-rings. In our algorithm we consider f-rings, f-chains and overlapping f-rings of rectangular shape.

4.1.1 Formation of f-rings and f-chains

It is assumed that the nodes have self testing mechanisms that are variously presented in literature [54]. Each node monitors links incident on it. If a node x becomes faulty, each of its neighbors conclude that the link connecting it to x is faulty. Since we consider block faults each node can have faulty links in at most one dimension. Thus if a node has faulty links in more than one dimension, it marks itself as faulty and stops sending status signals to its neighbors.

Nodes on an f-ring can be in one of the eight possible positions: North West (NW) corner, North, North East (NE) corner, East, South East (SE) corner, South, South West (SW) corner, and West. Nodes determine their positions on an f-ring as follows:

1- If a node has faulty neighbors in first dimension (DIM 0, in our case it is *x*-dimension), it sends a message with information on its faulty neighbors to each of its non-faulty neighbors in DIM 1 (in our case *y*-dimension), and vice versa.
2- Each node determines its position according to received messages and its own link status.

Thus the f-ring is formed by using local fault information. The f-chain is formed in the same manner. The only difference is the end-node. However, only end-nodes can know that they are on the f-chain, while others consider themselves on f-ring since the algorithm is the same for both f-rings and f-chains. In the case of overlapping f-rings nodes will have to distinguish between two distinct messages. As a node can have faulty neighbors in at most one dimension, the fault information from two different neighbors that are on the same dimension can be categorized and split into two distinct f-rings.

As every other approach this approach relies on the absence of faults during the interval in which the system state is being updated. This interval is bounded by the network diameter.

4.2 f-Cube4 Algorithm

The f-Cube4 algorithm was briefly described in Chapter 2. It is built on XY-routing algorithm. The f-Cube4 requires 2 Virtual Channels (2VCs) per physical channel for non-overlapping f-rings. Here, messages are divided into row and

column messages. These messages that are still taking their row hops are either West East (WE) or East West (EW) messages, and these that have completed their row hops and passed to column hops are either South North (SN) or North South (NS) messages. Row messages may become column messages while vice versa is not true. VCs are divided into two sets where c_0 is used by row messages and c_1 is used by column messages.

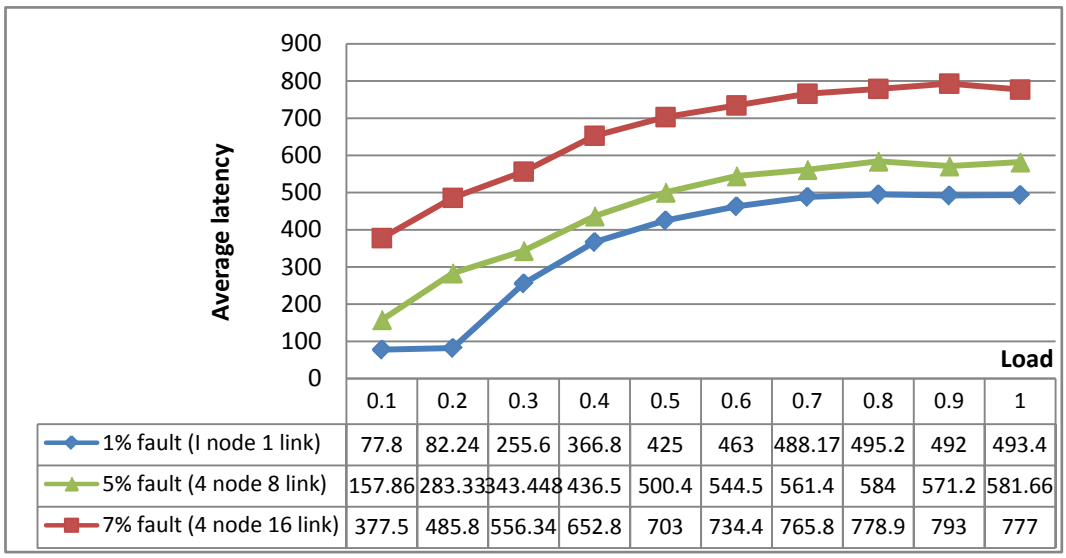
According to the f-Cube4 algorithm that is used for non-overlapping f-rings a direction of a NS message's that encounters fault (misrouted) is set clockwise and misrouted SN message's direction is set counter-clockwise. A misrouted WE message's direction is set clockwise (respectively counter-clockwise) if the destination is a row above (respectively row below) the current host. A misrouted EW message's direction is set clockwise (respectively counter-clockwise) if the destination is row below (respectively above) the current host. Either orientation can be chosen when the host is on the same row with destination. The algorithm uses the term e-cube hop, which is defined by [34] as a path that lies on a message's e-cube path. Recall that XY-routing algorithm is also called e-cube algorithm (see Chapter 2), thus the e-cube path is the natural path that should be used by a message under fault-free conditions.

In f-Cube4 algorithm messages are routed around the fault region in such a manner that the WE messages always use the west boundaries of f-ring while EW messages use east boundaries. NS and SN messages move in opposite direction and never share the same physical channel. However, when f-rings overlap, the East (South) boundary of one f-ring becomes a West (North) boundary of another one. Thus, the physical channels used by both row and column messages are no longer disjoint. As a solution two more VCs per physical channel are provided, overall of four VCs per channel. Now WE, EW, NS, and SN messages use their own VCs c_0, c_1, c_2, c_3 respectively.

4.2.1 Performance Evaluation of f-Cube4 Algorithm

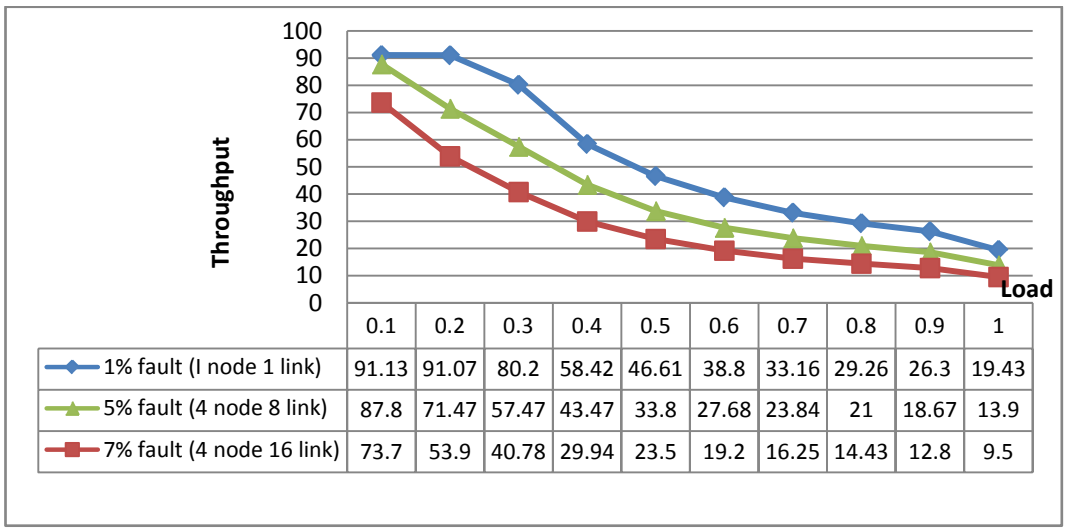
We have simulated a 16x16 mesh, since radix 16 is generally used in studies [34]. The network implements f-Cube4 routing algorithm. The implemented router model is the one presented in [52]. Since the f-Cube algorithm requires at least four VCs, every physical channel has four VCs. Injection rate is normalized. The latency of a packet begins when the message header is injected into the network at the source node and continues until the last unit of information is received at the destination node. The simulation results obtained in Chapter 3 shows that the packet size does not influence the throughput but only increases the average message latency. Hence, we decided to keep the message size small, 5 flits. It was also shown that the improvement is small when increasing the buffer depth. As we are trying to minimize the buffer size we chose it to be as small as possible and keep it as 4 flits per VC. Every simulation runs for 3000 cycles for warm up period and for 30 000 cycles for information gathering.

1%, 5% and 7% total link faults were injected randomly to the different places of a network. For the 1% case we have randomly chosen a node and a link to be faulty. Every node has four links incident on it, thus totally 5 links of 480 links are faulty. For the 5% fault case, four nodes and eight links are set as faulty, and for 7% fault case four nodes and 16 links are set as faulty. The faults are constructed in such that the f-rings, overlapping f-rings and f-chains are also covered. Figure 24 illustrates average latency obtained from simulation, and Figure 25 shows the throughput.



	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
1% fault (I node 1 link)	77.8	82.24	255.6	366.8	425	463	488.17	495.2	492	493.4
5% fault (4 node 8 link)	157.86	283.33	343.448	436.5	500.4	544.5	561.4	584	571.2	581.66
7% fault (4 node 16 link)	377.5	485.8	556.34	652.8	703	734.4	765.8	778.9	793	777

Figure 24 Average latency of f-Cube4 under various fault patterns

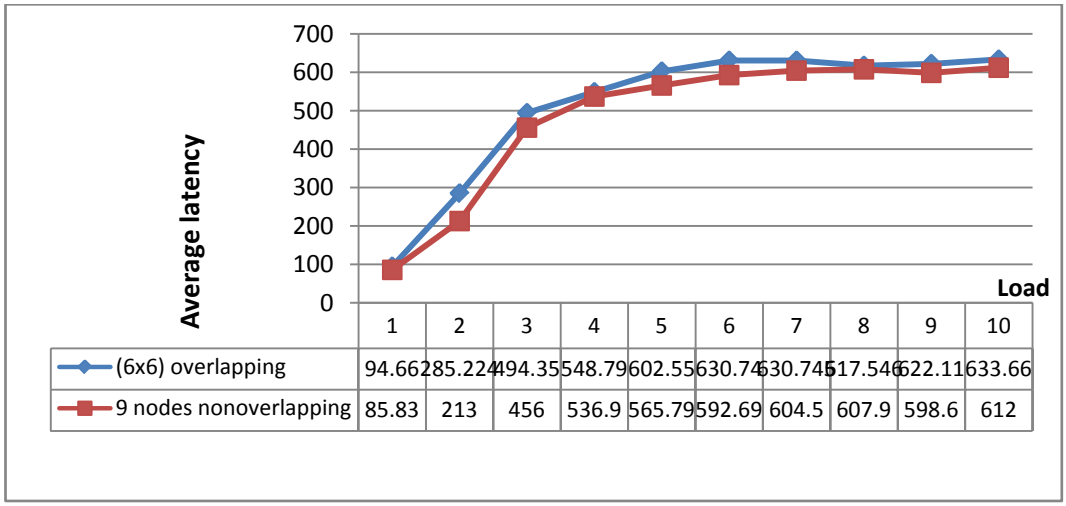


	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
1% fault (I node 1 link)	91.13	91.07	80.2	58.42	46.61	38.8	33.16	29.26	26.3	19.43
5% fault (4 node 8 link)	87.8	71.47	57.47	43.47	33.8	27.68	23.84	21	18.67	13.9
7% fault (4 node 16 link)	73.7	53.9	40.78	29.94	23.5	19.2	16.25	14.43	12.8	9.5

Figure 25 Throughput of f-Cube4 under various fault patterns

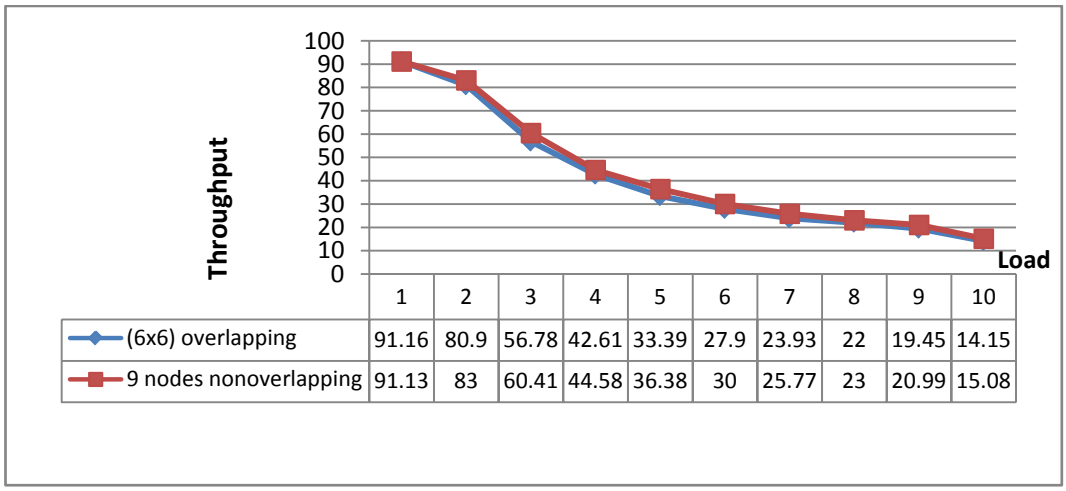
Since injection of random faults into different parts of a network increases the probability that the message encounters multiple faults, the performance degrades rapidly with the increase of load. The message average latency reaches up to 800 Cycles. In order to see the difference with the faults which occur randomly with those which are localized, we have injected multiple faults that are gathered in one place. Figures 26 and 27 present the results obtained for one non-overlapping f-

ring containing 9 nodes and 2 overlapping f-rings each containing 6 nodes. Both of them are placed into the middle of a network, which is the most heavily loaded part.



	1	2	3	4	5	6	7	8	9	10
(6x6) overlapping	94.66	285.22	494.35	548.79	602.55	630.74	630.74	617.54	622.11	633.66
9 nodes nonoverlapping	85.83	213	456	536.9	565.79	592.69	604.5	607.9	598.6	612

Figure 26 Average latency of f-Cube4 under localized fault sets



	1	2	3	4	5	6	7	8	9	10
(6x6) overlapping	91.16	80.9	56.78	42.61	33.39	27.9	23.93	22	19.45	14.15
9 nodes nonoverlapping	91.13	83	60.41	44.58	36.38	30	25.77	23	20.99	15.08

Figure 27 Throughput of f-Cube4 under localized fault sets

Performance of the f-Cube4 algorithm under the localized fault set is better in compared to random faults even though the overall link faults are more than these injected randomly (7.5% and 10%). This is because faults are localized and these

messages whose path does have to pass through the faulty components have very small latency thus decreasing the overall message latency. We believe that in reality the fault that is localized as a node failure affects others thus forms an f-ring in that place.

Naturally the average message latency of network with overlapping f-ring is higher than the one that contains non-overlapping f-ring (Fig.3). Routing around one f-ring take less time, while in overlapping f-ring a message has high probability to encounter a fault twice. However, the difference in message latency is insignificant. This is also valid for throughput. The reason may be that although overall faulty node number is bigger the overlapping f-ring contains a small amount of faulty nodes (6x6) in each f-ring, while non-overlapping f-ring contains 9 nodes in one f-ring.

4.3 Shared Buffer Algorithm

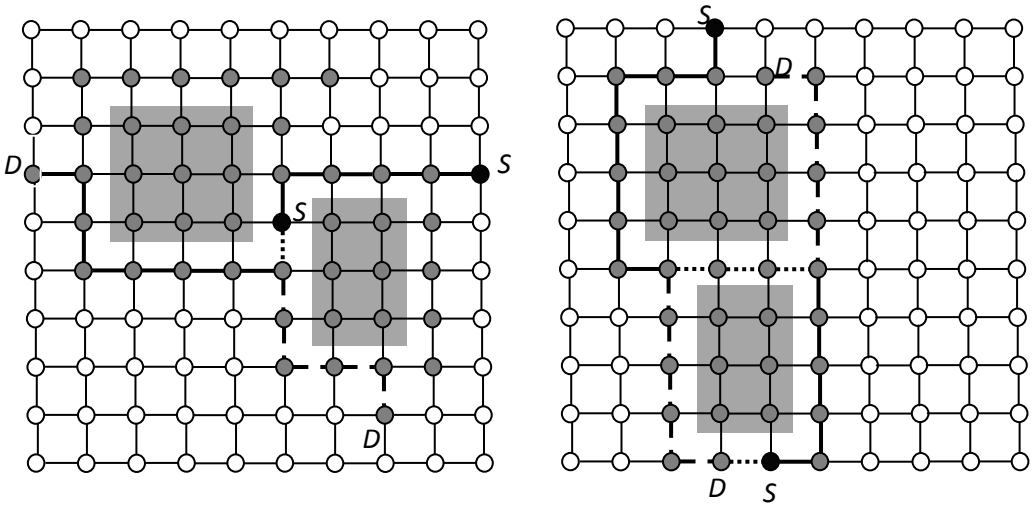
In a general router model presented in Chapter 3, edge buffers are used. Thus when the link fails buffers associated with it are also considered as faulty. Despite the fact that VCs are called channels, indeed they are normal buffers. Fault tolerance requires many buffers to be embedded into routers, as misrouting around the faulty regions leads to channel dependency thus creating deadlocks. Since it is not known when the fault occurs and from what side, every physical channel has equal size of VCs. However, when that physical channel fails, all these VCs remain idle. When a link is faulty it does not necessarily mean that associated buffers are also faulty.

We propose to make use of central VCs. This can be done by placing VCs in the middle of a router rather than assigning them to each input. Every input controller (IC) can see every VC. In fault free conditions ICs use their own set of buffers. However, when fault occurs some buffers remain idle. Thus, they can be allocated dynamically and be useful in avoiding fault regions. In order to see how useful it

can be, we have implemented the central buffer mechanism in the f-Cube4 algorithm, thus reducing the number of VCs to two that is sufficient for both non-overlapping and overlapping f-rings.

While implementing f-Cube4 algorithm we observed that the channels are significantly underutilized. This algorithm provides four VCs per physical channel. However, under fault free condition only one of them is used. Even in a faulty environment NS (WE) messages will never use their SN (EW) VCs and vice versa, since messages are not allowed to move backward, except the situations, when a message encounters f-chains. It is noteworthy that the NS and SN messages never encounter an f-chain on North and South boundaries of a network, while WE and EW messages never encounter it on West and East boundaries.

Figure 28 shows the movement of messages around overlapping f-rings. Since WE and EW messages share channels on *y*-axis (Figure 28a) they may block each other when multiple overlapping f-rings occur, if we use two VCs per physical channel. The similar condition is valid for column messages (Figure 28b). In order to separate movement of every message type, f-Cube algorithm proposes four VCs per physical channel. Thus, channels used by every message are disjoint, allowing messages to move in every direction around the f-ring and not causing deadlock.



(a) (b)

Figure 28 Overlapping f-rings sharing *y*-axis and *x*-axis

However, when closely observed it can be seen that row messages share channels only on *y*-axis and column messages on *x*-axis. Thus, messages need additional VCs only on their shared channels. This requires VC number to be different for particular physical channels. Due to unpredictable nature of fault occurrences, four VCs are used as a guarantee. When central buffers are used there are always free buffers around f-rings. They can be dynamically allocated, thus eliminating the need for additional VCs.

In our shared buffer (SB) algorithm NS and WE messages use shared buffers that are provided by those nodes that have faulty links. As it was described in our fault model, every node knows its position over f-ring. For blocked fault model a message can have faulty links in at most one dimension. If a node has a faulty link on its West or East side it divides VCs associated to that link to equal numbers (depends on VC number per physical channel), and notifies its North and South neighbors that it is sharing a VC. This can be done by sending a message containing the VC ID. The same is done if a fault is either on North or South of a node. In this case a node notifies its West and East neighbors. Thus every node that is on the f-ring contains shared buffers, except for the corner nodes. For this algorithm at least two VCs are needed, so that for one link failure in one dimension, a node can give one shared VC for each neighbor from the other dimension.

Row messages use c_0 and column messages use c_1. The movement of EW and SN messages is the same as in f-Cube4 algorithm. However, WE and NS messages use shared VCs in particular places. Since WE messages move in opposite direction to EW messages when an e-hop is provided they never share a channel. But, when a row message encounters a fault it has to turn to another dimension (*y*-axis in our case). In order to escape channel dependency WE messages look further to the position of a next node on an f-ring. If it is a corner node that indicates an end of an f-ring, it allocates the normal VC (c_0), otherwise it requests a shared VC that is provided by the next node. As long as a WE message enters shared buffers it moves along until it meets a corner node. As the corner node assures that its e-hop

is available from that position, WE message again uses the normal VC. The same concept is used for NS messages that enter shared buffers only when they have to move along the *x*-dimension. However, the additional constraint used for NS messages is that they move along shared buffers until they meet a corner node or until a *y*-hop is available. Here, we define a *y*-hop is a hop that the message was to move before it encounter first fault. Since in normal condition NS messages move only on *y*-axis, this constraint is required in order to ensure deadlock-safety. Now, the messages can move in any direction along the f-ring provided that the nodes around the f-ring share at least two VCs - one for a direction and other in the opposite direction.

The routing algorithm is deadlock free. Row messages may turn into column messages after a few hops, but column messages never turn into row messages. Since row messages use only class 0 virtual channels and column messages use only class 1 virtual channels, there cannot be deadlock involving both classes of virtual channels. Therefore, if there is a deadlock, then it is among the channels of class 0 or class 1 only.

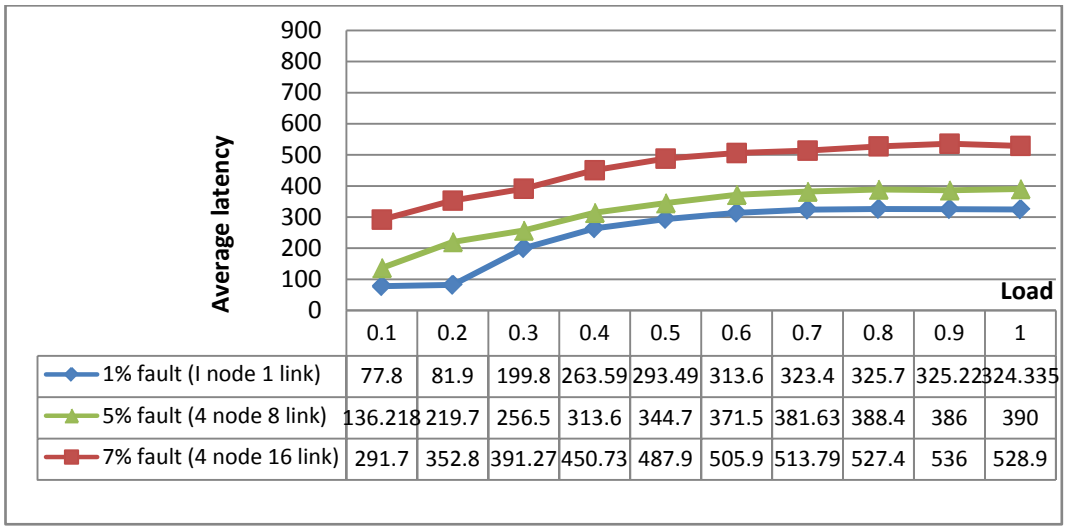
Class 0 channels are used only by WE and EW messages. When they move on the *x*-axis they use disjoint set of physical channels. When a WE message encounters a fault it allocates a shared buffer that is used only by the WE message on the *y*-axis. The only time a message uses VC of class 0 in a misrouted status is when it meets corner nodes that do not have shared buffers. However, corner nodes indicate that the e-hop of a message is available or that there are other shared VCs if the rings overlap thus, WE messages eventually free the corner channels and cannot block EW messages.

Class 1 channels are used by NS and SN messages. When they move on the *y*-axis their directions are always opposite, thus they use disjoint set of channels. When an NS message encounters a fault it moves along the shared buffers, which are used only by NS messages along the *x*-axis. The only time the NS message uses VC of class 1 in misrouted form is when it passes to corner nodes and while it is moving along the *y*-axis. However, as the *y*-axis is always free for NS messages and cannot

be used by SN messages that move in opposite direction, the corner nodes are eventually freed. Thus NS messages cannot block SN messages and vice versa.

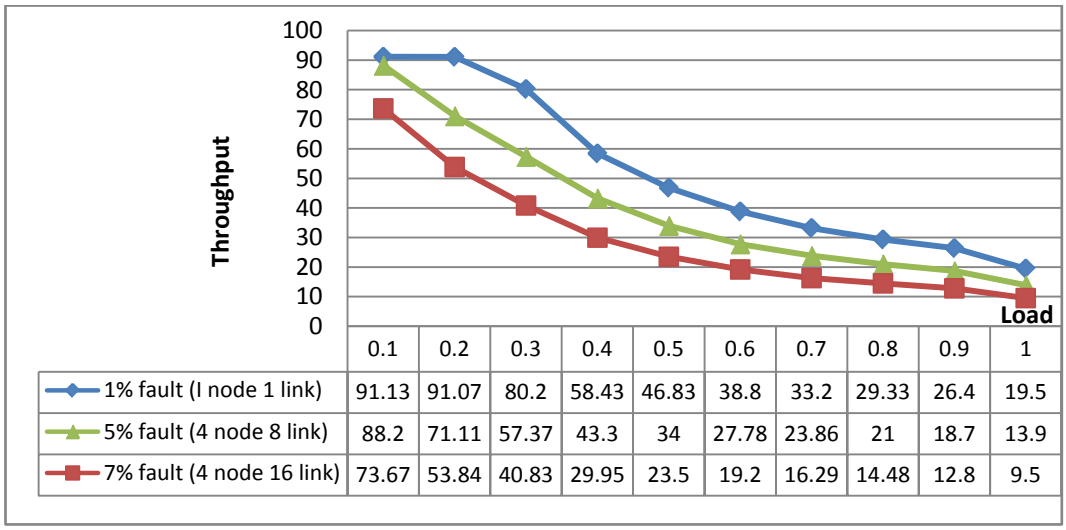
4.3.1 Performance Evaluation of SB algorithm

The same set of faults (1%, 5% and 7%) was injected into the network simulating the SB algorithm. The number of VCs per physical channel is two. The SB algorithm achieves the same throughput as an f-Cube4 algorithm. The performance of SB algorithm is shown Figures 29-32. The average message latency is significantly decreased. It will be shown later that even under the fault-free condition f-Cube4 has much bigger average message latency.



	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
1% fault (I node 1 link)	77.8	81.9	199.8	263.59	293.49	313.6	323.4	325.7	325.22	324.335
5% fault (4 node 8 link)	136.218	219.7	256.5	313.6	344.7	371.5	381.63	388.4	386	390
7% fault (4 node 16 link)	291.7	352.8	391.27	450.73	487.9	505.9	513.79	527.4	536	528.9

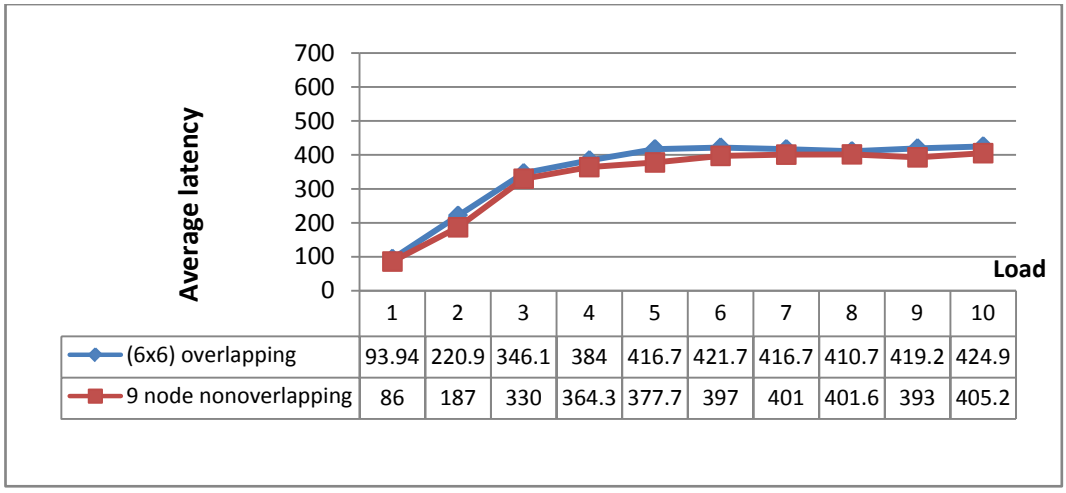
Figure 29 Average latency of SB algorithm under random fault



	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
1% fault (I node 1 link)	91.13	91.07	80.2	58.43	46.83	38.8	33.2	29.33	26.4	19.5
5% fault (4 node 8 link)	88.2	71.11	57.37	43.3	34	27.78	23.86	21	18.7	13.9
7% fault (4 node 16 link)	73.67	53.84	40.83	29.95	23.5	19.2	16.29	14.48	12.8	9.5

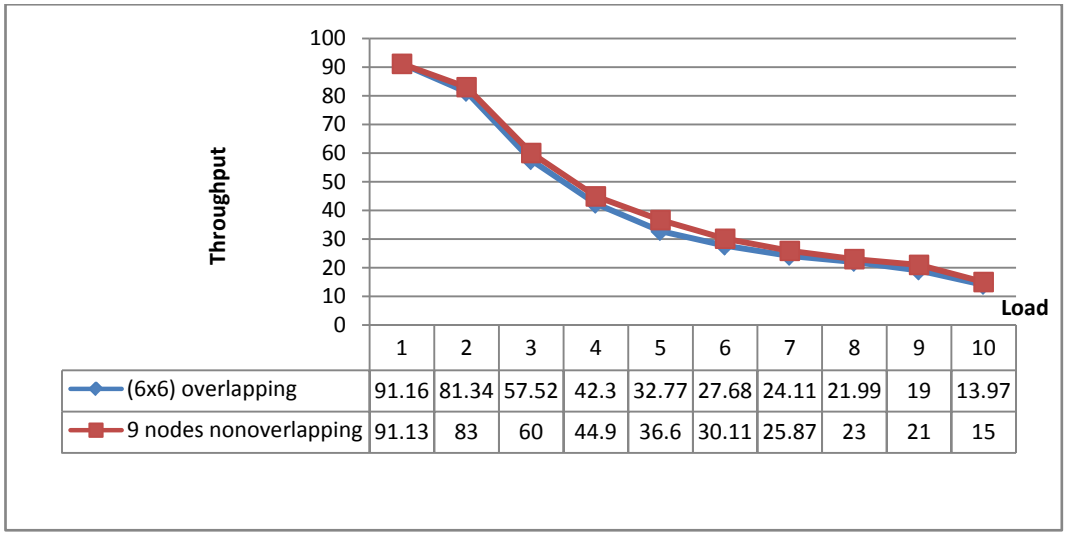
Figure 30 Throughput of SB algorithm under random faults

In 7% randomly placed faults SB reaches maximum of 540 Cycles average message latency, while it reaches 800 in f-Cube4 algorithm. The throughput is almost the same as f-Cube4 algorithm. This is explainable. While providing four VCs per physical channel f-Cube4 restricts their usage, so that only particular message types can allocate them. However, every node has its own internal traffic generator, which is also provided with these four VCs. Since a traffic generator sends a message from its internal queue to VCs as soon as they become free, the increase in injection rate always keeps these internal VCs full with messages. As there are four internal VCs there is high probability that one of them always has highest priority in allocating the VC that was also requested by an incoming message. Thus, a message spends several cycles (depends on message size in flits) in every router waiting for a requested VC to be available. This problem of multiplexing multiple VCs over one physical channel reduces in the SB algorithm, thus reducing the message latency. However, the throughput remains the same, since the router using the f-Cube4 algorithm sends the same amount of flits every cycle as in SB routers.



	1	2	3	4	5	6	7	8	9	10
(6x6) overlapping	93.94	220.9	346.1	384	416.7	421.7	416.7	410.7	419.2	424.9
9 node nonoverlapping	86	187	330	364.3	377.7	397	401	401.6	393	405.2

Figure 31 Average latency of SB algorithm under localized faults

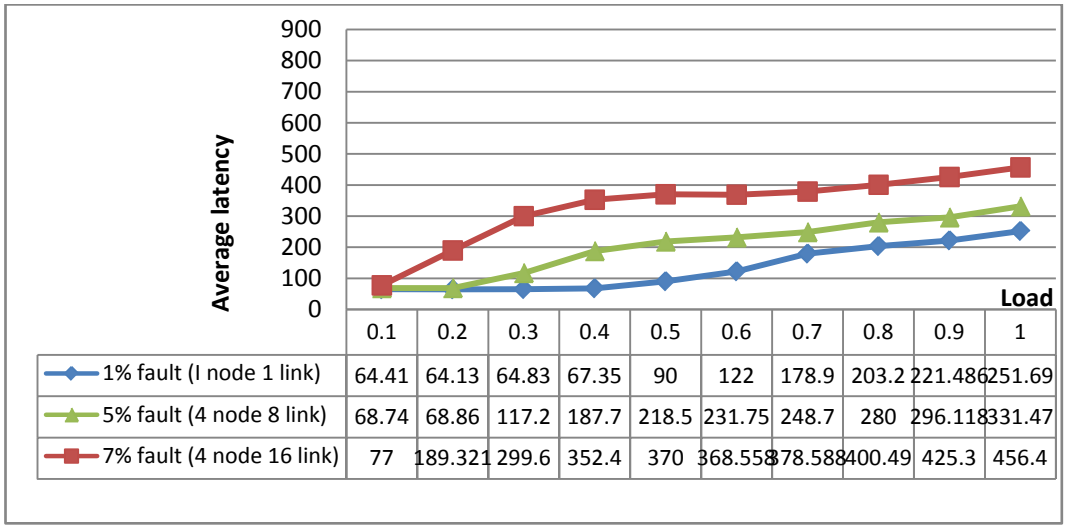


	1	2	3	4	5	6	7	8	9	10
(6x6) overlapping	91.16	81.34	57.52	42.3	32.77	27.68	24.11	21.99	19	13.97
9 nodes nonoverlapping	91.13	83	60	44.9	36.6	30.11	25.87	23	21	15

Figure 32 Throughput of SB algorithm under localized faults

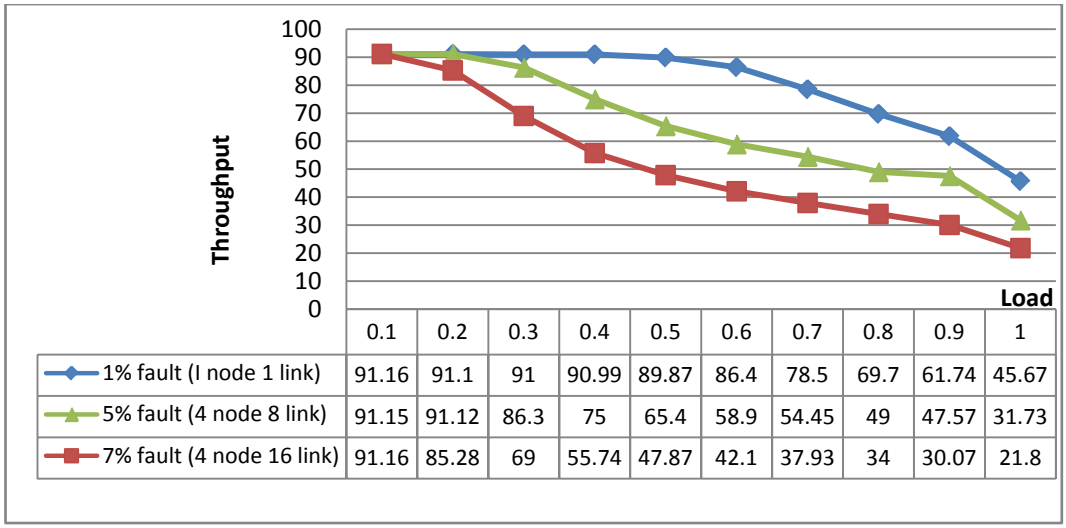
In order to see what can be achieved by using the same number of VCs that were used in f-Cube4 algorithm we added two more VCs to the SB algorithm. Here, c_0 and c_1 are used by row messages, and c_2 and c_3 are used by column messages. This may look similar to f-Cube4 algorithm. However, in f-Cube4 an incoming message can request only one VC according to its type. In SB with four VCs an incoming message can request either of the two VCs, thus reducing the blocked message number in a router.

The simulation result for SB4 algorithm is presented in Figures 33-36. Since SB4 algorithm provides messages with alternative VCs the network saturates much slower than the previous two algorithms. The average message latency reaches up to 450 Cycles under 7% random faults. This is nearly the half of the average latency that is achieved by f-Cube4 algorithm. The throughput is almost 1.5 times better than SB and f-Cube4 algorithms. This indicates that SB algorithm can achieve much better performance when using the same number of VCs that is used by f-Cube4 algorithm.



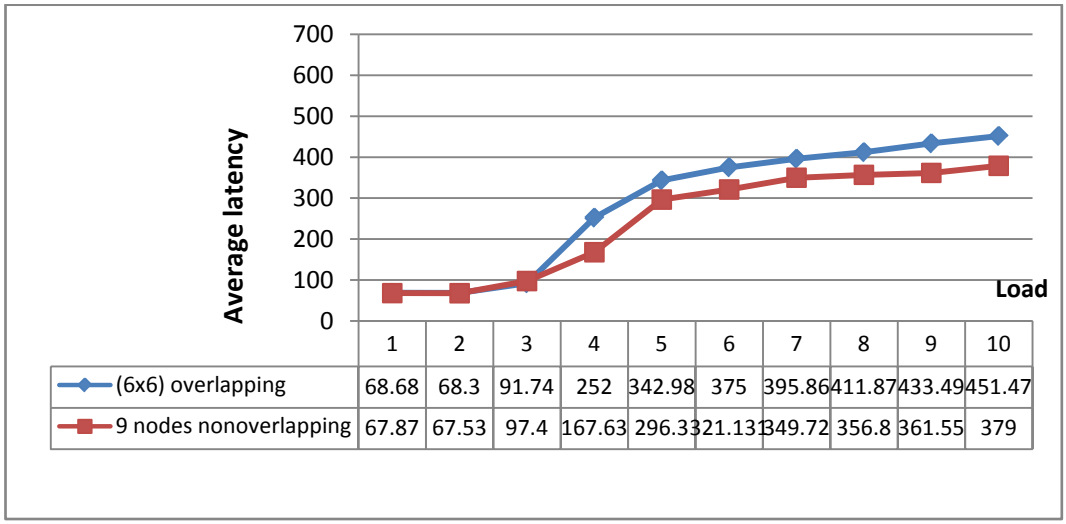
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
1% fault (I node 1 link)	64.41	64.13	64.83	67.35	90	122	178.9	203.2	221.486	251.69
5% fault (4 node 8 link)	68.74	68.86	117.2	187.7	218.5	231.75	248.7	280	296.118	331.47
7% fault (4 node 16 link)	77	189.321	299.6	352.4	370	368.558	378.588	400.49	425.3	456.4

Figure 33 Average latency of SB with 4 VCs under random faults



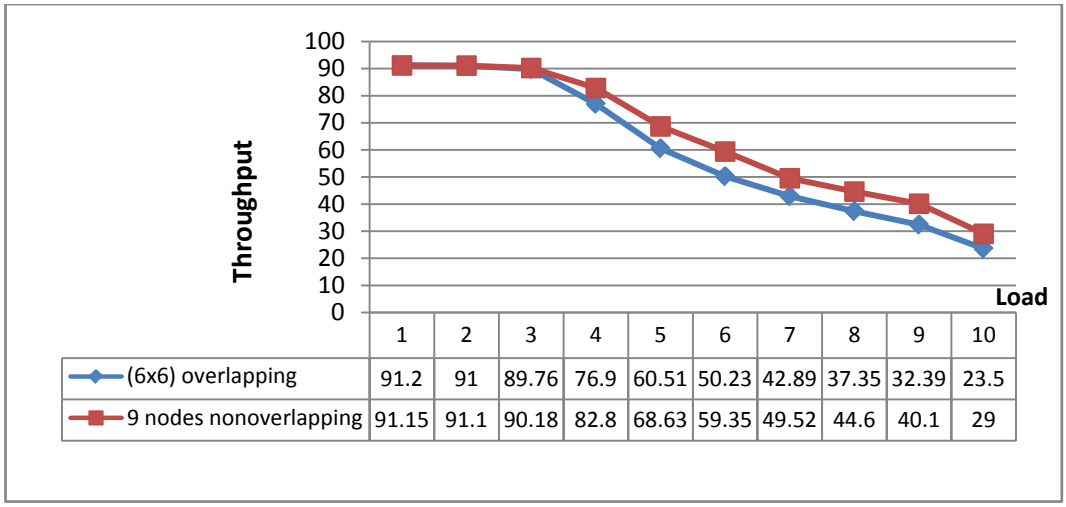
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
1% fault (I node 1 link)	91.16	91.1	91	90.99	89.87	86.4	78.5	69.7	61.74	45.67
5% fault (4 node 8 link)	91.15	91.12	86.3	75	65.4	58.9	54.45	49	47.57	31.73
7% fault (4 node 16 link)	91.16	85.28	69	55.74	47.87	42.1	37.93	34	30.07	21.8

Figure 34 Throughput of SB with 4 VCs under random faults



	1	2	3	4	5	6	7	8	9	10
(6x6) overlapping	68.68	68.3	91.74	252	342.98	375	395.86	411.87	433.49	451.47
9 nodes nonoverlapping	67.87	67.53	97.4	167.63	296.33	321.13	349.72	356.8	361.55	379

Figure 35 Average latency of SB with 4 VCs under local faults



	1	2	3	4	5	6	7	8	9	10
(6x6) overlapping	91.2	91	89.76	76.9	60.51	50.23	42.89	37.35	32.39	23.5
9 nodes nonoverlapping	91.15	91.1	90.18	82.8	68.63	59.35	49.52	44.6	40.1	29

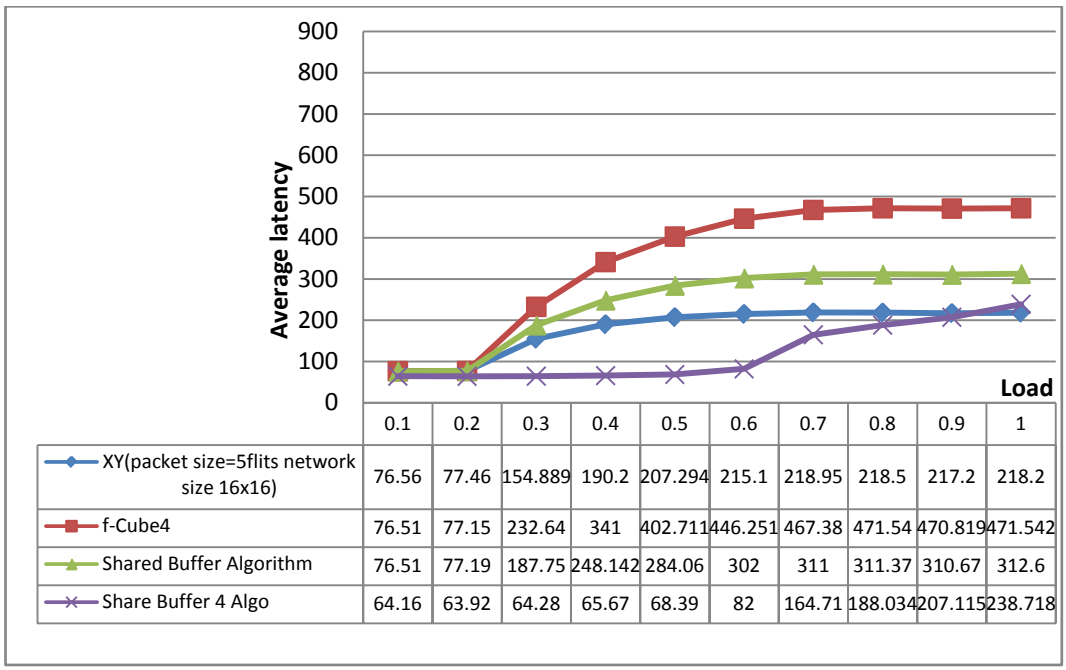
Figure 36 Throughput of SB with 4 VCs under local faults

However, in Figure 35 we can observe that the latency of messages in SB4 algorithm under overlapping f-rings reaches up the latency of SB algorithm. Since in these parts where the f-rings overlap the only one physical channel is given as an option, using two VCs per physical channel increases the contention for that particular channel. This is due to additional shared buffers that also content for resources, introducing the problem of multiplexing of VCs over one physical channel. That may be the reason of an increase in message latency. But the throughput is still better compared to f-Cube4 and SB algorithm since additional VCs releases the blocked messages.

4.4 General view on algorithms

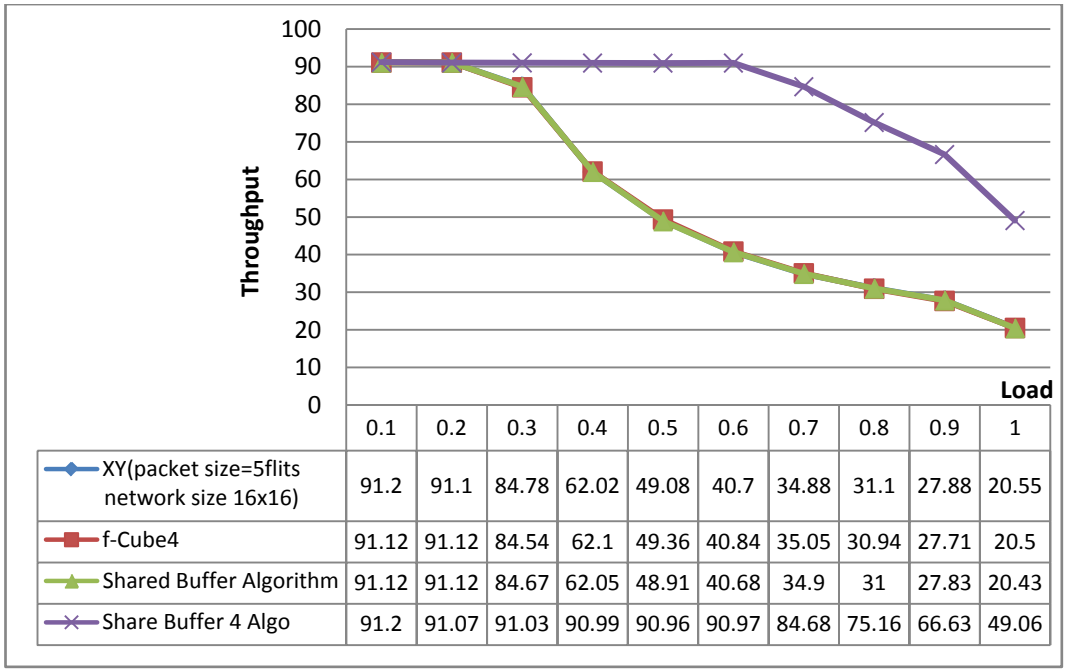
Here we investigate all these algorithms under fault-free conditions. XY-routing algorithm is also included. It is desirable for a fault-tolerant algorithm to be fast and require small amount of buffers. Since many fault tolerant algorithms require a big number of VCs in order to preserve deadlock-free operations, in the fault-free cases their performance is not good. The general effort in implementing fault-

tolerance is to keep the speed and compactness of deterministic algorithms (XY-algorithm), while being able to tolerate any number of faults with any kind of shapes.



	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
XY(packet size=5flits network size 16x16)	76.56	77.46	154.889	190.2	207.294	215.1	218.95	218.5	217.2	218.2
f-Cube4	76.51	77.15	232.64	341	402.711	446.251	467.38	471.54	470.819	471.542
Shared Buffer Algorithm	76.51	77.19	187.75	248.142	284.06	302	311	311.37	310.67	312.6
Share Buffer 4 Algo	64.16	63.92	64.28	65.67	68.39	82	164.71	188.034	207.115	238.718

Figure 37 Average latency of algorithms under fault free conditions



	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
XY(packet size=5flits network size 16x16)	91.2	91.1	84.78	62.02	49.08	40.7	34.88	31.1	27.88	20.55
f-Cube4	91.12	91.12	84.54	62.1	49.36	40.84	35.05	30.94	27.71	20.5
Shared Buffer Algorithm	91.12	91.12	84.67	62.05	48.91	40.68	34.9	31	27.83	20.43
Share Buffer 4 Algo	91.2	91.07	91.03	90.99	90.96	90.97	84.68	75.16	66.63	49.06

Figure 38 Throughput of algorithms under fault free conditions

Figures 37 and 38 illustrate the performance achieved by algorithms under fault-free conditions. The throughput is the same for XY, f-Cube4 and SB algorithms. The reason is that the router in all three cases sends equal number of flits every cycle. However the latencies differ. The highest latency is achieved by f-Cube4 algorithm. As it was explained before, the reason is the usage of more VCs per physical channels, while providing only with one choice for VC request according to message type. The lowest message latency and highest throughput is achieved by SB that uses 4 VCs. That is because it provides with alternative VCs, thus releasing blocked messages in every router.

Figures 39 and 40 present the performance of algorithms in the presence of single block fault region ranging in size from 4 to 20 failed nodes. Some of them form single f-ring and some of them form overlapping f-rings. Message injection rates are chosen from the results obtained from the performance evaluation under fault-free conditions that is illustrated in Figures 37 and 38. Since at load 0.2 the algorithms perform below saturation point, by inducing faulty nodes we can observe the degradation level of a network. Load 0.6 is chosen because of SB4 algorithm which saturates only when the load exceeds 0.6 in fault-free network.

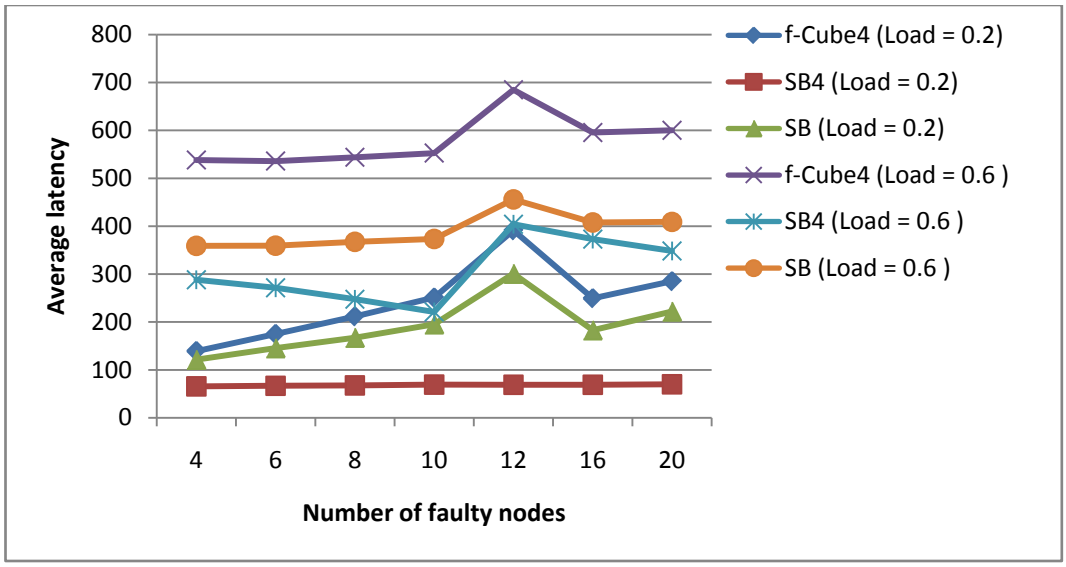


Figure 39 Average latency as a function of node failures

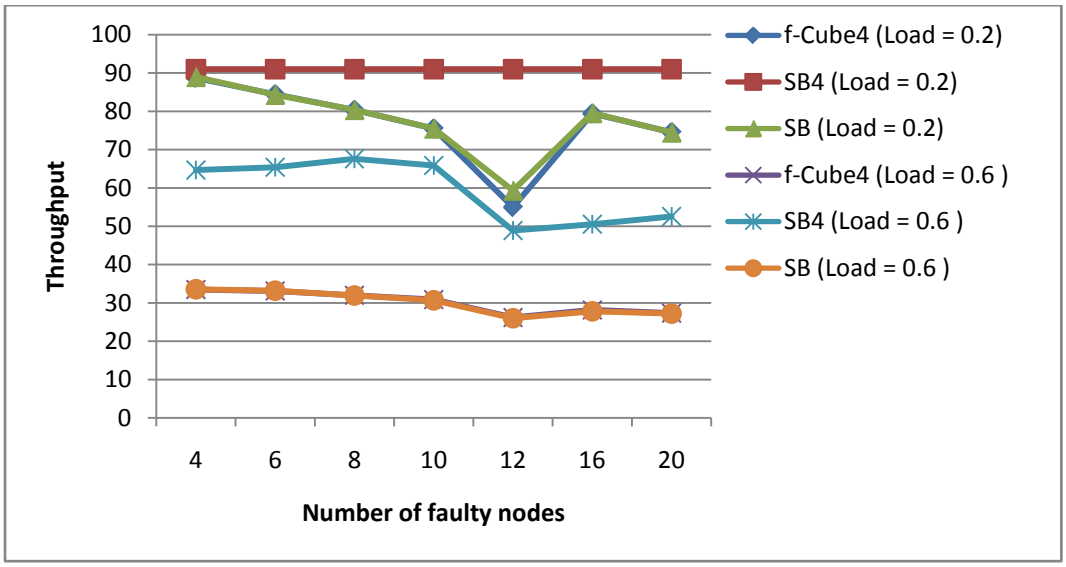


Figure 40 Throughput as a function of node failures

From the figures it can be seen that the SB4 algorithm performs well when the applied load is 0.2. It does not degrade even after injection of 20 faulty nodes. This is due to the saturation point of this algorithm which is far above 0.2.

We observed that the shape of an f-ring greatly influences the performance of the network. The performance degrades when the multiple faults mostly span *x*-axis rather than *y*-axis, since WE and EW messages can turn into NS and SN messages thus encounter a faulty block twice. Besides, NS and SN messages have to make long path in order to reach their destination after being misrouted. That is the reason of unstability in lines shown in Figures 39 and 40, as we are changing the shape of a faulty region by adding faulty nodes either on *x*-axis or *y*-axis. It is also observed that a single large faulty-block degrades the network performance greatly rather than the overlapping f-ring which consists of smaller faulty set. The peak that occurred in a graph is a non-overlapping f-ring containing 12 nodes and mostly spans *x*-axis, thus roughly increasing message latency and decreasing network throughput. The other faulty regions are taken either as overlapping f-rings each containing smaller amount of faulty nodes or non-overlapping f-ring spanning *y*-axis.

4.4.1 Efficiency level of buffer utilization

As it was mention before f-Cube4 algorithm significantly underutilizes VCs. In order to understand the buffer utilization we put counters into every VC of a router. The router was chosen from the middle of a network, which is the most overloaded part of a network [18]. Since we are considering the worst case the applied load was maximum, i.e. 1. In a fault-free network f-Cube4 utilizes only 40% of given VCs. SB and SB4 algorithms make use of 60% of the given VCs.

To evaluate the buffer utilization we put the overlapping f-ring into the middle of a network and applied the maximum load. However the VC utilization of messages depends on the position of a router. That's why we put counters into VC's of several routers: to the routers that are located on the shared path where two f-rings overlap, to the routers on the West, South, East and North of an f-ring.

On the shared path between two f-rings the high buffer utilization is shown by SB4 algorithm that periodically allocates 80% VCs placed in routers, while SB and f-

Cube4 algorithm make use of 70% and 45% respectively. North and South parts of an f-ring showed to be the most overloaded parts of an f-ring and hence f-Cube4 utilizes up to 55% VCs while SB and SB4 utilizes 90%.

SB algorithm is seen to be more efficient in terms of buffer utilization, since it efficiently uses these buffers that are given, dynamically allocating them according to its needs. On the other hand, f-Cube4 algorithm rarely utilizes even the half of the given VCs.

4.4.2 Comparison of algorithms

Both SB and f-Cube4 algorithms have the same routing concept. However, f-Cube4 provides every physical channel with four VCs in order to be able to tolerate overlapping f-rings. But many of these VCs are hardly utilized even at maximum load. Besides, additional VCs are never utilized under fault-free conditions. We decided to look more optimistically and consider that the VCs associated with faulty link are not faulty and make use of these idle VCs around the fault region. This requires every VC to be seen by every input controller and hence, increases the wiring complexity in a router. However, we consider that the increase in wiring complexity is tolerable since we reduce the number of VCs and utilize them at full capacity. The same performance can be achieved with small amount of VCs. The simulation results show that the performance improves significantly when adding two more VCs to SB algorithm. In addition to improvements in performance SB4 algorithm fully utilizes VCs at hand, thus rarely keeping them idle. Table 1 shows the similarities and differences of algorithms. The minimum and maximum throughput and average latency of algorithms under various conditions are given (Teble 1). Here, minimum latency and throughput is the performance results obtained with minimum load, while the maximum throughput and latency indicate the results obtained when the applied load is the maximum. Maximum load is the maximum sustainable throughput when a network is loaded with uniform random traffic. The table shows the general view on performances of algorithms. SB4 algorithm degrades much slower than the other

two algorithms. Besides, SB with two VCs has smaller average latency in compared to f-Cube4 algorithm.

In [34] the simulation of f-Cube4 was conducted by giving the physical channel arbitrary number of VC, up to eight, that were implemented as a free pool. However, the free pools belong only to one physical channel, which means that the other links cannot make use of these VCs that belong to faulty links. We do not implement a router with such huge amount of VCs per physical channel, since our goal is to decrease buffer size required for every router.

Algorithms	f-Cube4	SB	SB4
Fault model	Convex (f-rings, f-chains, overlapping f-rings)	Convex (f-rings, f-chains, overlapping f-rings)	Convex (f-rings, f-chains, overlapping f-rings)
Routing technique	XY-routing	XY-routing	XY-routing
Buffer utilization	40%-55%	60%-90%	60%-90%
Min-max throughput under fault free condition (in percentage)	20.5%-91.12%	20.43%-91.12%	49%-91.2%
Min-max average latency under fault free condition (in cycles)	76.5-471.5	76.5-312.6	64.2-238.7
Min-max throughput under random faults (7% faults, in percentage)	9.5%-73.7%	9.5%-73.7%	21.8%-91.15%
Min-max avg. latency under random faults (7% faults, in cycles)	377.5-777	291.7-528.9	77-456
Min-max throughput under localized faults (10% faults, in percentage)	14.2%-91.2%	13.97%-91.2%	23.5%-91.2%
Min-max avg. latency under localized faults (10% faults, in cycles)	94.7-633.7	93.9-424.9	68.7-451.5

Table 1 Comparison of fault-tolerant algorithms

As our algorithm is dependent on VCs of failed links, it may look like we are not considering the buffer failures. However, fault-tolerant algorithms are generally proposed to be used as escape channels for adaptive algorithms. Since adaptive algorithms also require a big number of VCs the link failure gives a lot more shared buffers than the fault-tolerant algorithm itself. It is not likely that all these buffers will fail at the same time.

CHAPTER 5

CONCLUSION AND FUTURE WORK

In this thesis we mainly concentrated on fault-tolerant routing in NoC. We implemented the well known fault-tolerant f-Cube4 algorithm, which considers edge buffers. When the edge buffers are implemented in a router only one input controller can see these VC that are incident on it. The failure of a physical link automatically disables VCs incident on it. However fault-tolerance requires large buffer sizes, thus a link failure disables large amount of buffers leading to buffer underutilization. This contradicts with the requirements of NoC that limits the buffer utilization. As a solution we proposed to implement VCs as a central pool in the middle of a router, so that every input controller can see them. Thus VCs are dynamically allocated when the link to which the VC associates is faulty. Besides, we proposed our own algorithm, SB, which requires only 2 VCs per physical channel under fault-free conditions and dynamically allocates idle VCs when the fault occurs. Simulation results showed that two VCs are enough to escape f-rings, f-chains and overlapping-rings of rectangular shape.

The implementation of VC as a central pool significantly improves the buffer utilization. While f-Cube4 algorithm uses only 55% of given buffers SB algorithm uses up to 90%. With the right management of buffer utilization the performance of a network can be significantly improved. However, the implementation of central buffers may increase the wiring complexity in a router. We consider that

central buffers can be implemented in those routers where small amount of VCs are needed, since the wiring complexity increases with the number of VCs.

In order to be able to tolerate faults our algorithm requires at least 2 VCs to be spare so that the router can share them with two neighbors from the other dimension. Besides, SB algorithm still marks non-faulty nodes as faulty in order to get f-rings of rectangular shape.

The concave (non-rectangular) faulty blocks are left as a future work, since they are more complicated and need extensive work. All our simulations use Poisson-distributed injection rate and destinations are uniformly distributed, since no other realistic traffic models were proposed. Investigation of realistic traffic models for NoC is still an open problem and could be a good area of research. It is not easy, however, since it requires to be deepened on specifics of particular applications.

REFERENCES

[1] **A. Hemani, A. Jantsh, S. Kumar, A. Postula, J. Oberg, M. Millberg, and D. Lindqvist**. *Network on a Chip: An architecture for billion transistor era*. In Proceeding of the IEEE NorChip Conference, 166--173. 2000

[2] **J. Henkel, W. Wolf, and S. Chakradhar**. *On-chip networks: a scalable, communication-centric embedded system design paradigm*. Proceedings of 17th International Conference on VLSI Design. 2004, pp. 845-851.

[3] **S. Lee, K. Lee, S. Song, and H. Yoo**. *Packet-switched on-chip interconnection network for system-on-chip applications*. IEEE Transaction on Circuits and Systems, 52(6):308--312, 2005.

[4] **P. P. Pande, G. D. Micheli, C. Grecu, A. Ivanov, and R. Saleh**. *Design, synthesis, and test of networks on chips*. from Networks on Chip. s.l. : Kluwer Academic Publishers, 2003.

[5] **L. Benini and G. D. Micheli**. *Networks on chips: a new SoC paradigm*. IEEE Computer. January 2002, Vol. 35, 1, pp. 70-78.

[6] **R. Saleh**. *An approach that will NoC your SoCs off! Design & Test of Computers,2005,22(5):488-488.*

[7] **W. J. Dally and B. Towles**. *Route packets not wires: on-chip interconnection networks*. Proceedings. Design Automation Conference, page 684- 689, 2001.

[8] **M. E. Kreutz, L. Carro, C. A. Zaferino and A. A. Susin**. *Communication architectures for System-on-Chip*. Proceedings of the 14th symposium on Integrated circuits and systems design, p.14, September 10-15, 2001

[9] **R. Marculescu, U. Y. Ogras and N. H. Zamora**. *Computation and communication refinement for multiprocessor SoC design:a system-level perspective*. in ACM Trans. on Design Automation of Electronic Systems, Special Issue on Novel Paradigms in System-Level Design, Vol.11, No.3, pp. 564-592, July, 2006.

[10] **J. Hu and R. Marculescu**. *Energy-and performance-aware mapping for regular NoC architectures*. IEEE transactions on computer aided design of integrated circuits and systems. vol 24; numb 4, pages 551-562, 2005

[11] **U. Y. Ogras, J. Hu and R. Marculescu**. *Key research problems in NoC design: a holistic perspective*. Third IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, 2005. CODES+ISSS '05. Sept. 2005, pp. 69-74.

[12] **L.M. Ni and P. K. McKinley**. *A survey of wormhole routing techniques in direct networks*. IEEE Computer 26 (2): 62-76. 1993

[13] **L. Benini, and D. Bertozzi**. *Network-on-chip architectures and design methods*. IEE Proceedings - Computers and Digital Techniques. March 2005, Vol. 152, 2, pp. 261-272.

[14] **S. Kumar, A. Jantsch, J. P. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrja, A. Hemani.** *A network on chip architecture and design methodology*. Proceedings of IEEE Computer Society Annual Symposium on VLSI, 2002. April 25-26, 2002, pp. 105-112.

[15] **S. Lee, K. Lee and H. Yoo**. *Analysis and implementation of practical, cost-effective networks on chips*. Design & Test of Computers, IEEE, 422-433 2005

[16] **F. Karim et al.** *An Interconnect Architecture for Networking Systems on Chips*. IEEE Micro, vol. 22, no. 5, pp. 36-45, Sept./Oct. 2002.

[17] **M. Coppola et al**. *OCCN: A network on chip modeling and simulation framework.* Proc. Design, Automation and Test in Europe, IEEE CS Press, 2004, pp. 174-179.

[18] **J. Duato, S. Yalamanchili, L. Ni.** *Interconnection Networks: an Engineering Approach.* s.l. : Morgan Kauffman, 2002.

[19] **W.J. Dally and C.L. Seitz.** *The Torus Routing Chip*. Technical Report 5208:TR: 86, Computer Science Dept., California Inst. of Technology, pp. 1-19, 1986.

[20] **P. Guerrier and A. Greiner**. *A Generic Architecture for On-Chip Packet-Switched Interconnections*. Proc. Design and Test in Europe (DATE), pp. 250-256, Mar. 2000.

[21] **P.P. Pande, C. Grecu, A. Ivanov, and R. Saleh.** *Design of a Switch for Network on Chip Applications*. Proc. Int'l Symp. Circuits and Systems (ISCAS), vol. 5, pp. 217-220, May 2003.

[22] **P. P. Pandeet al.** *Performance evaluation and design trade-offs for network-on-chip interconnect architectures*. *IEEE Transactions on Computers.* August 2005, Vol. 54, 8, pp. 1025-1040.

[23] **R. I. Greenberg and H. Oh**. *Universal wormhole routing.* IEEE Transactions on Parallel and Distributed Systems, vol.8, No.3, March 1997, pp. 254-262

[24] **P. P. Pande, C. Grecu, A. Ivanov and R. Saleh**. *High-throughput switch-based interconnect for future SoCs*. Proceedings of 3rd IEEE International Workshop on System-on-Chip for Real-Time Applications, June 30-July 2, 2003, Calgary, Canada.

[25] **W. J. Dally.** *Virtual-channel flow control*. IEEE Transactions on Parallel and Distributed Systems. March 1992, Vol. 3, 2, pp. 194-204.

[26] **C. Busch, M. Herlihy and R. Wattenhofer.** *Routing without Flow Control.* ACM Symposium on Parallel Algorithms and Architectures 2001

[27] **L.G. Valiant; G. J. Brebner**. *Universal schemes for parallel communication.* Proc. 13th ACM. Symp. on Theory of Computing, Milwaukee, IL, pp.263-277, May 11-13, 1981

[28] **T. Nesson; S. L. Johnson**. *ROMM Routing on Mesh and Torus Networks.* Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures, p.275-287, Santa Barbara, California, United States , June 24-26, 1995

[29] **Andrew A. Chien and Jae H. Kim**. *Planar-adaptive routing: low-cost adaptive networks for multiprocessors.* Journal of the ACM (JACM),Volume 42 , Issue 1 , pages 91 - 123 , (January 1995)

[30] **W.J. Dally and H.Aoki**. *Adaptive routing Using Virtual Channels*. MIT Laboratory for Computer Science. 1990. Technical report.

[31] **C. J. Glass and L. M. Ni.** *The turn model for adaptive routing.* Proceedings of the 19th International Symposium on Computer Architecture, pp. 278–287, May 1992.

[32] **M. S. Chien and K. G. Shin**. *Adaptive fault-tolerant routing in hypercube multicomputers*. IEEE Transactions on Computers, vol. C-39, no.12, pp. 1406-1416, December 1990.

[33] **S. Konstantinidou and L. Snyder**. *The chaos router: A practical application of randomization in network routing*. Proc. 2nd Annual Symposium on Parallel Algorithms and Architectures, pp. 21-31, July 1990.

[34] **R.V. Boppana and S. Chalasani.** *Fault-tolerant wormhole routing algorithms for mesh networks*. IEEE Transactions on Computers, July 1995, pp. 848-864.

[35] **S. Park, J. Youn and B.Bose**. *Fault-tolerant wormhole routing algorithms in meshes in the presence of concave faults*. In Proceedings of IPDPS. 2000, 633-633.

[36] **J. Zhou and F. C. M. Lau**. *Adaptive fault-tolerant wormhole routing in 2D meshes*. Proc. 15th Int'l Parallel and Distributed Processing Symp. (IPDPS 2001), p. 56, 2001.

[37] **E. Oh, J. Kim and H. Lee**. *Fault-tolerant routing in mesh-connected 2D Tori*. Proc. The International Conference on Computational Science (ICCS '03), 527-536 2003

[38] **J.Shih**. *Adaptive fault-tolerant wormhole routing algorithms for hypercube and mesh interconnection netwotks*. Proc. 11th Int'l Parallel Processing Symp., pp. 333-340, Apr. 1997.

[39] **C. J. Glass and L. M. Ni.** *Fault-tolerant wormhole routing in meshes*. 3rd Annual International Symposium on Fault-Tolerant Computing, 1993, June 22-24.

[40] **Y. Suh, B. V. Dao, J. Duato and S. Yalamanchili.** *Software based fault-tolerant oblivious routing in pipelined networks*. IEEE Transactions on Reliability, September 2005, pp. 449-458.

[41] **W. J. Dally et al..** *Architecture and implementation of the reliable router*. Proceedings of Hot Interconnects Symposium II, August 1994.

[42] **P. T. Gaughan et al.** *Distributed, deadlock-free routing in faulty, pipelined, direct interconnection networks*. IEEE Transactions on Computers, vol. 45, June 1996, pp. 651-665.

[43] **J. H. Kim, Z. Liu and A. A. Chien.** *Compressionless routing: A framework for adaptive and fault-tolerant routing*. Proceedings of the 21st International Symposium on Computer Architecture, April 1994, pp. 289-300.

[44] **Vassos Soteriou, Hangsheng Wang, Li-Shiuan Peh**. *A Statistical Traffic Model for On-Chip Interconnection Networks.* International Conference on Measurement and Simulation of Computer and Telecommunication Systems (MASCOTS '06), September, 2006.

[45] **S. Mahadevan et al**. *A network traffic generator model for fast network on-chip simulation*. In Proc. of the Design, Automation and Test in Europe Conference and Exhibition (DATE’05) Vol. II, pp. 780–785, March 2005.

[46] **G. V. Varatkar and R. Marculescu**. *On-chip traffic modeling and synthesis for MPEG-2 video applications.* IEEE Transactions of Very Large Scale Integration (VLSI) Systems, Vol. 12, No. 1, Jan. 2004.

[47] **Zhonghai Lu, A. Jantsch**. *Traffic configuration for evaluating networks on chips.* Fifth international Workshop on System-on-Chip for Real-Time Applications (IWSOC). 20-24 July 2005. pp. 535 - 540

[48] **Kevin Fall, Kannan Varadhan.** *The ns Manual.* The VINT Project, June 20, 2001. Also available at *http://www.isi.edu/nsnam/ns/ns-documentation.html*

[49] **Y. Sun, S. Kumar and A. Jantsch.** *Simulation and evaluation for a network on chip architecture using ns-2.* Proc. Of 20th IEEE Norchip Conference, November 2002

[50] **A. Hegedus, G. M. Maggio and L. Kocarev.** *A ns-2 simulator utilizing chaotic maps for network-on-chip traffic analysis.* IEEE International Symposium on Circuits and Systems, May 2005, pp. 3375-3378.

[51] **M. Ali, M. Welzl, A. Adnan and F. Nadeem.** *Using the ns-2 simulator for evaluating network on chips (NoC).* IEEE International Conference on Emerging Technologies, November 2006.

[52] **L. S. Peh and W. J. Dally.** *Flit-reservation flow control*. Proceedings of High Performance Computer Architecture. January 2000, pp. 73–84.

[53] **C. A. Zaferino, F. Santo and A. A. Susin**. *ParIS: a parameterizable interconnect switch for networks-on-chip.* Proceedings of the 17th symposium on Integrated circuits and system design, September 07-11, 2004, Pernambuco, Brazil.

[54] **R. Leveugle, T. Michael, and G. Sauicier.** *Design of microprocessors with built-in on-line test.* 20th Annual International Symposium. Fault-Tolerant Computing, pp. 450-456, 1990.