Ville Rantala

On Dynamic Monitoring Methods for Networks-on-Chip

Turku Centre *for* Computer Science

TUCS Dissertations
No 154, December 2012

On Dynamic Monitoring Methods for Networks-on-Chip

Ville Rantala

To be presented, with the permission of the Faculty of Mathematics and Natural Sciences of the University of Turku, for public criticism in Auditorium Lambda on December 12, 2012, at 12 noon.

University of Turku
Department of Information Technology
20014 Turun yliopisto

2012

Supervisors

Adjunct Professor Juha Plosila
Department of Information Technology
University of Turku
20014 Turun yliopisto
Finland

Adjunct Professor Pasi Liljeberg
Department of Information Technology
University of Turku
20014 Turun yliopisto
Finland

Reviewers

Professor Gert Jervan
Department of Computer Engineering
Tallinn University of Technology
Raja 15, 12618 Tallinn
Estonia

Associate Professor Zhonghai Lu
Department of Electronic Systems
Royal Institute of Technology
Forum 120, 16440 Kista
Sweden

Opponent

Professor Timo D. Hämäläinen
Department of Computer Systems
Tampere University of Technology
33101 Tampere
Finland

ISBN 978-952-12-2822-3 (PRINT)
ISBN 978-952-12-2823-0 (PDF)
ISSN 1239-1883

Abstract

Rapid ongoing evolution of multiprocessors will lead to systems with hundreds of processing cores integrated in a single chip. An emerging challenge is the implementation of reliable and efficient interconnection between these cores as well as other components in the systems. Network-on-Chip is an interconnection approach which is intended to solve the performance bottleneck caused by traditional, poorly scalable communication structures such as buses. However, a large on-chip network involves issues related to congestion problems and system control, for instance. Additionally, faults can cause problems in multiprocessor systems. These faults can be transient faults, permanent manufacturing faults, or they can appear due to aging. To solve the emerging traffic management, controllability issues and to maintain system operation regardless of faults a monitoring system is needed. The monitoring system should be dynamically applicable to various purposes and it should fully cover the system under observation. In a large multiprocessor the distances between components can be relatively long. Therefore, the system should be designed so that the amount of energy-inefficient long-distance communication is minimized.

This thesis presents a dynamically clustered distributed monitoring structure. The monitoring is distributed so that no centralized control is required for basic tasks such as traffic management and task mapping. To enable extensive analysis of different Network-on-Chip architectures, an in-house SystemC based simulation environment was implemented. It allows transaction level analysis without time consuming circuit level implementations during early design phases of novel architectures and features.

The presented analysis shows that the dynamically clustered monitoring structure can be efficiently utilized for traffic management in faulty and congested Network-on-Chip-based multiprocessor systems. The monitoring structure can be also successfully applied for task mapping purposes. Furthermore, the analysis shows that the presented in-house simulation environment is flexible and practical tool for extensive Network-on-Chip architecture analysis.

Acknowledgements

The research, presented in this thesis, was carried out during years 2008-2012 at the Department of Information Technology, University of Turku. The work was funded by Academy of Finland and Turku Centre for Computer Science (TUCS) Graduate School, with financial support from the Nokia Foundation and the Finnish Foundation for Technology Promotion. This thesis would not have been done without the support from these organizations.

I would like to thank my supervisors Juha Plosila and Pasi Liljeberg for steering me through this process which practically started in 2006 when I worked as a summer trainee at their laboratory. I wish to express my gratitude to Teijo Lehtonen for co-authoring several papers and guiding me through these years. I would also like to thank reviewers Gert Jervan and Zhonghai Lu whose comments improved this thesis significantly. I also express my gratitude to professor Timo D. Hämäläinen for agreeing to act as the opponent.

Warm thanks go also to the coffee room crews both at the Department of the Information Technology as well as at the Business Innovation Development Unit, especially to Sami Nuuttila for solving each and every computer related problem I was able to discover.

Finally I would like to thank my parents Anne and Pasi for encouragement and support towards whatever I have been doing.

Turku, November 5, 2012

Ville Rantala

Contents

List of Figures

List of Tables

List of Abbreviations

ALOAS	Arbitration Look Ahead Scheme
BE	Best Effort
DCM	Dynamically Clustered Monitoring
DyAD	Dynamically Adaptive and Deterministic
E	East
FIFO	First In, First Out
Flit	FLow control digIT *or* FLow control unIT
GS	Guaranteed Service
GT	Guaranteed Thoughput
I/O	Input/Output
ID	Identifier
IP	Intellectual Property
IVAL	Improved VALiant's randomized algorithm
MPSoC	Multiprocessor System-on-Chip
N	North
NI	Network Interface
NoC	Network-On-Chip
PE	Processing Element
S	South
SoC	System-On-Chip
TDM	Time Division Multiplexing
TLM	Transaction-Level Modeling
TTL	Time To Live
VHDL	VHSIC Hardware Description Language
VHSIC	Very-High-Speed Integrated Circuit
W	West

Chapter 1

Introduction

Network-on-Chip (NoC) is a promising interconnection paradigm for current and future high-performance Systems-on-Chip (SoC) [9, 16, 49]. These SoCs can include hundreds of separate processor cores, memories and other Intellectual Property (IP) blocks, and the number of these components is increasing. Communication between these components is challenging. It can form a performance bottleneck, and therefore, largely determine the maximum performance and capacity of the whole system. Traditional SoCs utilize shared-medium, bus-based communication infrastructures. Bus-based structures are adequate for a small number of components but are poorly scalable for larger systems. When the number of components increases each component has less and less communication capacity available, the shared-communication medium is able to handle only one transaction at a time. The bus structure is poorly scalable and therefore not suitable for high-performance multi-core or multiprocessor systems.

NoC is a nanoscale on-chip computer network which consists of routers and links. System components can be connected to the routers through network interfaces. On-chip networks can be constructed following different network topologies and they can utilize various routing algorithms to steer data from a source to a destination. The data can be transferred in the form of data packets. The main advantages of a network-based communication infrastructure are scalability and flexibility. The size of a system can be increased without blocking the communication infrastructure, which is a problem in traditional bus-based communication infrastructures. Scaling the size of the network up increases the network capacity and offers new additional communication paths while in bus-based structures up-scaling increases capacitance and rises possibility of congestion.

Two essential issues concerning high-performance NoCs are the congestion and faultiness of the involved resources. Highly loaded computational resources can generate a large amount of traffic which causes congestion to the network and in the worst case blocks the network. A typical issue is unbalanced communication resource utilization. While part of network is highly loaded there can still be other resources in idle state. Network topologies and routing algorithms can be designed to minimize these issues but they are not able to balance network utilization without network monitoring and

monitoring-based reconfiguration. The monitoring is also essential to detect faults and reconfigure the system to operate despite such faults.

Manufacturing of complex multiprocessor systems requires utilization of modern nanoscale integrated circuit fabrication processes. These complex integrated circuits can include manufacturing faults which prevent the utilization of components in the faulty areas. Aging effects, for instance electromigration, can compose additional faults to the system [62]. To improve system reliability and increase manufacturing yield these complex systems should have mechanisms to maintain the functionality even though there are faulty components in the system. These mechanisms should be able to detect inadequate functionality and reconfigure the system so that faulty parts can be isolated and the overall functionality is maintained with a reasonable performance.

A monitoring system is an essential part of system reconfiguration in the cases of high traffic load or faulty system conditions. It consists of probes and monitors and requires dedicated or shared communication medium to move data between probes, monitors and other network components. A monitoring system collects various status data from the system and delivers the observed data to the components of the system. The collected data can be used to reconfigure system operation and that way maintain and improve the functionality of the system. The usage of a monitoring system is not limited to avoidance and solving of traffic and fault related issues. The operation of a large and complex multi-core or multiprocessor system requires extensive knowledge of the status of different components so that the operation of the system can be controlled and optimized. The monitoring system can be used for statistics collection or task mapping purposes, for instance.

1.1 Objectives

The main objective of this work is to develop a scalable monitoring structure for high-performance NoC based multiprocessor systems which makes it possible to maintain system functionality under high computational and communication load and even though there are faulty components in the system. The monitoring system should collect the status information from the system and deliver it as widely as necessary to ensure system functionality. A goal is to implement the system as far as possible without centralized control so that maximal scalability and architectural fault tolerance can be obtained. Architectural fault tolerance refers to system architecture where faulty components have local influence on the functionality of a system but do not distract the overall functionality which is typically the case when a centralized controller becomes faulty. In a faultless network the maximal performance requires the balanced utilization of network resources. Therefore, the communication infrastructure should be able to monitor the network load and direct the traffic also through less loaded areas. Balanced resource utilization also balances the energy consumption and, furthermore, equalizes the heat dissipation. Energy consumption and heat dissipation are both essential issues in modern integrated circuits. Additionally, the Network-on-Chip monitoring system should be applicable for

system control purposes which include for instance task mapping methods.

To enable extensive analysis of Networks-on-Chip a simulation environment is needed. The simulator should operate on the transaction level without in-detail circuit-level implementations so that new ideas and approaches can be easily analyzed without time consuming circuit implementations. This simulation environment should be able to model large NoCs with different routing algorithms and monitoring structures.

1.2 Contributions

The main contributions of the thesis include:

- Dynamically clustered distributed monitoring structure which is scalable, flexible and where the monitoring infrastructure is evenly distributed over the monitored system. The dynamically clustered structure is presented in Chapter 4.
- SystemC based Network-on-Chip simulation environment which enables transaction level analysis of different Network-on-Chip architectures and their features. The simulation environment is presented in Chapter 5.
- Analysis of traffic management related features of the dynamically clustered monitoring structure. The analysis is studied in Chapter 6.
- Lightweight distributed task mapping method which is based on the dynamically clustered monitoring structure and where the cores are able to carry out task mapping autonomously without complete knowledge of the system status. The lightweight task mapping method is presented and studied in Chapter 7.

1.3 Publications

The results, presented in this thesis, are partly published previously in journal articles

- V. Rantala, P. Liljeberg and J. Plosila. Status Data and Communication Aspects in Dynamically Clustered Network-on-Chip Monitoring. In *Journal of Electrical and Computer Engineering*, 2012(2012), 2012.
- V. Rantala, T. Lehtonen, P. Liljeberg and J. Plosila. Analysis of Monitoring Structures for Network-on-Chip: a Distributed Approach. *IGI International Journal of Embedded and Real-Time Communication Systems (IJERTCS)*, 2(1):49–67, 2011.

and conference articles

- V. Rantala, T. Lehtonen, P. Liljeberg and J. Plosila. Analysis of Status Data Update in Dynamically Clustered Network-on-Chip Monitoring. In *1st International Conference on Pervasive and Embedded Computing and Communication Systems, PECCS 2011*, March 2011.

- V. Rantala, T. Lehtonen, P. Liljeberg and J. Plosila. Multi Network Interface Architectures for Fault Tolerant Network-on-Chip. In *International Symposium on Signals, Circuits and Systems (ISSCS '09)*, July 2009.

The results and discussion presented in Chapter 7 are new and they have not been published previously.

1.4 Organization

The thesis is organized as follows. The basics of Networks-on-Chip are presented in Chapter 2. It includes the descriptions of NoC components and topologies as well as presents typical problems regarding Networks-on-Chip. Monitoring is discussed in Chapter 3, including different monitoring structures and the purposes of monitoring. A dynamically clustered distributed monitoring approach for Networks-on-Chip is presented in Chapter 4. The chapter includes the concept level analysis of the dynamically clustered monitoring approach and discussion concerning routing and monitoring algorithms. Chapter 5 presents SystemC based transaction level Network-on-Chip simulation environment which is used to simulate and analyze the presented monitoring approach in chapters to follow. Issues concerning traffic management using dynamically clustered monitoring are presented and analyzed in Chapter 6. In Chapter 7 the dynamically clustered monitoring is applied for task mapping purposes. Issues related to task mapping using the presented monitoring system are discussed and the task mapping functionality demonstrated. Finally, conclusions are drawn and future work discussed in Chapter 8.

Chapter 2

Network-on-Chip

Network-on-Chip (NoC) is an approach to implement the interconnection of large and complex integrated circuits. Principles of Networks-on-Chip are basically similar to principles of any computer network. The main components are similar with the components of computer networks but implemented in nanoscale. Topology of the network determines how the components are organized and connected to each other. A significant part of a network implementation is a routing algorithm which defines along which path data are transferred from a source to a destination.

Typically, in Network-on-Chip data are transferred in packets. Each packet includes a header and a payload. The actual data are stored as a payload while the header includes important control information. The most important part of data in the header is an address of a destination but it can also include for instance an address of a sender, packet ID and some statistical data. Network flow control is a protocol which determines how individual packets are moved in the network. Besides these, types of the network traffic and basic problems are discussed in this chapter.

2.1 Components

Components of Networks-on-Chip include routers, links, network interfaces and cores. Additionally, an NoC can include probes and monitors. A *router* is the essential component of a network. It is in charge of that all the data are transferred from sources to correct destinations. The structure of a router is represented in Figure 2.1. There are several input and output ports, switch to connect input and output ports together and two logic blocks to control the switch. The core of the routing logic is a routing algorithm which decides to which output port the input port is connected when a packet arrives at the router. The arbitration logic decides from which input port a packet is received at any point of time. The arbitration logic unit listens to the input ports and takes care that data from all directions are accepted and forwarded. The arbitration algorithm can vary based on different prioritization requirements.

*Link*s are used to connect routers to each other. Basically, they are wires optionally

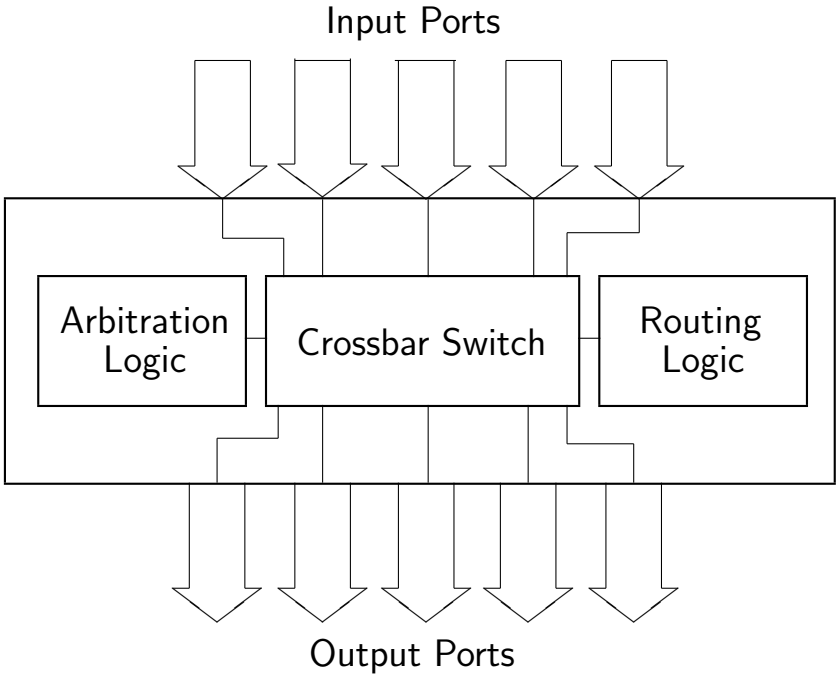


Figure 2.1: 5-port router.

with data buffering capabilities. If a transfer distance is relatively long, a link can have repeaters to amplify the transferred signal. Buffer registers, where packets are stored while waiting for transferring forward, are implemented in links or in routers.

The purpose of a Network-on-Chip is to connect the components of an integrated circuit to each other. These components are in this thesis called as *core*s. These components can be processors, memories or any intellectual property (IP) blocks.

In networks data are transferred in packets so the packets have to be generated before the data are sent to the network and also the packets have to be opened when they have been received from the network. These two operations are done in a network interface (NI) which, as its name indicates, is an interface between a core and a network. A network interface can be implemented as a part of a core and the core is able to connect to an interconnection network through it.

The connections of the basic components are represented in Figure 2.2. Additionally, a Network-on-Chip can include diagnostic components, *probe*s and *monitor*s. These components are used to collect diagnostic and statistic data from the system. A probe is a small component which is attached to an observed component. It delivers the observation data to a monitor which takes care of data exploitation. The monitoring data can be used for reconfiguration or statistical purposes, for instance.

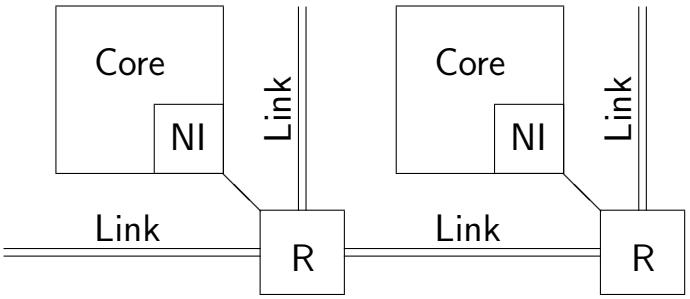


Figure 2.2: Basic components of a Network-on-Chip: router (R), link, core and network interface (NI) in a corner of a larger network.

2.2 Topologies

Topology defines the structure of a network. Interconnection networks can be categorized into four classes based on the network topology. These classes are shared-medium networks, direct networks, indirect networks and hybrid networks. In the shared-medium networks the transmission medium is shared by all components of the system. A bus is an example of a shared-medium network. Each device connected to the bus has a certain time slot when it is allowed to use the bus and only one component can use the shared medium at a time. Arbitration logic is used to control who is able to use the bus and for what duration. The major drawback of a shared-medium network is its lack of scalability. When new components are connected to a bus, the time slot for each component has to be shortened and that way the communication performance most likely decreases.

Direct networks are also known as point-to-point networks. In a direct network each component is directly connected to all the components it is communicating with. Therefore, any two components are not able to communicate with each other if they do not have a direct connection between each other. Shared-resource and direct networks are typically circuit switched networks which means that data are transferred without dividing it into packets. [51]

In a packet switched network, the raw data are divided into packets which are transferred in the network individually. Packet switched communication is utilized in indirect networks. An indirect network consists of routers which compose a direct network between them. Devices of the system are connected to routers through a network interface. [18] A hybrid network is a network which consists of a combination of different kinds of networks.

A network is non-blocking if it is able to fulfill all the requests that are offered to it. In a packet switched networks, this kind of network is also called as a non-interfering network. A non-interfering network can deliver all the packets in guaranteed time. [14] A basic network topology has one hierarchy layer where all the nodes are equal. An improved version of this network topology is a hierarchical network where a network is

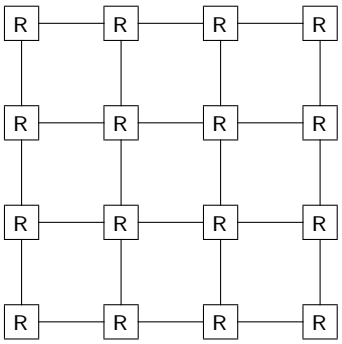


Figure 2.3: Mesh network.

divided into multiple hierarchy layers. Networks on the lower layer are called as the subnetworks of a higher layer.

2.2.1 Basic Topologies

Mesh and Torus Probably the most utilized Network-on-Chip topology is mesh, presented in Figure 2.3. A mesh network consists of `m` columns and `n` rows. In the mesh network, presented inf Figure 2.3, both `m` and `n` are four. The routers are located in the intersections of rows and column. Addresses of routers and resources can be easily defined as X- and Y-coordinates of a mesh. The strength of the mesh network is its simplicity. It can be easily placed on a chip without complicated routing of wires. However, weakness of a mesh topology is varying distance between routers. Average distance from center of a network is shorter than from routers at the edges of mesh. [14]

A torus network is an improved version of the basic mesh network. A simple torus network is a mesh where the heads of the columns are connected to the tails of the columns and the left sides of the rows are connected to the right sides of the rows. A torus network has better path diversity than a mesh network, which means that there are more alternative routes between two nodes of the network. The average distance between routers is constant on the contrary to the mesh. A torus network is shown in Figure 2.4. The chip implementation of a torus network is more complicated than the mesh network due to the links from an edge to another edge. These links can also require signal repeaters due to their relatively long length. [38]

Tree and Butterfly Topologies In a tree network routers are placed at different levels and cores are connected to the routers on the lowest level [11]. The routers on higher levels are called as ancestors of the routers below them. A fat-tree topology is illustrated in Figure 2.5. The cores can be connected to four routers at the bottom of the figure. In the fat-tree topology, the routers are connected to multiple ancestors which increases the path diversity in the network and offers multiple routes between cores.

A butterfly network is a duplicated tree network. It consists of two tree networks

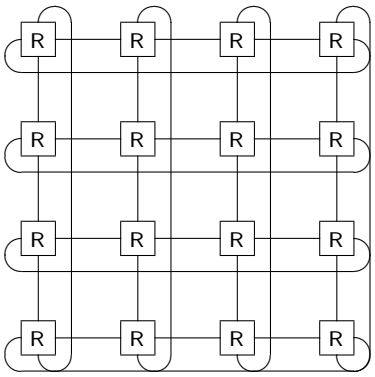


Figure 2.4: Torus network.

from which another is turned around and connected above the original tree network. This way cores can be connected not only to the routers at the bottom of the network but also to the routers on the top of the network. The highest ancestor routers are in the middle of the network. [28]

Polygon and Star Topologies The simplest polygon network is a circular network where packets travel in a loop from a router to another in uni- or bidirectional path. Network becomes more diverse when chords are added to the circle. A polygon network is fully connected if there is a direct link from every router to every other router in the network. A fully connected polygon network is presented in Figure 2.6.

A star network, which is represented in Figure 2.7, is another simple network topology. It consists of a central router in the middle of the star, and cores in the spikes of the star. The capacity requirements of the central router are quite large, because all the traffic between the spikes goes through the central router. That causes a remarkable possibility of congestion in the middle of the star.[60]

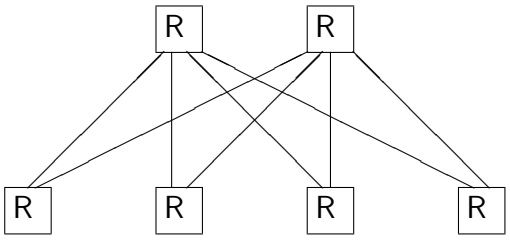


Figure 2.5: Fat-tree network.

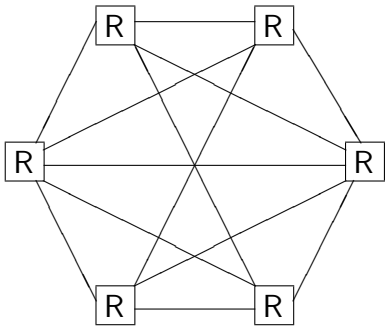


Figure 2.6: Fully connected polygon (hexagon) network.

2.2.2 Hierarchical Topologies

Hierarchical topologies are based on an idea that multiple networks are connected to each other with an upper level network, a global network. The originally separate networks, or subnetworks, are also called as the clusters of the network. This approach is useful in situations where most of the traffic is between components in a cluster but there is still a need to have a connection to the components located in the other clusters.

A hierarchical hybrid topology is a combination of two different network topologies. Subnetworks have different topology than the global network. A hierarchical hybrid network with local mesh networks and a global polygon network is represented in Figure 2.8.

2.2.3 Irregular Topologies

Irregular networks do not follow any regular topology. An irregular network has an application specific structure where routers are connected to the cores and to each other based on a known traffic pattern. Only the communication paths, which are required by the application, are implemented.

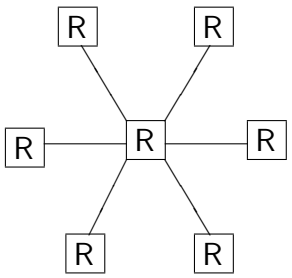


Figure 2.7: Star network.

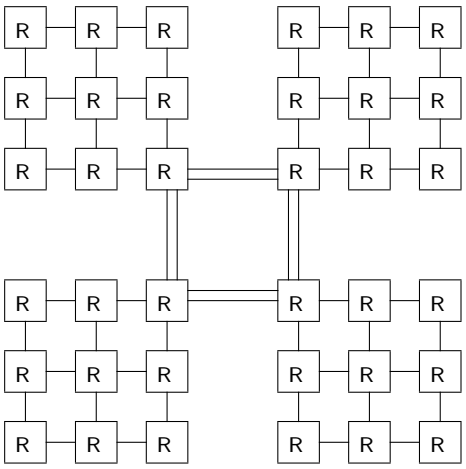


Figure 2.8: Hierarchical hybrid network with a global ring topology and local mesh topologies.

2.3 Network Traffic Classification

The traffic in Networks-on-Chip may be classified into two categories: guaranteed throughput (GT) traffic and best effort (BE) traffic. Guaranteed throughput is also called as guaranteed service (GS). When guaranteed throughput is utilized the specification of the system guarantees that some defined portion of sent data reach its destination in a given time frame. Guaranteed throughput works best with a routing algorithm that operates similarly as a circuit switched network, which means that the packets are sent from a sender to a receiver through a fixed path and this path is not used by other senders at the same time. All interference has to be minimized on the used path.

Best effort packets are transferred as trustworthy as possible. There are still no guarantees that best effort packets will ever reach the receiver. Latencies can vary and in the worst case packets can be lost for instance due to un-delivered packet removal mechanisms. Traffic in a basic packet switched network is mostly best effort traffic. In a packet switched network packets from a sender to a receiver may not move along the same path especially when adaptive routing algorithms are used (see Section 2.5.3). The packets from different senders move simultaneously in the network and share the network resources. [14] Therefore, fixed performance cannot be guaranteed but the network performance depends on the contemporary traffic pattern.

2.4 Network Flow Control

Network flow control determines how packets are transmitted inside a network. The flow control is not directly dependent on a routing algorithm so that usage of a certain routing algorithm does not necessarily require utilization of a certain flow control. However, some

algorithms may be designed to use some given flow control.

Store-and-Forward. Store-and-forward is the simplest network flow control. Packets move in one piece, and an entire packet has to be stored in the memory of a router before it can be forwarded to the next router. The buffer memory in a router has to be as large as the largest packet in the network. The latency is the combined time of receiving a packet and sending it forward. [6]

Virtual Cut-Through. Virtual cut-through is an improved version of the store-and-forward flow control. A router can begin to send a packet to the next router as soon as the next router gives a permission. The packet is stored in the router until the forwarding begins. Forwarding can be started before the whole packet is received and stored to the router. This network flow control needs as much buffer memory as the store-and-forward mode, because it is not guaranteed that the forwarding can begin before the whole packet has been received. However, average latencies can be decreased. [34]

Wormhole Routing. In wormhole routing packets are divided into small and equal sized flits (*FLow control digIT* or *FLow control unIT*). A first flit of a packet is routed similarly as packets in the virtual cut-through routing. After the first flit, the route is reserved to route the remaining flits of the packet. This route is called as a wormhole. The wormhole flow control requires less memory than the two other modes because only one flit has to be stored at a time to a router. Also the latency is smaller whereas a risk of deadlock is higher (see Section 2.6). [59]

The risk of deadlock can be reduced by using virtual channels, which means that multiple virtual channels are multiplexed to a physical port using time division multiplexing, for instance. The usage of virtual channels enhances the stability of a network and reduces the risk of congestion and network blockage [15]. Virtual channels overcome channel blocking problems by allowing other packets to use the channel resources regardless of blocking packets. Without the use of the virtual channels, one blocking packet reserves the whole channel until the packet is removed.

2.5 Routing Algorithms

Routing on Network-on-Chip is similar to routing on any computer network. A routing algorithm determines how the data are routed from a source to a destination.

Routing algorithms are divided into three categories: deterministic, stochastic and adaptive algorithms. Deterministic and stochastic algorithms are oblivious algorithms because they route packets without any information about fault and traffic conditions of the network. Deterministic algorithms route packets from a sender to a receiver always along the same route while stochastic routing is based on randomness and probabilities.

Adaptive algorithms reconfigure the routing based on the status of the network. The algorithm uses a monitoring method to be aware about traffic levels, congestion spots and faulty components in a network. Typical problems, such as deadlock and livelock, are discussed later in Section 2.6.

2.5.1 Deterministic Routing

Deterministic routing algorithms route packets every time from a certain point A to a certain point B along a fixed path. [18] In congestion free networks, deterministic algorithms are reliable and have low latency. They are well suitable for real-time systems because packets always reach the destination in their original order which eliminates the need for packet reordering. The latencies are also predictable as long as the network stays faultless and congestion free. In one of the simplest cases, each router has a routing table that includes routes to all other routers in the network and the routing decisions are simple table look-up operations.

Some of the deterministic algorithms are suitable for both regular and irregular networks. Algorithms, which are used in the irregular networks, have to be based on routing tables. When the structure of the irregular network changes, every router has to be updated.

Dimension Order Routing Dimension order routing is a typical minimal turn algorithm which is suitable for being utilized in mesh and torus networks. In minimal turn algorithms, packets are routed from a source to a destination using as few turns as possible. The dimension order algorithm determines to what direction packets are routed during every stage of the routing. [14, 43]

Probably the most used dimension order algorithm is an XY routing algorithm. It routes packets first in X- or horizontal direction to the correct column and then in Y- or vertical direction to the destination. The procedure of XY routing is illustrated in Figure 2.9. XY routing suits well in mesh and torus networks, where addresses of the routers are their XY-coordinates. Deterministic XY routing never ends up in deadlock or livelock. [17]

There are some problems in the traditional XY routing. Typically the traffic does not distribute evenly over the whole network but the routers in the middle of a network become more loaded than the routers at the edges of a network. Therefore, there is a need for algorithms which equalize the traffic load over the whole network.

Pseudo Adaptive XY Routing A pseudo adaptive XY routing algorithm has been proposed to work in a deterministic or an adaptive mode depending on the state of the network. The algorithm works in deterministic mode when the network is at most only slightly congested. When a network becomes blocked, the algorithm switches to the adaptive mode and starts to search routes that are not congested.

The pseudo adaptive XY routing works on mesh networks. Each port of a router has a small temporary storage buffer and a 2-bit status identifier called quantized load

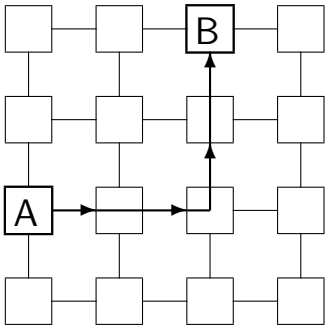


Figure 2.9: XY routing from router A to router B.

value. The identifier tells the other routers if the router is congested and cannot accept new packets.

A router assigns priorities to incoming packets when there are more than one incoming packet arriving simultaneously. Packets from north have the highest priority, then south, east and finally packets incoming from west have the lowest priority. While the traditional XY routing causes network loads more in the middle of the network than to the lateral areas, the pseudo adaptive algorithm divides the traffic better over the whole network. [17]

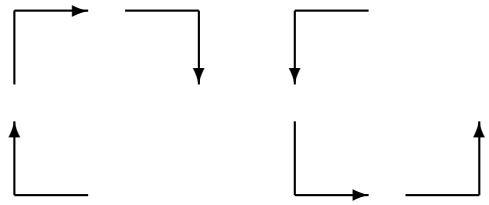
Turn Models Turn model algorithms determine turns which are and which are not allowed while routing packets through a network. Turn models are deadlock-free because the turn prevention eliminates the possibility to transmit packets in full circle. Three examples of turn model algorithms are presented below. [32]

1. West-first routing algorithm prevents all turns from any direction towards west. Therefore, the packets going west must be first transmitted as far towards west as necessary. Routing packets towards west is not possible later on.

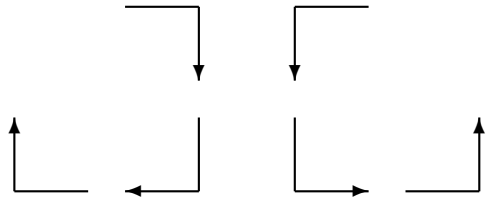
2. North-last routing algorithm disables turns away from north direction. Thus, the packets which need to be routed towards north, must be first moved in east-west-dimension and finally towards north to the destination.

3. Negative-first routing algorithm allows all other turns except turns from a positive direction to a negative direction. The network is considered as a coordinate system where the origin is in the bottom left corner and the positive directions are from bottom to top and from left to right. Packet routings in negative directions must be done before anything else.

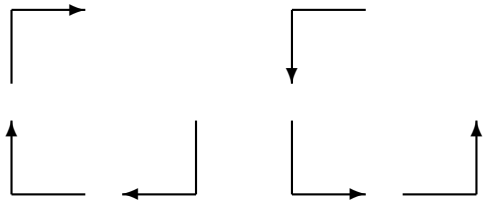
Allowed turns in these three turn model algorithms are illustrated in Figure 2.5.1.



(a) West-first



(b) North-last



(c) Negative-first

Figure 2.10: Allowed turns in turn model routing algorithms. The origin of network coordinates is located in the bottom left corner of the networks.

Shortest Path Routing Shortest path routing algorithms transfer packets always along the shortest path from a sender to a receiver. A distance vector routing algorithm is the most basic shortest path routing algorithm. Each router has a routing table that contains information about neighbor routers and all possible packet destinations. Routers exchange routing table information with each other and this way keep their own tables up to date. Routers route packets by calculating the shortest path based on their routing tables and then send packets forward. Distance vector routing is a simple method because each router does not have to know the structure of the whole network. [5]

Link state routing is somewhat more complex than the distance vector routing. The basic idea is the same as in the distance vector routing, but in the link state routing algorithm each router shares its routing table with every other router in the network. The link state routing presented for Network-on-Chip systems is a little bit customized version of the traditional distance vector routing. The routing tables covering the whole network are stored in memories of routers already during the manufacturing phase. Routers use their routing table updating mechanisms only if there are remarkable changes in the structure of the network or if some new faults emerge. [5]

Source Routing In source routing a sender makes all routing decisions concerning the complete routing path of a packet. The whole route is stored in the header of a packet before sending, and routers along the path do the routing following the predefined routing path.

Arbitration look ahead scheme (*ALOAS*) is a faster version of source routing. Information of a routing path has been supplied to routers along the path before the packets are even sent. Route information moves along a special channel that is reserved only for this purpose. [16, 35, 71]

Contention-free routing is an algorithm based on routing tables and time division multiplexing (TDM). Each router has a routing table that includes corresponding output ports and time slots to every potential sender–receiver pairs. Contention-free routing algorithm is used in Philips *Æthereal NoC* system and it is also called as a *clockwork routing*. [24, 39, 48, 50]

Destination-tag Routing A destination-tag routing algorithm is a kind of an inverted version of the source routing algorithm. The sender stores the address of the destination, also known as a destination-tag, to the header of a packet. Every router makes routing decisions independently based on the address of the receiver. The destination-tag routing is also known as *floating vector routing*. [14, 71] The routing decisions are based on routing tables which are stored in every router.

Topology Adaptive Routing Deterministic routing algorithms can be improved by adding some adaptive features to them. The algorithm works like a basic deterministic algorithm but it has one feature which makes it suitable for dynamic networks. A central

controller can update the routing tables of the routers if necessary. A corresponding algorithm is also known as an *online oblivious routing.* [8] The cost and latency of the topology adaptive routing algorithm are near to the costs and latencies of basic deterministic algorithms. An advantage of topology adaptiveness is its suitability to irregular and dynamic networks.

2.5.2 Stochastic Routing Algorithms

Stochastic routing algorithms are based on coincidence and an assumption that every packet sooner or later reaches its destination. Stochastic algorithms are typically simple and fault-tolerant. Throughput of data is especially good but as a drawback, stochastic algorithms use plenty of network resources.

Stochastic routing algorithms determine *time to live* (TTL) of packets. It is a time how long a packet is allowed to move around in the network. After the predefined time has been reached, the packet will be dropped from the network. When a packet is dropped, the sender of the packet has to be notified so that it could resend the data. A few stochastic routing algorithms are presented below.

Probabilistic Flood The simplest stochastic routing algorithm is the probabilistic flooding algorithm. [19, 52] Routers send a copy of an incoming packet in all possible directions without any information about the location of the destination of a packet. The copies of a packet diffuse over the whole network similarly as a flood. Finally, most probably at least one of the copies will arrive at its destination and the redundant copies will be removed.

Directed Flood A directed flood routing algorithm is an improved version of probabilistic flood. It directs packets approximately in the direction where their destination is located. The main advantage of a directed flood is its lower network resource consumption compared with a probabilistic flood. [20, 53]

Random Walk A random walk algorithm sends a predetermined amount of copies of a packet to a network. Every router along the routing path sends incoming packets forward through some of its output ports. The packets are directed in the same way as in the directed flood algorithm. The network resource load, caused by a random walk routing algorithm, is lower than the load of the two stochastic algorithms presented above. [53]

Valiant's Random Algorithm Valiant's random algorithm is a partly stochastic routing algorithm. One main problem in deterministic routing algorithms is that they cause irregular load in a network. The load is typically especially high in the middle areas of the network. Valiant's random algorithm equalizes traffic load on networks that have a good path diversity. First the algorithm randomly picks one intermediate node and routes packets to it. Then the packets are simply routed to their destination.

Routing from beginning to the intermediate node and then to the destination are done using a deterministic routing algorithm. [68]

IVAL (*Improved VALiant's randomized routing*) is an improved version of the Valiant's random algorithm. It is similar to turn around routing. At the first stage of the algorithm packets are routed to a randomly chosen point between the sender and the receiver by using oblivious dimension order routing. The second stage of the algorithm works almost equally, but this time the dimensions of the network are handled in reverse order. Deadlocks are avoided in IVAL routing by assigning virtual channels to the physical channels of a router. [67]

2.5.3 Adaptive Routing

Adaptive routing algorithms are able to reconfigure the routing during run-time based on the status of the network. Several adaptive routing algorithms for Network-on-Chip are presented below.

Minimal Adaptive Routing A minimal adaptive routing algorithm routes packets always along the shortest network path. The algorithm is effective when more than one minimal, or as short as possible, paths between a sender and a receiver exist. The algorithm uses the route which is least congested. [14]

Fully Adaptive Routing A fully adaptive routing algorithm always uses a route which is least congested regardless of the length of the route. Typically, an adaptive routing algorithm ranks alternative congestion free routes to order of superiority. Then the shortest route used. [14]

Congestion Look Ahead A congestion look ahead algorithm gets information of congestion from other routers. Based on this information the routing algorithm can direct packets to bypass the congestion spots. [35]

Turnaround Routing Turnaround routing is a routing algorithm for butterfly and fat-tree networks. The senders and receivers are all on the same side of the network, as illustrated in Figure 2.11. Packets are at first routed from the sender to some of the intermediate nodes located on the other side of the network. In this node, the packets are turned around and then routed to the destination. The routing from the intermediate node to the receiver is done with the destination-tag routing algorithm.

Routers in turnaround routing are bidirectional which means that packets can flow through a router in both forward and backward directions. The algorithm is deadlock-free because packets only turn around once from a forward channel to a backward channel.

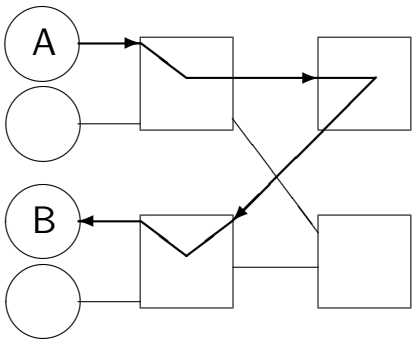


Figure 2.11: Turnaround routing from core A to core B.

Turn-Back-When-Possible Turn-back-when-possible is an algorithm for routing in tree networks. It is a slightly improved version of the turnaround routing presented above. When turn-back channels are busy, the algorithm looks for a free routing path on a higher router level. A turn-back channel is a channel between forward and backward channels. It is used to change the routing direction in the network. [32]

Q-Routing The functionality of the Q-routing algorithm is based on the network traffic statistics. The algorithm collects information concerning latencies and congestion areas, and maintains statistics about network traffic status. The Q-routing algorithm does the routing decisions based on these statistics. [40]

Odd-Even Routing The odd-even routing is an adaptive algorithm used in dynamically adaptive and deterministic (DyAD) mesh-shaped Network-on-Chip systems. The odd-even routing is a deadlock free turn model which prohibits turns from east to north and from east to south at tiles located in even columns, and turns from north to west and south to west at tiles located in odd columns (odd and even in terms of X coordinates in a mesh network). The DyAD system uses a minimal odd-even routing which reduces energy consumption and eliminates the possibility of livelock. [29]

Slack-Time Aware Most of the adaptive routing algorithms do not fit in systems which require strict real-time operation. In adaptive routing the latencies can vary. Packets can also be transferred along different paths, thus they can arrive at the receiver in wrong order. The delayed packets cause interruptions for example to audio or video stream.

Slack-time aware routing scheme divides traffic to guaranteed throughput (GT) and best effort (BE) traffic. In this scheme the Quality of Service is improved by utilizing the links of the GT traffic for the BE traffic when there is not much GT traffic in the network. [7]

Hot-Potato Routing The hot-potato routing algorithm routes packets without temporarily storing them in a buffer memory of a router. Packets are moving all the time without stopping until they reach their destination. When one packet arrives at a router, the router forwards it immediately towards its destination. However, if there are two packets going in the same direction simultaneously, the router directs one of the packets in some other direction. This other packet can possibly flow away from its destination. This occasion is called as misrouting. In the worst case, packets can be misrouted far away from their destination and misrouted packets can interfere with other packets. The risk of misrouting can be decreased by waiting short random time before sending each packet. Cost of the hot-potato routing is quite low because the routers do not need any buffer memory to store packets during routing. [21]

2.6 Network Problems

The two main categories of routing algorithms are deterministic and adaptive algorithms. Problems on deterministic routing, or routing without the information of the state of the network, typically arise when a network starts to block traffic. The only solution to these problems is to wait for the reduction of traffic, and try again. Deadlock and starvation are potential problems in deterministic algorithms as well as in adaptive algorithms which adapt the routing based on the state of the network. Additionally, livelock can occur in systems utilizing adaptive routing algorithms.

Deadlock Deadlock is a situation where two or more packets are waiting each other to be routed forward. All the packets reserve some resources and both are waiting each other to release the resources. Routers do not release the resources before they get the new resources and so the routing is locked. [65]

Deadlock situation can be solved by removing one of the packets temporary or permanently. However, better solution would be to design the routing algorithm in a way that it avoids the deadlocks.

Livelock Livelock occurs when a packet keeps spinning around its destination without ever reaching it. This problem exists in routing algorithms which do not route the packets always along the shortest path between the sender and the destination. Livelock should be cut out to guarantee throughput of packets. [65]

There are a couple of solutions to avoid the livelock. Time to live (TTL) counter counts how long a packet has travelled in the network. When the counter reaches some predetermined value, the packet will be removed from the network. However, packet dropping is rarely an absolutely good solution because the data payload is lost at least until a retransmit. Another solution is to give packets a priority which is based on the age of a packet. The oldest packet always finally gets the highest priority and will be routed forward.

Starvation Using different priorities can cause a situation where some packets with lower priorities never reach their destinations. This applies to prioritization which is not based on the age of a packet. The starvation occurs when packets with higher priorities reserve resources all the time and the lower priority packets do not get any resources to be transmitted. Starvation can be avoided by using a fair routing algorithm, which is an algorithm without fixed priorities, or reserving some bandwidth only for low-priority packets. [42]

2.7 Summary

This chapter presented essential principles and basics of Network-on-Chip paradigm. Basic components of NoC were presented and different network topologies were studied. Vital details, including network traffic classification, network flow control as well as routing algorithms were presented. Finally, typical problems arising in Networks-on-Chip were studied.

Chapter 3

Monitoring on Network-on-Chip

Network-on-Chip (NoC) is a promising interconnection paradigm for future high-performance integrated circuits [9, 16, 49] and its features and potentiality have been discussed in Chapter 2. To enable the full potential of the NoC there is a need for monitoring services to diagnose the system functionality, to optimize the performance and to do run-time system reconfiguration. The reconfiguration is required to keep the system working with reasonable performance regardless of faults and unbalanced load in the system [64].

Monitoring services can be roughly divided into two categories based on their main purpose: system diagnostics and traffic management. The former aims to improve the reliability and performance of the computational parts while the latter concentrates to the same issues in the communication resources. It includes fault detection, performance monitoring as well as computation load management. The other part of the monitoring services, the traffic management, focuses on the communication infrastructure which enables the interaction between computational components. Traffic management contains features to maximize communication infrastructure performance and reliability while optimizing its power consumption. It is used to balance the utilization of communication resources and to avoid congestion in the network as well as to reconfigure the routing in the network in case of faults or congestion related problems. The goal is to maintain the network functionality regardless of above-mentioned issues. The core of traffic management is network monitoring which observes network components and delivers the observed information so that it can be used to reconfigure the network. Typically there is a monitoring system to collect traffic information from the network and an adaptive routing algorithm which adapts its operation when the conditions in the network change (see Section 2.5).

Two types of information are collected in network monitoring: traffic status in the network and locations of faults in the network. Traffic statuses can be observed from different parts of a network: router activity, router FIFO occupancy, or link utilization, for instance. Fault information can cover the faultiness of different network components: routers or links, for instance. A network component is considered as faulty when it does not work as it should by its specification. The network components have to have

mechanisms to detect these faults [26]. There are several methods to do the detection. For instance, faulty links can be detected using methods which are based on the usage of spare resources or error control coding [37, 10]. Traffic management is discussed and analyzed in Chapter 6 while a significant part of a system management is studied in Chapter 7.

In a Network-on-Chip data are typically transferred as packets which have a destination address and an identifier which shows the type and purpose of the packet. These packet types can include for instance data packets, monitoring packets or mapping packets. All these packets are identical in terms of data transfer in the network. They are transferred in a network from a sender to a destination based on a routing algorithm. Identifiers are used to recognize packets so that every packet is forwarded to right component or handled in correct process. The majority of packets are data packets which are used to transfer payload data from a core to another. The control packets, which include for instance monitoring and mapping packets, transfer some information which is used for system management purposes. Mapping packets are discussed in Chapter 7. Network monitoring related data are transferred in monitoring packets. When a monitoring packet is received in the destination indicated by its destination address, the packet is forwarded to the monitor component for further processing.

The NoC monitoring systems which use shared communication resources transfer the network status data using monitoring packets. When centralized or clustered monitoring structures are used, these packets have to be initialized in probes, routed from probes to a monitor and from the monitor to the routers. Centralized control has its strengths and it is essential for several tasks. However, to optimize performance some of the traffic management tasks could be carried out with simpler distributed, or dynamically clustered, monitoring structures to decrease the load of the centralized control system. [55]

3.1 Purpose of Monitoring

The main purposes of monitoring in Network-on-Chip include traffic management and system diagnostics. Traffic management related monitoring is mainly based on the observation of communication resource usage. System diagnostics can be also applied for traffic management purposes for instance by using fault data for router reconfiguration.

System diagnostics can include various information which is collected for statistical or system control purposes. One application of system diagnostics is task mapping which is studied in more detail later on.

3.1.1 Traffic Management

To enable the maximum throughput, the communication resource utilization has to be balanced so that all the resources are utilized and the congestion is avoided. Adaptive routing and route reconfiguration are keys to balance traffic and to avoid congestion [14, 18, 57]. Traffic management should distribute traffic evenly over the whole network

and all the usable resources so that all the resources are used on their optimal load and highly loaded traffic hot spots are not formed.

Implementation of traffic management requires network monitoring; there should be a monitoring system which collects the information needed to optimize the routing. This information includes knowledge of current traffic conditions as well as knowledge concerning network functionality and faults. Basically, it is not necessary to have complete knowledge of the network status in each router; a router should have information on the network state at the region around it so that it can direct the traffic in the least congested directions.

3.1.2 Fault Detection

Modern integrated circuits are sensitive to transient faults. Because of a complex manufacturing process it is also possible that the circuits can contain permanent manufacturing faults which can occur as run-time errors e.g. due to electromigration. [54, 66, 70] The objective is to design a system that tolerates faults or recovers from errors caused by faults. The circuits should also be capable to reliable operation even though there are a few permanent faults.

The detection of faults requires a monitoring system to locate them and to provide the information for other components. For instance in the case of a manufacturing fault, the system could disable the faulty component and migrate its tasks to some other component. This kind of operation requires redundancy in the system to enable task migration.

The fault detection is also a part of the traffic management. When there are faults in the communication infrastructure, the monitoring system can detect them and inform the system to reconfigure the routing through faultless communication resources.

3.1.3 Task Management

A monitoring structure can be applied for different management and reconfiguration tasks in a Network-on-Chip. An essential task, included in the system management, is task mapping.

Typically tasks are mapped to cores at the initialization phase of system operation. The system diagnostic features of a NoC monitoring system make it possible to collect task mapping related statistics and use this data to map new tasks run-time. The goal is to find a suitable core which is able to execute the new task and at the same time to keep the computational load evenly distributed over the system. Balanced load distribution, both computational and communication load, is essential because that way the maximum performance can be achieved without formation of highly loaded hot spots. These hot spots are unfavorable in terms of performance as well as unbalanced power consumption and heat dissipation. Task mapping on NoC is studied in Chapter 7.

3.2 Monitoring Structures

An NoC monitoring system has the same main requirements as the NoC itself; it should be flexible, scalable and it should also be capable of real-time operation [12]. Scalability and flexibility ensure that a monitoring system can be used in different sized Networks-on-Chip without a time consuming redesigning process. A monitoring system itself should also be fault tolerant. A network monitoring structure consists of monitoring units (monitors) and probes as well as communication resources (e.g. routers), or wires, to connect these components to each other and to the NoC, as illustrated in Figure 3.1. The monitors are computational components which control the monitoring process and deliver observed data to its clients. The probes are connected to network components, e.g. routers, network interfaces or links, whose functionality they observe and deliver the observed data to the monitoring units.

A monitoring structure can have dedicated resources for communication between probes and monitors, or it can share the resources of the data network. In our research, we focus on shared-resource structures which require less additional resources than dedicated structures. We have also paid attention to dedicated resources for serial monitoring communication between monitors. In shared-resource structures non-intrusive operation of the monitoring system is a significant issue while in the serial monitoring communication the delays are crucial in terms of usefulness of the monitoring data.

If the network resources are dedicated for communication between the probes and the monitors, the probes can be controlled from the monitors. Otherwise the probes have to be more autonomous to be able to communicate over the shared packet switched network. Monitors are computational components which control the monitoring. Monitors collect the monitoring data, exchange the data with other monitors and provide information for network components which can use it to reconfigure their operation. The system diagnostics can share the resources of the traffic management but may also require

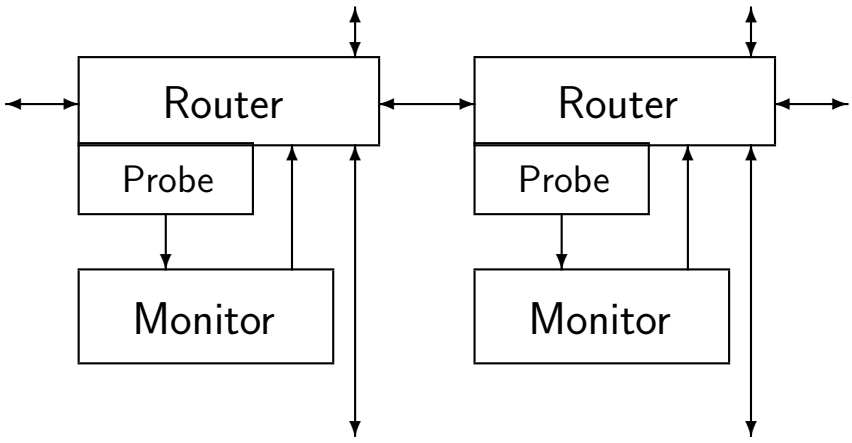


Figure 3.1: Network components and their connections.

dedicated components e.g. probes which are attached to the computational resources and centralized control devices which are not necessarily required in the traffic management.

Depending on the purpose, requirements and operational principles the monitoring system can have centralized, clustered, distributed or localized structure. The structure of the monitoring system defines how the monitoring data are delivered to the other components in the system and which of the components are able to access this information. Monitoring structure defines the number and type of monitors and probes, their placement, connections and tasks. A centralized monitoring structure has one central monitor and several probes that observe the data and deliver it to the monitor. In centralized structure, the central monitor has complete overall knowledge of the network but it causes significant amount of monitoring related traffic in the network. A clustered monitoring structure has a few cluster monitors and several probes. The network is divided into subnetworks, clusters, each of them having a cluster monitor and several probes. The complete network knowledge can be reached using inter-cluster communication but most of the tasks can be executed inside a cluster. However, a clustered structure still causes a considerable amount of monitoring traffic. [55]

Our research focuses on a scalable NoC monitoring structures where the knowledge of network status is spread widely enough over the network. There are two main factors taken into account while designing our NoC architecture. First, the structure should be not only aware of traffic but also aware of network faults so that network-level fault tolerance can be actively maintained during routing. Second, the structure should also be fully scalable to any mesh size. All the probes and monitors are identical and they work autonomously without any centralized control. The structure should also be easily reconfigurable for new applications which could turn up later on. The presented ideas can be adapted to NoC topologies of a different kind but due to its popularity, we have decided to concentrate on the mesh topology.

3.2.1 Centralized Monitoring

In a system with centralized monitoring there is a central monitoring unit and a number of probes which are attached to the network components. Probes deliver the data to the central monitor which collects the monitoring data and delivers information all around the system. Centralized monitoring has advantages on performance monitoring and on collecting statistical information from the system. Resource allocation for computational tasks may also require centralized control. However, a centralized monitoring system is inflexible and scales poorly. When the size of a system increases the average distance between the probes and the central monitoring component increases as well. Long distances between the probes and monitors increase the probability of network faults and cause more traffic to the network especially if the monitoring system uses shared communication resources. In a highly loaded network, this additional traffic can increase the probability of congestion. In centrally monitored systems, all the monitoring is carried out in a single component which makes it very susceptible to faults.

NoC monitoring systems have been presented in several papers. A dedicated control

network is used in a centralized operating system controlled NoC [47]. It has separate data NoC and control NoC from which the latter is used for centralized monitoring and control. An operating system controls the network management through the dedicated control NoC by collecting data from the processing elements (PE).

Another centralized NoC monitoring implementation is presented in [44] where a dedicated embedded processor is used to observe FIFO occupancies and packet transfer latencies. The monitoring processor is connected to routers with dedicated links. The collected data are used for partial dynamic reconfiguration purposes.

In [13] a transaction monitoring system for Æthereal NoC [24] has been presented. This system, which monitors transactions in NoC components, can be configured to shared or dedicated communication resources. The probes monitor bit-level data which have been interpreted as transactions and forwarded to a centralized monitor (Monitoring Service Access point).

A congestion control system which monitors links and uses shared communication resources is presented in [69]. It measures the amount of congestion by monitoring link utilization and adjusts data sources to control congestion. All these systems include a centralized monitoring unit which collects the observed data and controls the system operation.

All the above implementations are meant for traffic management while a monitoring system for run-time optimization and resource allocation was presented in [23]. The probes are implemented within NIs and they observe throughput, latency and event occurrences in PEs.

3.2.2 Clustered Monitoring

In clustered and distributed NoC monitoring there is no centralized monitoring unit where the global monitoring data are collected. In the clustered approach the system is divided into subsystems, clusters, each of which has a cluster monitoring unit where the monitoring data from the probes in the cluster is collected. The cluster monitors can exchange information with each other. A clustered monitoring structure with the cluster size of four is presented in Figure 3.2. If the cluster size is relatively large the number of monitors can be kept low. However, the smaller the cluster is, the less effect a faulty monitor has on the system operation. Communication distances are naturally also shorter when cluster size is small.

ROAdNoC is a runtime observability infrastructure which uses shared communication resources and basically has a clustered structure but can also utilize centralized control if necessary. It observes routers and NIs and collects error, run-time and functional information. [2] A notably similar structure is presented in [12]. Both of these structures utilize event-based communication and the latter can be used to monitor all the NoC components. On event-based communication, the probes generate events from the monitoring data which allows the monitoring data abstraction and leads to communication overhead reduction. A clustered monitoring structure is presented also in [41]. It collects information from the routers, identifies which senders cause the conges-

tion, and does traffic shaping by restraining these sender cores. The control data have dedicated communication resources.

3.2.3 Localized Monitoring

Localized monitoring is the simplest monitoring scheme to be implemented in an NoC. Typically there are no actual monitor components but the probed values are used locally without further processing or they are delivered to the neighboring routers. The localized monitoring is typically used in adaptive routing where the probes observe output channel reservation or the router utilization levels.

Localized monitoring systems have been presented in several papers as a part of adaptive routing mechanism; for example for Nostrum NoC [46]. In the basic version of the Nostrum NoC there is a probe on each router. Those provide traffic level information for the neighboring routers. The difference from distributed structures is that there are no separate monitors which collect the information and exchange it with each other.

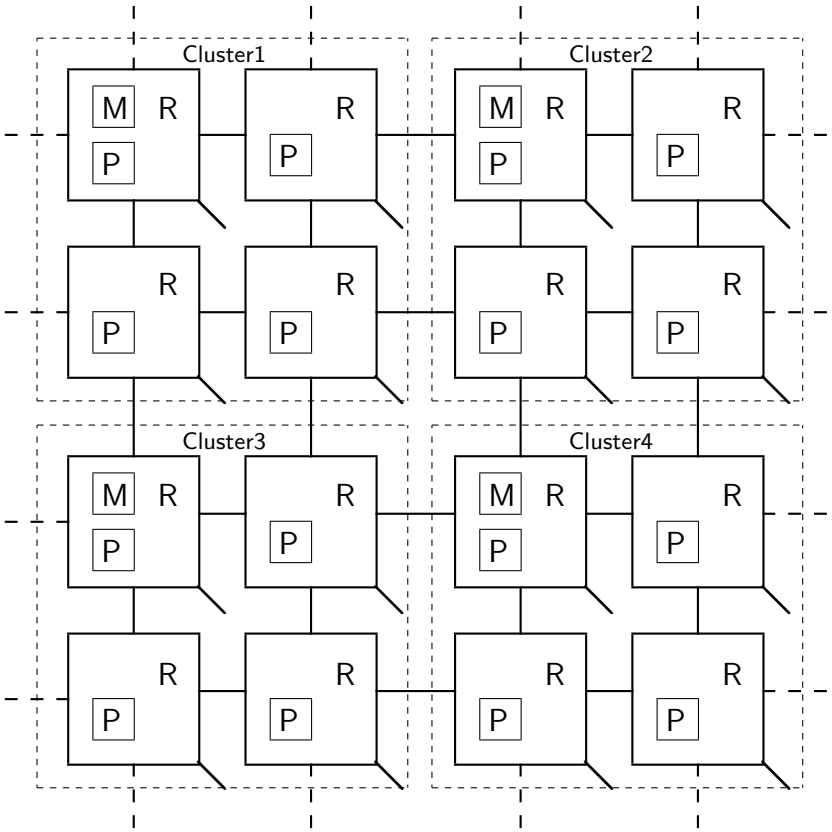


Figure 3.2: Clustered monitoring (cluster size: 4). Probes (P) and monitors (M) are integrated into the routers (R).

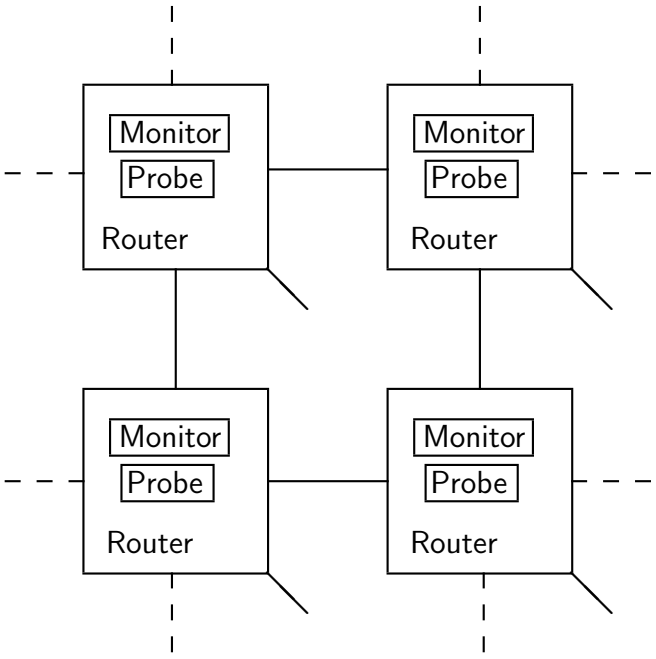


Figure 3.3: Distributed monitoring in NoC with 4 routers. Probes and monitors are integrated into the routers.

3.2.4 Distributed Monitoring

In a distributed monitoring structure the monitoring system is distributed over the system. There is a monitor unit at each router in the network and the monitors communicate directly with each other. This should have positive impact on the communication overhead because the monitoring related communication is done straight from a monitor to another monitor without circulating it through any central unit which could possibly be located relatively far away from both of the communicating monitors. A distributed monitoring system is presented in Figure 3.3. In a distributed monitoring system, each monitor does not necessarily need complete knowledge of the current network state, but a good knowledge of the status in its neighborhood and the resources near it. Complete knowledge may not be essential in large NoCs where there are functionally consistent resources situated in different locations around the system. A scalable monitoring with regional congestion awareness is presented in [25]. It is aimed to balance the workload in the network based on the amount of congestion.

When the monitoring is highly distributed, the impact of faulty monitoring units is minimal to the overall functionality of the system. Faultiness of a single monitor or probe has direct impact only to the router to which the faulty component is attached and to some extent to the neighbors of the router which cannot get valid information from its faulty neighbor.

3.3 Monitoring Trade-offs

In a Network-on-Chip the monitoring is essentially not a vital part but a major additional service. Having a monitoring system is not absolutely crucial in terms of system functionality but it can have notable positive influence on system performance. A monitoring system brings several advantages to the monitored system. However, arising trade-offs should be taken into account as well.

Optimal network monitoring coverage requires monitoring of every single component of the system. However, this implicates high monitoring overhead in area, energy consumption and communication. In our research, the network monitoring is implemented by monitoring the load of routers. Monitoring of every single link would multiply the monitoring overhead. One weakness in a router monitoring based network monitoring is the link status and traffic flow direction uncertainty. By monitoring only routers, the observer cannot be sure which channels of a router are loaded and which are free to use. However, regardless of the routing direction, a traffic flow loads the routing decision and switching logic. This reduces the significance of traffic flow direction awareness.

3.3.1 Resource Allocation

Implementation of the monitoring services affects the system to be monitored. The resource allocation should be taken into account when balancing between manufacturing costs, area overhead, intrusiveness and system performance. [3, 30, 33]

A monitoring system with dedicated resources is separated from the actual system and built separately of the data NoC [73]. The monitoring components and the communication resources, used to move the monitoring data, are only in the use of the monitoring system. A monitoring system with dedicated resources is straightforward to design and it has a minimal impact on system performance. However, this approach increases the area overhead and the complexity of the system, which typically also means an increment in power consumption.

If the system does not include dedicated resources for monitoring, a monitoring system can be integrated into the communication system. The monitoring data are transferred in the same communication infrastructure with the actual data. The resource sharing requires virtual channels in the routers to guarantee monitoring system functionality under congested network conditions [15]. A monitoring probe can be connected to the network through a dedicated network interface or it can share a network interface of a processing element.

The intrusiveness of a monitoring system defines how much the monitoring process disturbs the functionality of the system which is monitored. The objective is to design as non-intrusive monitoring system as possible. The probes should operate without disturbing the devices which they are probing and the traffic overhead on shared communication resources should be kept low. A monitoring system using shared resources does not require additional communication resources, which limits the area overhead. However, resource sharing can affect communication performance and that way interfere

with the actual operation of the system. Shared-resource monitoring systems are fascinating because of their minimal added complexity. Nevertheless, due to intrusiveness aspects, shared-resource monitoring systems have to be designed carefully. [13]

3.4 Fault Tolerance of Monitoring Systems

The monitoring systems can be used to improve the fault tolerance of NoCs as was described in Section 3.1.2. However, the monitoring systems themselves are also vulnerable to faults which has to be taken into account in the design process.

The faults can occur in the data moved between monitors or they can affect to the functionality of the actual monitor components. If a monitoring system shares the resources with an NoC, the fault tolerance methods utilized to the actual data can be applied also to the monitoring data. Otherwise, if a monitoring system has dedicated communication resources there should be separate fault tolerance mechanisms for these. The fault tolerance mechanisms for data could be e.g. triple modular redundancy, or error control codes [10, 31, 36, 45].

To ensure the intended functionality of the actual monitor components a built-in self-test mechanism could be implemented [72]. Self-test mechanism regularly checks the functionality of a component so that faulty components can be recognized, disabled and possibly replaced with spare resources. Similar mechanisms for NoC routers and links have been implemented previously in [22] and [27] correspondingly.

3.5 Summary

Introduction to monitoring in Networks-on-Chip were given in this chapter. At first, the purposes of network monitoring were studied. Different monitoring structures were presented. Finally, trade-offs of monitoring and fault tolerance of monitoring systems were discussed.

Chapter 4

Dynamically Clustered Distributed Monitoring

Scalability is an essential feature of Networks-on-Chip [9]. The distributed approach makes it possible to scale the monitoring system similarly as the actual NoC can be scaled. Centralized monitoring solutions scale poorly and are built separately with the NoC. While a centralized structure can be feasible on a small system the limited scalability is against the main principles of the NoC paradigm: an NoC should be fully scalable in any size and its performance should not be weakened when the size of the system is increased. Our hypothesis is that the use of distributed monitoring systems has a positive impact on traffic overhead caused by network monitoring, which decreases the demand for dedicated resources. The distributed monitoring structure has potential to be more fault tolerant than the centralized and clustered systems because the faultiness of a single component has an immediate influence only to just a few other components. Faultiness of a centralized component may endanger the functionality of the whole system.

The distributed monitoring system has several potential advantages when compared with the centralized and clustered monitoring. In centralized and clustered monitoring the monitoring data have to be moved to the monitoring unit for processing and then back to the routers before the information can be exploited. In a distributed system, the monitor receives information from its local probe and its neighbors and performs the required processing. It is also possible to implement centralized control at the operating system level while the traffic management, for instance, uses distributed monitoring.

4.1 Dynamically Clustered Monitoring Structure

We propose an NoC monitoring structure which is originally focused on traffic management, including routing reconfiguration and traffic load balancing, but can also be utilized for system reconfiguration and problem recovery for instance in the case of faults or congestion. Fortunately, the origin of a problem is not relevant from the traffic

management point of view.

Scalable NoC monitoring is based on an idea that there is no need for comprehensive status information of the whole network in a single centralized monitoring unit. To guarantee high and predictable performance the congestion should be avoided beforehand. We propose that efficient traffic management can be achieved without centralized network monitoring. When the traffic overhead of the traffic management is minimized, the remaining bandwidth can be used for normal data traffic as well as essential centrally controlled features which cannot be realized with a distributed structure. Most of the traffic management tasks can be executed without centralized control, thus, there is no reason to waste resources on these purposes.

In our monitoring approach every router has its own status which is based on the utilization level of the router and its neighbors. Every router has up-to-date information about the statuses of five or the twelve closest neighbor routers around it. This monitoring protocol can be implemented in centralized, clustered or distributed monitoring systems. The distributed version of this monitoring structure is also called the dynamically clustered structure.

A dynamic cluster is the area around a router from where the router status have been collected and to where the status of the router has been delivered. In statically clustered systems (see Section 3.2.2) the cluster borders are fixed and that way the routers near the cluster borders have unequal amount of status information from neighbor routers because some neighbors are in the same cluster and others in a different cluster as the router itself. In dynamically clustered structure the router is always placed in the middle of its own dynamic cluster which balances the amount of neighbor status data.

Figure 4.1 illustrates an NoC with 100 routers. Every router has a dynamic cluster but only three of them are illustrated to keep the presentation readable. A router has the status information of the components in its own dynamic cluster. This information drifts from the network components to a monitor along simple neighbor to neighbor information exchange. The clustered structures, presented earlier in the Section 3.2.2 are statically clustered, which means that the borders of the clusters are fixed.

In statically clustered systems, the cluster monitor defines an overall status of its local cluster. This overall status is delivered to the neighboring clusters where it is forwarded to the routers. In a dynamically clustered structure, every router has an equal amount of neighbor status information. This differs from statically clustered structures where the amount of neighbor data depends on the location of a router in a cluster. The routers which are situated near the edges and corners of static clusters have inaccurate neighbor status knowledge compared to the routers situated in the middle of a cluster. In a centralized implementation, the accuracy of monitoring statuses is on the same level with the distributed implementations. The difference is that in the centralized systems the status update requires a large amount of data to be transferred between routers and the central monitoring unit (see Section 3.2.1).

The presented monitoring systems can be also used on small system reconfiguration tasks which do not require centralized control. Reconfiguration messages can be broadcasted over the monitoring system all around the network. The traffic diffusion analysis,

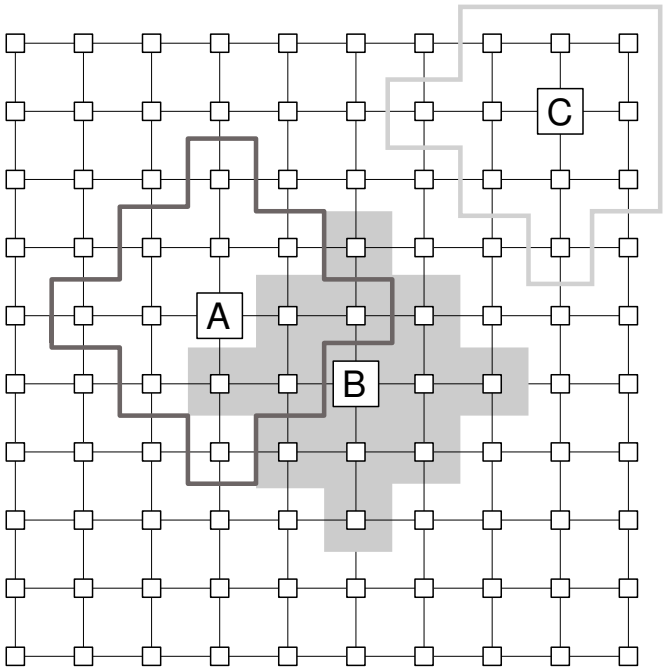


Figure 4.1: Dynamic clusters of three routers, A, B and C. Each of the 100 routers in this network has a similar dynamic cluster around it.

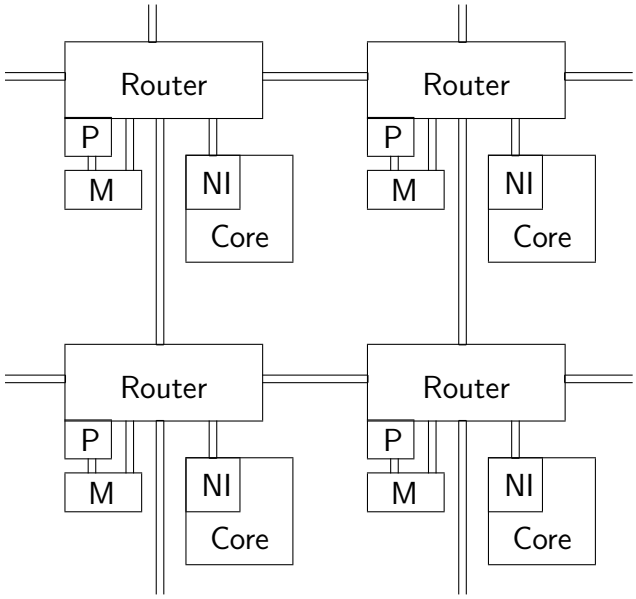


Figure 4.2: Network topology showing the connections between routers, networks interfaces (NI), monitors (M), probes (P) and cores in a part of mesh shaped Network-on-Chip.

which is strongly related with broadcasted reconfiguration messages, is presented later.

Dynamically clustered monitoring (*DCM*) can be considered as distributed monitoring because it does not require any centralized control. There is a simple monitor and a probe attached to each router in the network. The used mesh topology is illustrated in Figure 4.2. Centralized control is not required but the monitors exchange information autonomously with each other. The delivery of router status is called status data diffusion. The dynamic clusters of different routers overlap with each other. The simplest dynamic cluster includes 4 closest neighbors of a router but it can be expanded to neighbors' neighbors and so on. A system which uses DCM for traffic management could have for instance operating system level control for tasks that need complete knowledge of the system. When traffic management is implemented with a DCM structure the load of the network can be optimized.

4.2 Principles of Analysis

We analyzed and compared the different monitoring structures and their features at conceptual level as well as at transaction level. The conceptual analysis, presented in this chapter, is based on calculations made by hand and with MATLAB. Network-on-Chip simulation environment, which is presented in Chapter 5, makes possible to analyze NoC designs on transaction level. The analysis based on the simulation results is presented in Chapter 6. The conceptual level analysis is sufficient in terms of monitoring structure

comparison. In our analysis, we use widely exploited mesh topology and assume that in every case there is an equal amount of network status data on every router and the data update frequency is constant [14, 42].

A monitor is always attached to a router. Therefore, the transfer of the status data between router and its local monitor does not load the network. We also assume that

$$N_{width} = N_{height} \tag{4.1}$$

which means that an NoC has an equal number of rows and columns. The NoC width and height represent the number of routers next to each other in horizontal and vertical dimensions, respectively. The corresponding assumption for clusters is

$$C_{width} = C_{height} \tag{4.2}$$

which defines that clusters are squares. The number of routers in an NoC is depicted using term N_{size} and the number of routers in a cluster using term C_{size}. The last assumption is that

$$N_{size} = kC_{size} \qquad k \in \mathbb{N} \tag{4.3}$$

which defines that an NoC consists of one or many complete clusters. The assumptions define the typical features of mesh Networks-on-Chip. They are carefully defined to enable structure comparison and analysis without remarkable inaccuracy. The metrics to be analyzed are traffic overhead caused by the monitoring system, the diffusion of the data in the network, maximum path lengths on monitoring data transfers and at a coarse level the cost of centralized, clustered and distributed NoC monitoring systems. The traffic overhead analysis is essential when estimating the intrusiveness of a monitoring system. The rate of the information diffusion in the network has an influence on the reaction speed and it is closely related with the maximum path length analysis. The maximum path length analysis is to some extent also related with monitoring system fault tolerance because the probability of fault related problems increases when the information transfer paths get longer. Additional challenges can be inflicted by the utilized fault tolerance mechanisms which may slow the system operation down at some point for instance due to retransmissions.

4.3 Monitoring Traffic Overhead

In a shared-resource NoC the monitoring system uses the same communication resources that are used for the actual data transfer. In terms of intrusiveness of the monitoring, the amount of monitoring traffic is an essential detail. This amount of monitoring related traffic is called monitoring traffic overhead.

The size and structure of the monitoring system, the monitoring cluster size and the number of clusters have an effect on the traffic overhead caused by the monitoring service. In statically clustered structures, there is communication (a) from probes to a monitor,

(b) between monitors and (c) from a monitor to routers. Because the distributed system can be seen as a clustered system with the cluster size of 1, there is no communication inside a cluster, only between them (b). In centralized structures there is only one large cluster which means that the communication is only between probes and monitors (a) as well as between monitors and routers (c) It is assumed that the amount of data on a traffic status update between two network components is so small that a single packet can hold all the information which is transferred. One transaction in this analysis means moving a packet from a router to its neighboring router, from a probe to a monitor or from a monitor to a router. The number of transactions required in a complete network status update in a centralized monitoring structure is represented with equation

$$T_{cent} = 2H_{avg}\Big(N_{size} - 1\Big) \tag{4.4}$$

where H_{avg} is the average hop count between the monitor and any router and N_{size} is the size of the NoC that is the number of routers in the NoC. The multiplier 2 depicts bidirectional traffic between the routers and the monitor. The average hop count can be calculated if a deterministic routing algorithm is used but may vary in systems using adaptive routing with non-minimal routes. In a clustered monitoring structure the number of required transactions in a complete status update is represented with equation

$$T_{clust} = 2\left(H_{avg}\Big(C_{size} - 1\Big)\frac{N_{size}}{C_{size}} + C_{borders}\sqrt{C_{size}}\right) \tag{4.5}$$

where the multiplier of 2 illustrates bidirectional traffic in and between clusters. H_{avg} is the average hop count between any router and the monitor of the same cluster. N_{size} and C_{size} are sizes of an NoC and a cluster, respectively. Hence, the ratio of these is the number of clusters. The first term of the sum represents the traffic inside clusters and the second term between the monitors of neighboring clusters. The square root depicts the distance between the monitors. $C_{borders}$ is the number of borderlines between clusters and is calculated using equation

$$C_{borders} = 2\left(\left(\sqrt{\frac{N_{size}}{C_{size}}} - 1\right)\sqrt{\frac{N_{size}}{C_{size}}}\right) \tag{4.6}$$

where the second square root represents the number of rows in the network which is multiplied with the term representing the number of cluster borderlines in a row. The multiplier of 2 completes the equation to include also the borderlines in the other dimension, that is horizontal and vertical borderlines. In distributed monitoring structure there is traffic only between neighboring routers and the number of transactions is calculated using equation

$$T_{dist} = 4N_{size} - 4\sqrt{N_{size}}. \tag{4.7}$$

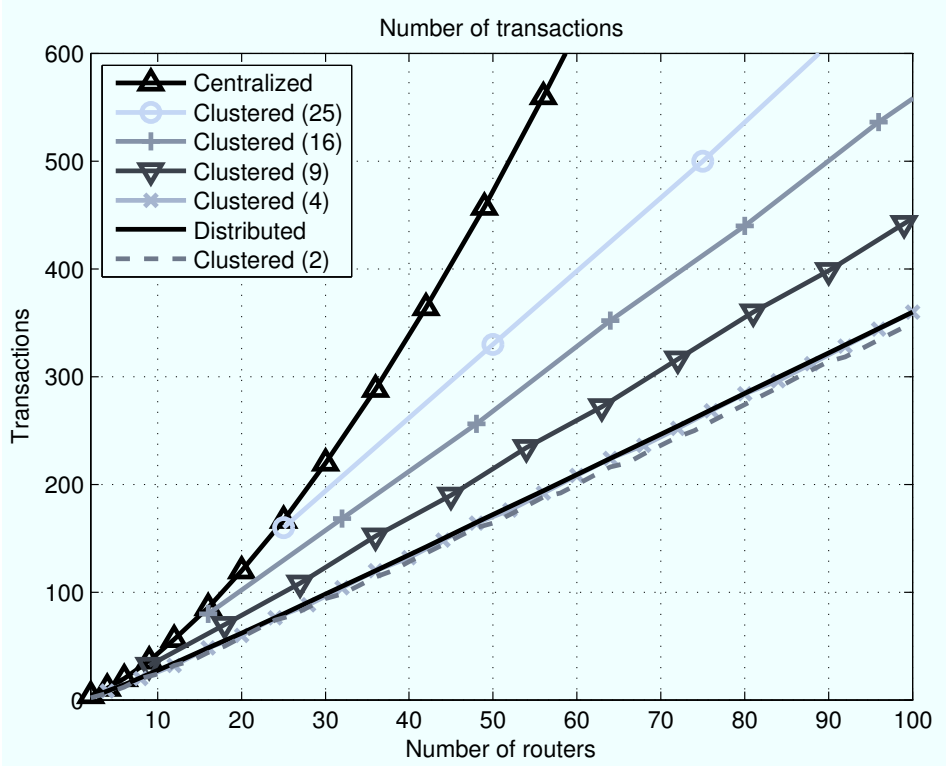


Figure 4.3: Number of transactions required in complete network status update.

The first term of the difference represents the traffic which is sent from each router in four directions. The second term corrects the amount of traffic in the edges of the network where there is traffic only in two or three directions. Note that $T_{dist} = T_{clust}$ when $C_{size} = 1$ and $T_{cent} = T_{clust}$ when $C_{size} = N_{size}$.

The amount of monitoring communication in NoCs of different sizes and with different cluster sizes is illustrated in Figure 4.3. The figure shows that larger cluster sizes cause more traffic and the centralized system clearly causes the largest traffic overhead. The cluster size of two is also analyzed to complete the analysis. One can notice that it causes the lowest overhead while the overhead caused by the distributed structure and the clustered one with the cluster size of four is just slightly higher. However, clusters with two routers are not well suited to symmetrically structured networks because they spread the traffic information unevenly.

The lengths of paths between routers, probes and monitors affect the delay of a complete update process. The comparison of the longest packet transfer paths was made by calculating the longest productive paths between the possible positions of routers and monitors. A productive path is the shortest routing path between two nodes in a network. This comparison is presented in Figure 4.4. The figure shows the longest, or

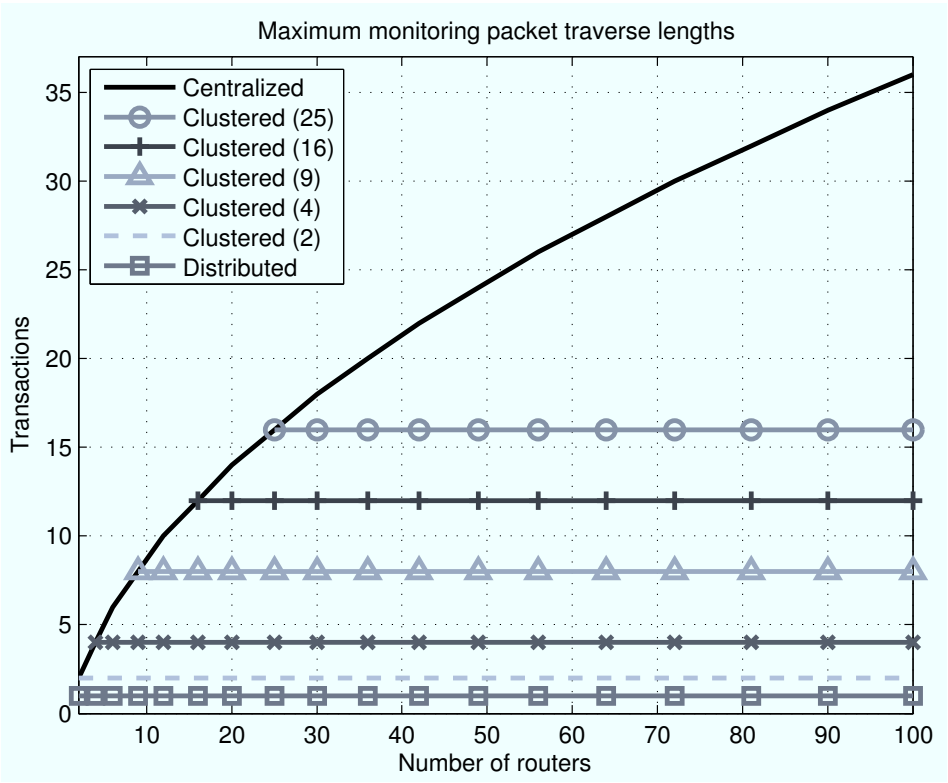


Figure 4.4: Comparison of the maximum traverse lengths of a packet on a monitoring status update.

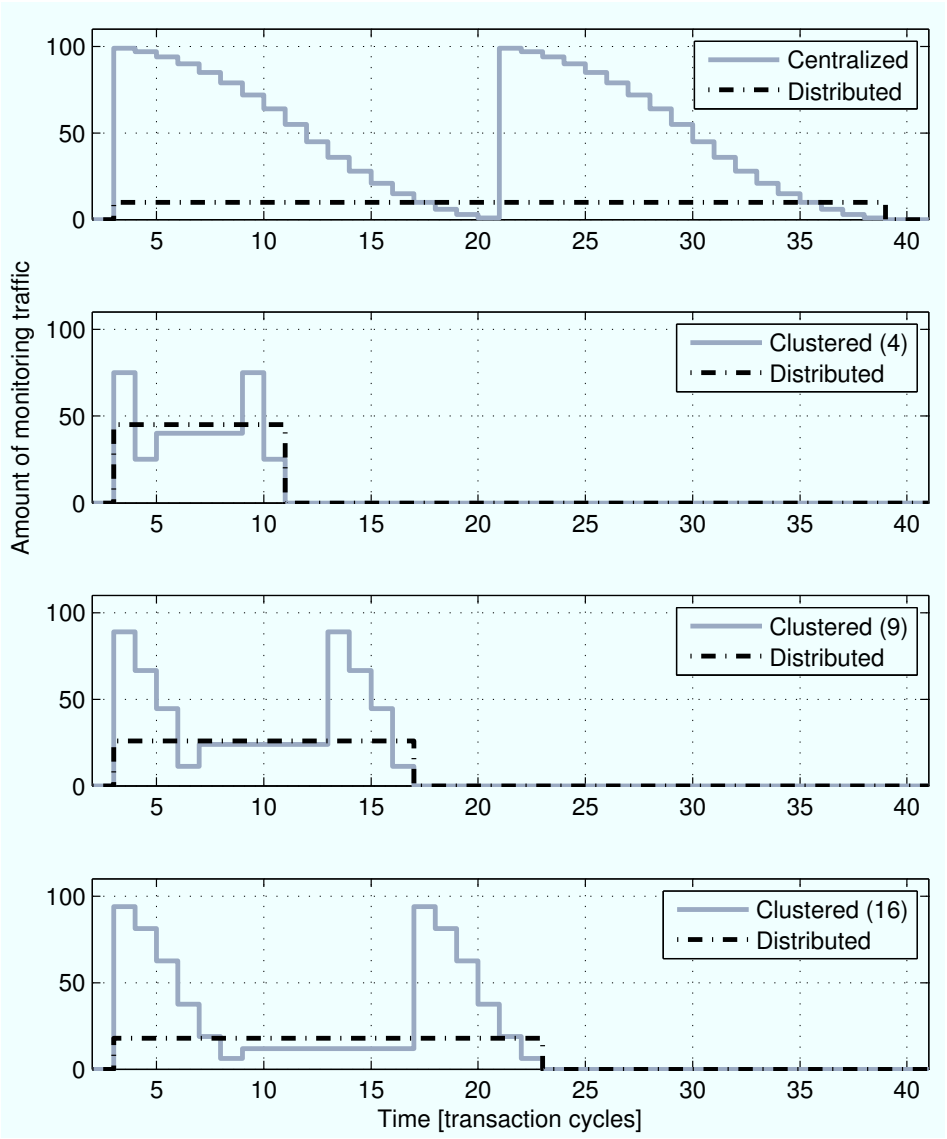


Figure 4.5: Distribution of monitoring traffic load in time.

worst case, distances between routers and monitors or two neighboring monitors (which one is longer) in different structures. Lengths of the information transfer paths have to be taken into account because the monitoring data transfer delay has a significant impact on the monitoring system performance.

The influence of congestion and other problems cannot be seen in this analysis. If the paths of the monitoring packets are significantly rerouted due to problems in the network, the traverse lengths and hop counts can increase. The severity of a problem for a short path is relatively significant and may completely disable the path. However, the probability of problems is higher for the longer paths and that way the longer the path, the more vulnerable to faults it is.

One thing to note is how the added monitoring communication overhead is distributed over the time. The amount of monitoring traffic in a network during a complete network status data update is presented as a function of time in Figure 4.5. The analysis includes centralized monitoring structure as well as statically clustered structures with cluster sizes 4, 9 and 16. Each of these structures is separately compared with the distributed monitoring structure. As can be seen from the Figure 4.5 the load caused by centralized and statically clustered monitoring structures is not only relatively high but also distributed uneven over time while distributed monitoring systems are able to cause a lot more even load as shown in Figure 4.5. The peaks are caused by varying distances between communicating units. At the beginning the amount of traffic is in its maximum and it decreases while the data reach the destinations closer to the sender. Two peaks exist because of the bidirectional nature of the traffic. The traffic between these two peaks is the caused by the communication between clusters. However, a dis-

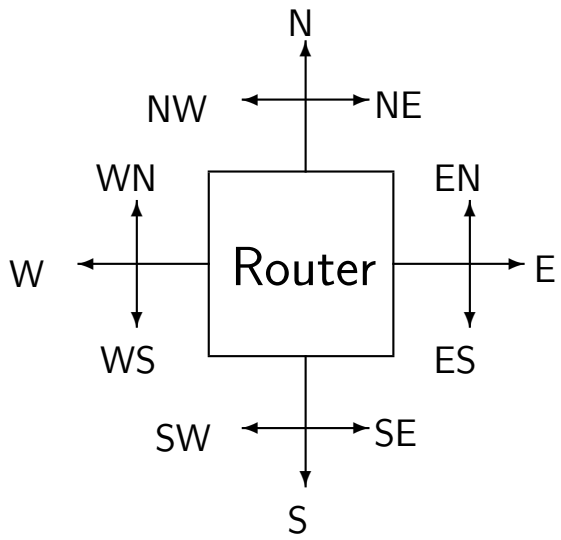


Figure 4.6: Routing directions. The abbreviations stand for N: north, S: south, E: east, W: west and R: router.

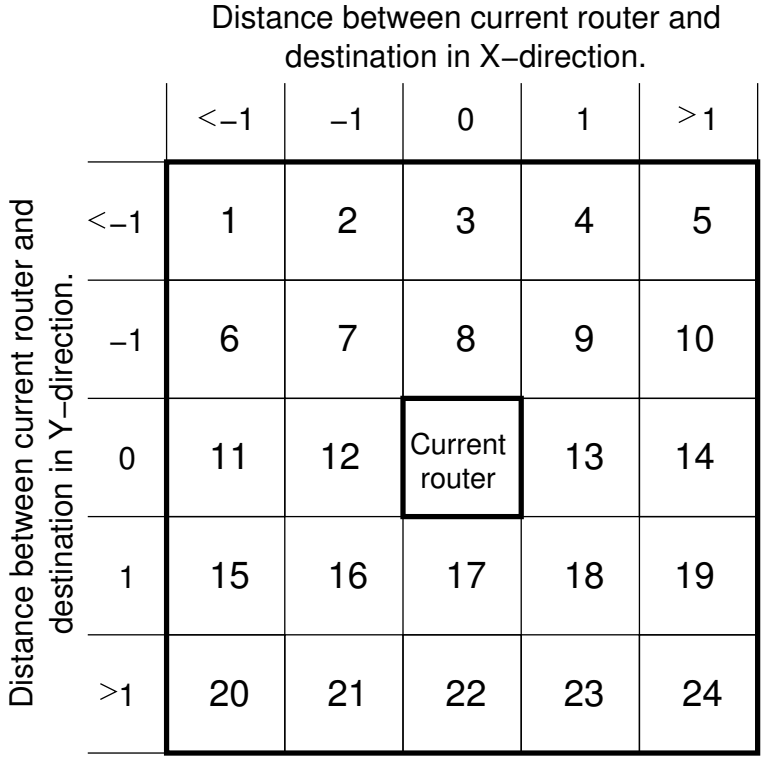


Figure 4.7: Routing directions in our experimental routing algorithm.

tributed monitoring system can be also configured to use the communication resources when the resources are free. This way there could be high monitoring communication peaks in the network but without caused interference against data communication.

4.4 Routing Algorithms

The Network-on-Chip simulation environment utilizes an adaptive routing algorithm [14]. The algorithm determines the routing direction among the twelve candidates which include four main directions (north, south, east and west) and the intermediate directions (for example north-west and south-east and their counterparts west-north and east-south). These directions are illustrated in Figure 4.6. The algorithm chooses an output port to be used among the actual routing direction and its nearest neighbor directions. The decision is based on the traffic status values and the link statuses in potential directions. A packet which cannot be delivered is put back in the router's memory and rerouted. A packet lifetime is also utilized to prevent undeliverable packets from blocking the network. To prevent congestion, the packets are not sent in directions where receivers' buffers are fully occupied.

We also propose an experimental routing algorithm where the destination's distances to the core in different routing directions are taken into account. In this algorithm the destinations are classified to 24 different routing directions which differ in varying distances in different routing dimensions (X and Y dimension). These directions are independent of the network but naturally more advantage is gained in large networks. These routing directions are illustrated in Figure 4.7. The idea behind this algorithm is that a packet should be routed always in a dimension where the distance to the destination is longer. This way the possibility to change the dimension remains making it possible to evade problematic areas without extending the routing path. The distance resolution in this experimental routing algorithm has three levels. The algorithm distinguishes the routing directions based on the following criteria: the destination is (*1*) in the current row/column, (*2*) in the next row/column in some direction or (*3*) further away. Hence, there are altogether 24 different routing directions. These routing directions enable extensive classification of different routing cases and that way each case can be handled optimally.

In each of these 24 routing directions we have ranked the possible output ports based on the destination and network conditions. Every time a packet is routed the algorithm identifies the routing direction and uses available traffic status and fault information to select the appropriate output port.

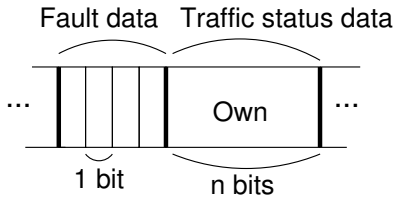
4.5 Monitoring Algorithms and Communication

The router statuses in the DCM structure are represented with two binary numbers, one for traffic status and another for fault information. In the DCM structure the status of a router is based on the occupancy of the FIFO buffer where packets are waiting to be routed forward. The status (S) of a router is calculated using Equation

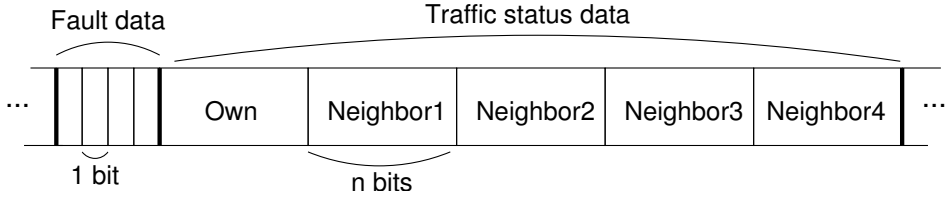
$$S = \frac{B_{occ}}{B_{size}} G \tag{4.8}$$

where B_{size} is the size of the buffer, B_{occ} the number of packets currently stored in the buffer and G is the used status data granularity. The status data granularity defines the maximum value of the status data while the minimum value is always zero.

In centralized and clustered monitoring structures the monitoring packets are transferred in a network similarly as the data packets. The dynamically clustered approach simplifies the monitoring communication because the routing of the monitoring packets is not needed but substituted with a packet type recognition. Every monitor sends its status data and the neighbor status data it is forwarding to all its neighbors. The receiver recognizes these packets as monitoring packets and does not send them forward. A monitor stores the status data from received monitoring packets to its memory and provides the received information forward to its own neighbors. This way the routers are able to receive information not only from their neighbors but also from the neighbors of their neighbors. In dynamically clustered monitoring structure the network status



(a) Payload when cluster size is 5.



(b) Payload when cluster size is 13.

Figure 4.8: Structure of monitoring packet payload with cluster sizes 5 and 13. n is the number of bits required to represent a traffic status value with a certain granularity.

data spread over the network without centralized control and without routing related processing.

In the basic DCM structure the monitoring data are transferred in packets using the actual data network. These packets are called monitoring packets. The monitoring packets have a higher priority in the routers so that they can be transferred even when there is congestion in the network. Monitoring packets are used to adjust the functionality of the network in critical situations so the delivery of monitored data has to be guaranteed. The monitoring packets are sent from a monitor to a monitor, but because the monitors are not directly connected to each other, the packets are transferred via routers and links.

The router statuses in the DCM structure are represented with two binary numbers, one for traffic status and another for fault information. The status of a router is based on the occupancy of the FIFO buffer where packets are waiting to be routed forward. The faultiness of a single component can be represented using a single bit while number of bits in the traffic status values is related to the size of the FIFO buffer, required accuracy as well as the used additional status data processing (see Section 6.2.1). The resolution of the traffic status data is defined with status data granularity. The granularity defines the number of different values which can be used to illustrate the level of traffic load. For instance, when the status granularity of a router is 4 there are 4 different levels of traffic (1: no or just a little traffic, 4: highly loaded and cannot receive new packets, 2-3: scaled linearly between the edge values). The finer the granularity the better the

accuracy of the status values is.

In the DCM structure the monitors exchange their own and their neighbors' statuses with each other. Typically monitoring packets include fault statuses of nearby links and one or more traffic statuses of routers depending on the size of the monitoring cluster. The structure of a monitoring packet payload in systems with monitoring cluster sizes 5 and 13 is presented in Figure 4.8. The contents of a monitoring packet payload are discussed in Section 6.2.

In centralized and clustered monitoring structures the monitoring packets are transferred in a network in the same way as the data packets (see Section 4.1). The dynamically clustered approach simplifies the monitoring communication because the routing of the monitoring packets is not needed but substituted with a packet type recognition. Every monitor sends its own status data and the neighbor status data, which it has received from its neighbors, to all other neighbors. The receiver recognizes these packets as monitoring packets and does not send them forward. The transfer distance of a monitoring packet is always one hop, from a router to its neighbor router. This simplicity of monitoring packet transferring combined with the simple routing procedure of monitoring packets makes possible to keep the latency overhead on tolerable level for most applications. The presented DCM structure is targeted for applications without strict real-time constraints because the in-time delivery of packets cannot always be guaranteed. This is a trade-off of the improved fault tolerance.

A monitor stores the status data from received monitoring packets to its memory and provides this information forward to its own neighbors. This way the routers are able to receive information not only from their neighbors but also from the neighbors of their neighbors. In dynamically clustered monitoring structure the network status data spread over the network without centralized control and without routing related processing.

4.6 Summary

This chapter presented the dynamically clustered distributed monitoring structure. The structure was analyzed on conceptual level concentrating on the traffic overhead caused by the monitoring system. Routing and monitoring algorithms, which are used with the presented monitoring structure, were studied and monitoring related communication discussed.

Chapter 5

Simulation Environment

SystemC based simulation environment has been implemented to enable extensive analysis of Networks-on-Chip. The simulation environment is required to model and analyze Network-on-Chip (NoC) architectures in terms of functionality, performance and fault tolerance.

SystemC is an extension of C++ programming language [63]. It makes possible to model concurrent systems using standard C++ language. The simulation environment is designed for transaction-level modeling and it utilizes transaction-level modeling libraries of SystemC (TLM 1.0). [1] Transaction-level simulation environment is suitable for Network-on-Chip architecture simulations especially in the early phases of system development. The environment is easily reconfigurable to model different architectural solutions and features without complex immersion to physical implementation for example using hardware description languages.

The simulation environment includes models of Networks-on-Chip with centralized, clustered and distributed traffic monitoring. In the first phase the simulation environment was designed to model NoCs utilizing shared-resource communication where the resources are shared between the data packets and the monitoring packets. Later the environment was further developed to model NoCs with dedicated communication resources. These dedicated resources are utilized for serial monitoring communication.

The essence of the simulation environment is the simulation mechanism which is able to execute transient analysis. In transient simulation the cores are sending packets following a specific traffic pattern and the key figures are recorded during the simulation. These figures include, among others, the numbers of sent and received packets, the transfer delay and the number of dropped packets from each node separately. The simulation duration can be customized and it is possible to run simulation in multiple NoCs simultaneously with identical simulation parameters. In this case, the results can be represented as averages from the simultaneously running NoCs. The amount of traffic during the simulation can be adjusted with the traffic pattern. There is also an option to put faults in the network. This feature enables the fault tolerance analysis of NoC. All the network faults are modeled using the link faults. For instance, a faulty router

can be modeled with a bunch of faulty links. Faults can be put in the system randomly or manually so that modeling of larger uniform fault areas is also possible.

5.1 Architecture

The simulation architecture consists of architectural components and simulator specific components. Architectural components include models of the components which are included in real NoC implementations: a router, a link, a network interface, a monitor and a core. The core is used to model components which include for instance computational cores, memories and I/O components. The simulation environment is designed for NoC communication infrastructure analysis so the core model includes functionality which is reasonable from the communication point of view.

The simulator specific components do not have physical equivalents but are required in the simulation environment. In real implementations a data packet is an abstract concept but in the simulation environment it is modeled with a standard C++ class. Furthermore, there is terminator components which are used for debugging and misrouting detection during simulation.

5.2 Architectural Components

The architectural components of the NoC simulation environment are implemented with SystemC modules (`sc_module`) and the communication between them uses standard SystemC ports (`sc_port`) as well as transaction-level communication defined in *TLM 1.0*. The components and their connections are presented in Figure 5.1 which represents a 2x2 mesh-shaped NoC. The topology and size of the simulated architecture are fully configurable. The structures of the simulator components are presented below. Each component has operational functions to model the actual run-time behavior, as well as reporting functions to collect the simulation results.

The simulation environment has multiple clocks to control different features. There are four independent clocks to control the network components: a core, a router, a link and a monitor. There are also two slow clocks to control the simulation process: a report clock (`report_clk`) and a configuration clock (`conf_clk`). These are used to trigger simulation data collection and simulation parameter reconfiguration during simulation.

5.2.1 Router

A router is a network component which forwards traffic in a network. The task of a router is to direct incoming packets in an appropriate direction. The router has multiple input and output ports through which it is connected to other routers and cores. Structure of a router was presented in Figure 2.1 on page 6. The three main parts of a router are switch, routing logic and arbitration logic. The routing logic controls the switch to

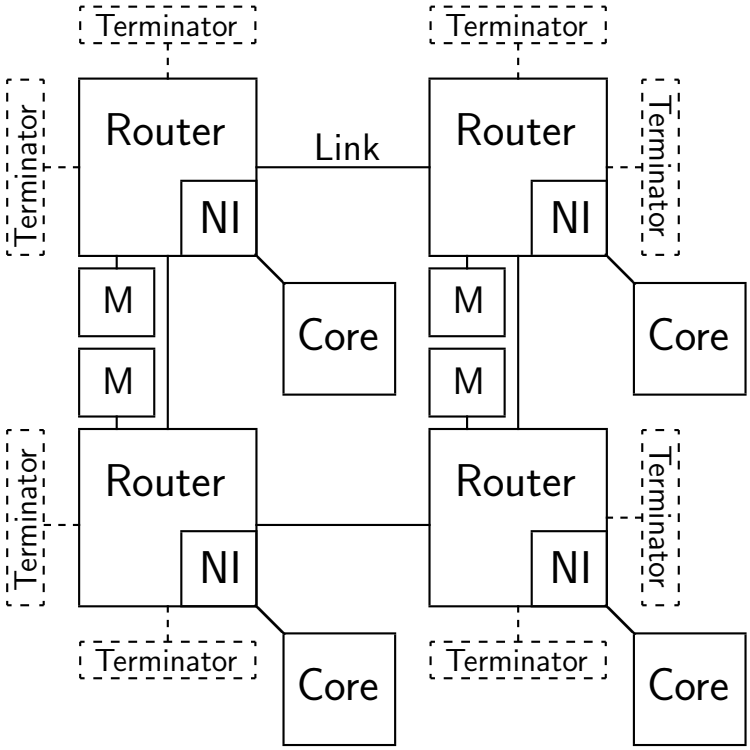


Figure 5.1: Structure of a 2x2 NoC as modeled in the simulation environment. NI denotes network interface and M is integrated monitor and probe. In distributed monitoring all the monitors are similar, otherwise there are central or cluster monitors and other monitors are simple components which send, receive and store the monitoring information.

forward incoming packets to correct output ports and the arbitration logic decides in which order the incoming packets are forwarded.

The router model is designed for mesh networks and so it has five ports, four for traffic to and from the neighboring routers and one for traffic to and from the local core. The number of ports can be modified. There is a FIFO (first-in-first-out) buffer in each input port and a centralized reroute FIFO for packets which cannot be routed at the first attempt but can be rerouted later. There is also input FIFO buffers for special packets: monitoring packets and mapping packets. These special FIFO buffers have implemented to enable packet type prioritization. If the routing fails constantly the packet is dropped, which means that it is removed from the network and not delivered to its destination. The router should notify the sender when dropping a packet. However, the reporting feature is not implemented in this simulation environment. The packet dropping typically happens in severe situations where routing is inhibited due to permanently faulty network resources making a destination unreachable. Sizes of the FIFO buffers are customizable as well as the packet lifetime before dropping. The

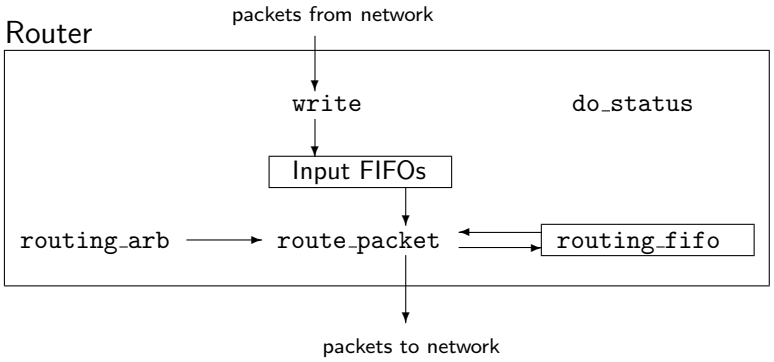


Figure 5.2: Structure of a router including the functions other than the report functions.

router model includes several different routing algorithms. The used algorithm can be chosen with the simulation parameters. The implemented routing algorithms have been discussed in Section 4.4.

To enable traffic monitoring there is a traffic monitoring probe whose functionality is implemented partly in router and monitor components. This probe measures the fill rate of router FIFOs real time, scales the measured value to the used scale and delivers this scaled value to the local monitor which is optionally attached to the router.

The routing algorithm is defined in the router module in the function `route_packet`. The current implementation utilizes a dimension order routing algorithm which is able to customize the routing based on the networks status information and link statuses.

The input and output ports for packets are named as `target_port#` and `initiator_port#` respectively. For monitoring purposes there is

`initiator_port_m` and `target_port_m` to move monitoring packets between the router and the local monitor. Input ports `link_status#` and `min#` are used to transfer neighbor link and router status data from the local monitor to the router. The traffic status of the local router is moved to the local monitor through output port `mout#`. Functions of the router component are presented below. The functions, other than the report functions, and their relationships are represented in Figure 5.2.

`write` Write function is executed when a packet is sent to the router through one of the `target_ports`. The router identifies the type of the packet. If the packet is a monitoring or a mapping packet, it is stored to the corresponding FIFO. Otherwise the packet is stored in the input FIFO of the port from where the packet arrived. If the corresponding FIFO in any case is fully occupied, the router rejects the incoming packet and the sender can try to send it to some other router or resend the packet to the current router later on.

In the system with distributed monitoring, there is a special feature for the monitoring packets. Incoming monitoring packets are forwarded straight to the local monitor

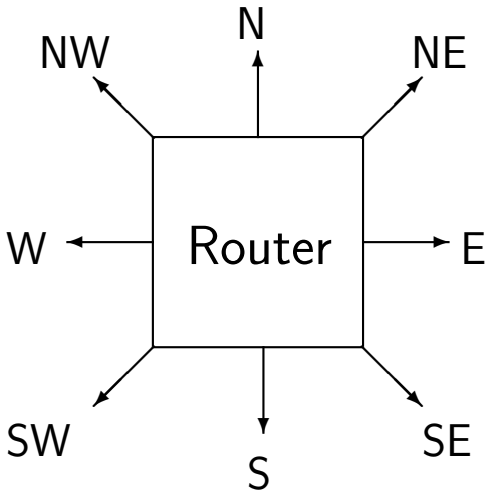


Figure 5.3: Productive routing directions. Letters N, S. E and W denote directions north, south, east and west, correspondingly.

and the packets, sent by the local monitor, are automatically sent to all neighbor routers.

`routing_arb` Routing arbitration function is sensitive to the router clock. The function takes packets from the FIFOs and calls the `route_packet` function to route packets forward. If the routing was successful, the function would remove the packet from the FIFO and increases the `processed` variable. If the routing failed, the function puts the packet in the reroute FIFO or if the packet has reached the maximum lifetime (defined with variable `lifetime`) the packet is dropped and the variable `dropped` is incremented by one. The arbitration algorithm goes through the FIFOs one after another so that every FIFO has equal opportunities to get the packets routed forward.

The usage of separate FIFOs for mapping and monitoring packets makes possible to prioritize these packets over data packets. Reroute FIFO is also prioritized over input port FIFOs when the reroute FIFO is fully occupied. This is done to prevent the blockage of the router. The simulation environment is configured so that the FIFOs are prioritized in the following order: 1) monitoring packet FIFO, 2) mapping packet FIFO, 3) reroute FIFO, 4) input port FIFOs.

`route_packet` The `route_packet` function does the routing decision and tries to send the packet to this direction. After the routing the function returns the result of the routing procedure, which are 1) routing succeeded, 2) routing failed, reroute packet and 3) routing failed, drop packet. Other results are marks of errors.

In the first phase the function determines the productive directions in x- and y-dimension, which means the direction which leads towards the destination. The routing algorithm can be also configured to use non-productive directions in problematic situations.

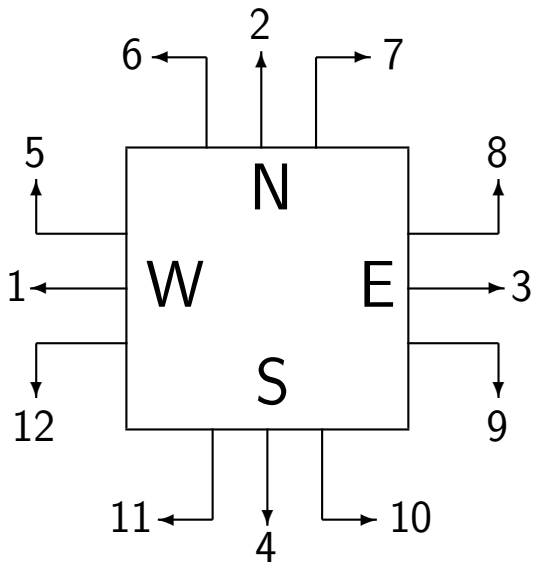


Figure 5.4: Routing directions. Letters N, S. E and W denote directions north, south, east and west, correspondingly.

In the second phase the function defines the routing direction. There is nine possible directions: *local*, *north*, *north-east*, *east*, *south-east*, *south*, *south-west*, *west* and *north-west* (see Figure 5.3). The intermediate direction (north-east, south-east, etc.) can be reached through two output ports, in these cases north or east, and south or east, respectively, as shown in Figure 5.4.

In the third phase, the algorithm compares the traffic and link statuses and tries to send the packet in the direction which it finds the best of the alternatives. The direction options are presented in Figure 5.4. Several routing algorithms have been implemented which are all based on the basic dimension order routing. The algorithm is selected using variable `algo`. The implemented algorithms include versions of minimal and non-minimal dimension order routing. [14]

`do_status` The `do_status` function calculates the current status of the router. The status is based on the number of packets on the rerouting FIFO, `router_fifo`, but can be configured to take the other FIFOs in the account too. The status is scaled to the used status granularity scale. The status data granularity is discussed in Section 6.3.1. The current status of a router is calculated using equation

$$Status_{router} = Fifo_{occ}\frac{Granularity}{Fifo_{max}} \tag{5.1}$$

where $Fifo_{occ}$ is the current number of packets in the FIFO buffer or buffers, $Granularity$ is the used status data granularity and $Fifo_{max}$ is the maximum number of packets which can be stored to the measured FIFO buffers. This status is sent to the monitor through

port `mout0`.

`reset` A reset function is implemented to enable the possibility to reset a router during a simulation. The reset function empties the FIFO buffers.

Reporting Functions Report functions `get_processed` and `get_dropped` are used to collect statistical data in the end of a simulation. The functions return the total numbers of processed and dropped packets during a simulation, respectively. There is also a FIFO related reporting function `get_fifosize` which returns the number of the packets which are stored to the `router_fifo` at that moment.

5.2.2 Network Interface

A network interface (NI) is an interface between a core and a router. The purpose of a network interface is to wrap the data in packets before sending the data to the network and unwrap the data from received packets before delivering the data to the local core.

Due to lack of fully functional cores the NI of the simulation environment has only the necessary functionality implemented. The component has a `write` function which is executed when a packet is sent to the NI and it directs the packet to the core or to the network. At this point, the simulator does not include specific network interface features but the NIs are combinations of two modules which move the packets to and from the cores. The packet wrapping and unwrapping features can be later added to the NI if fully functional cores are used.

5.2.3 Link

The model of a link in the NoC simulation environment is unidirectional. Links are used in bunches of two in between two routers, one in each direction. The link is a simple component which forwards the incoming packets. A link can be individually set in unusable or faulty state to disable it temporarily or model faults in the network. Unusable and faulty states are similar from the functional point of view. In both states the link is disabled and the data are not transferred through the link. A link has a variable `direction` which shows the direction of a link in the network. When a packet arrives at a link the link calls packet function `set_sourcedir` with its own direction as a parameter. The direction of the previous link on the journey of a packet is stored to a packet to be able to prevent U-turns in routing. The functions and their relationships of a link are represented in Figure 5.5.

`write` Packets are sent to a link by calling the write function. The function stores incoming packets to a FIFO buffer `link_fifo` which is of type `queue`. The size of the FIFO buffer is parametrized and can be adjusted with the simulation parameters.

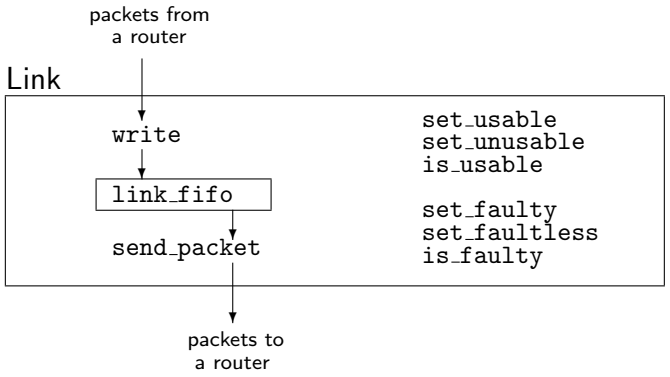


Figure 5.5: Structure of a link including the functions other than the report functions.

`send_packet` A packet sending function is sensitive to the positive clock edge of the link clock and sends a packet from the `link_fifo` once in a clock cycle to the next router. The delay of a link can be adjusted with the link clock.

Usability and Faultiness Functions A link can be set in faulty or unusable state when it cannot be used for data transfer. The link has an built-in model of a *link usability monitor* so that the usability of the link can be read from port `link_status`. The link has functions `set_faulty`, `set_faultless`, `set_unusable` and `set_usable` to modify usability and faultiness. Usability and faultiness are modeled separately so that a link can be set in unusable state temporarily while the faulty state is permanent. When a link is faulty it is always also unusable, but a link can be unusable without being faulty. This has been implemented to be able to separate temporary and permanent unusability. There is also functions `is_usable` and `is_faulty` which return the status of the link.

`reset` The reset function is sensitive to the configuration clock (`conf_clk`). The function clears the `link_fifo` and puts the link to the usable and faultless state.

5.2.4 Core

Cores are used to model computational cores and memories which are connected to the network through network interfaces and routers. A core sends packets to and receives packets from the network. Computational features are not modeled in this simulation environment but the cores are used to generate traffic which imitates real traffic schemes on a Network-on-Chip. The computational load of a core is modeled using core load status values `baseload` and `additionalload`. The computational status of a core is calculated using equation

$$CoreLoad = BaseLoad + AdditionalLoad. \tag{5.2}$$

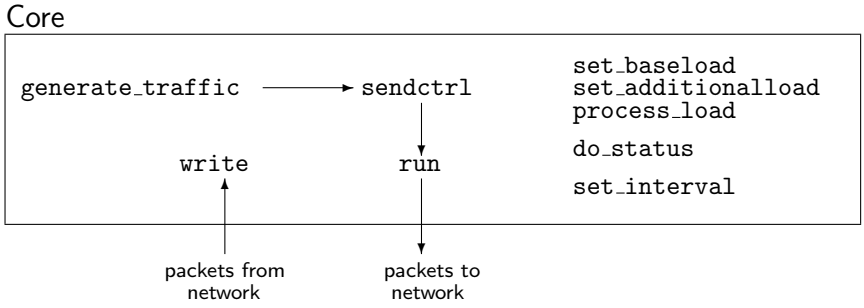


Figure 5.6: Structure of a core including the functions except the report functions.

The base load models static load while additional load can be changed during the simulation. The core load is utilized especially when task mapping features are simulated. The load values are set using functions `set_baseload` and `set_additionalload`. A function `process_load` calculates the total coreload from the base load and additional load values. A process `do_status` calculates the current core status using equation

$$CoreStatus = CoreCapacity - CoreLoad. \tag{5.3}$$

Core capacity is the maximum capacity of a core. In this context it is an abstract concept which is related to the core load. The core status is delivered to a monitor component for monitoring purposes especially for task mapping. The functions, other than the report functions, and their relationships are represented in Figure 5.6. A core has three different modes to send packets. In the first mode a core sends a data packet to a randomly chosen destination once in a period which is defined with a parameter `interval`. The interval is run-time reconfigurable using function `set_interval`. In the second mode a core sends multiple packets to a receiving core. The receiver is chosen randomly and packets are sent to the receiver once in an interval during specific time. The receiver core is selected randomly once in a reconfiguration clock cycle. The two first modes can be used together so that a function `sendctrl` randomly chooses which mode is used. The third mode is related to the core load modeling. A packet is sent once in an interval similarly as in the first mode. However, the used interval is based on the current core load value. The receiver core, which in here is called as a slave core, is set using function `set_slave`. If the third mode is used, the slave core replies to every packet sending the packet back to the original sender. The remaining functions of the core component are presented below.

`generate_traffic` Generate traffic function is used to generate destination addresses for packets which are sent to the network. Variable `traffic_pattern` defines which traffic pattern is used. The different traffic patterns are presented in Section 5.4.3.

`run` Run function controls the operation of the core module. The function is sensitive to a positive edge of the core clock. The function generates packets, and depending on the used mode, defines the destination addresses and sends packets to the network.

`write` Write function is executed when packets are written from the network to the core. The function puts the received packet to a `received_packets` vector, increases the counters of received packets, latency and hops, `rec_count`, `total_latency`, `total_hops`.

Report Functions The core component has several report functions. `get_hops()` returns a sum of hop counts of all the received packets in a core. Hop count is the number of routers which a packet goes through during its route from a sender to a receiver. `get_latency()` returns the sum of transfer latencies of all the received packets. `get_rec_count()` returns the number of received packets in a core so the average hop count and latency values can be calculated the results from `get_hops()` and `get_latency()` by dividing them with the result from function `get_latency()`. A function `get_tried` returns the number of packets which have been tried to send from the core. However, if the network is highly congested the sending could not be possible, the actual number of sent packets is returned by function `get_sent`. The core also has a function `get_load` which returns the current core load of the core. `reset_counters` function resets all the statistic values and packet counters.

Mapping Functions The core component includes functions to model task mapping process which is presented later in Chapter 7. `do_mapping` function is used to initialize task mapping process. It calls `send_mapping_packet` function with desired parameters, generates a mapping packet and sends it to a currently best core in the known neighborhood. The best neighbor is defined by the monitor component. When a mapping packet is received at a core a function `receive_mapping_packet` is executed. The function does a mapping decision, and depending on the decision, calls functions `set_additionalload`, `send_mapping_packet` or `return_mapping_packet`. The `return_mapping_packet` function sends the mapping packet back to the original initiator of the mapping process.

5.2.5 Monitor

The monitors are used to observe the functionality and status of the system. The monitor component in our NoC simulation environment includes both a probe to collect the monitoring data as well as the actual monitor to process the collected data. The monitor component can be also configured to act only as a probe or a monitor. This is useful for instance when analyzing centralized monitoring structures [55]. The monitors communicate with each other using shared or dedicated resources. The NoC simulation environment includes both implementations. When shared resources are used, the monitors send packets in the data network. When dedicated resources are used the resources

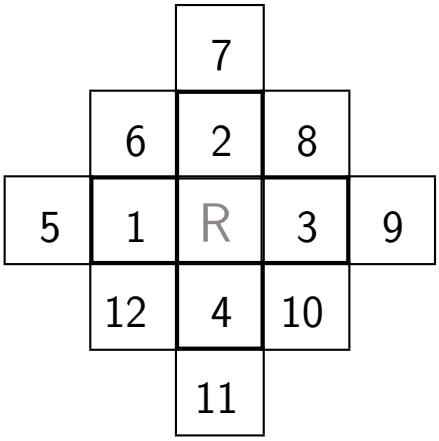


Figure 5.7: Indexes of the neighbors. R denotes the local router of which neighbors are indexed.

for the monitoring network are typically limited. Therefore, the serial communication is utilized. The dedicated resources are not separately implemented in the simulation model but their functionality is modeled using the existing models of links and routers. When the monitoring traffic is switched to serial mode the routers handle the monitoring packets so that they do not interfere with the data packets and the utilization of the monitoring data are delayed with time corresponding the delay of corresponding serial transfer of the data.

Monitor components have input ports for collected status data and outputs to deliver the data to the local router and core. A monitor component has four `lstin#` ports, an `in0` port and a `core_in0` port for probed data from links towards other routers, a local router and a local core, respectively. The input ports also include a `target_port` which can receive monitoring packets which include network status information. Output ports `out#`, `lstout#` and `core_out#` are used to deliver neighbor router, link and core status information to the local router. The indexes of the neighbor routers are presented in Figure 5.7 while the indexes of different output directions were presented in Figure 5.4 on page 52.

In the NoC simulation environment there is a monitor component attached to every router regardless of the used monitoring structure (see Section 3.2). In distributed monitoring every monitor has similar functionality while in clustered and centralized structures there are few cluster monitors or a central monitor, respectively, while other monitors simply store the received monitoring information to be used by the local router. In these cases the monitoring components can be configured to work as monitor, probe or both.

The collected status data are stored in arrays. The link statuses are stored in an array `linkstatus` and the network status data in arrays `status_i` and `status_o`. Received statuses are stored to `status_i` array. The monitor can do some processing and store the

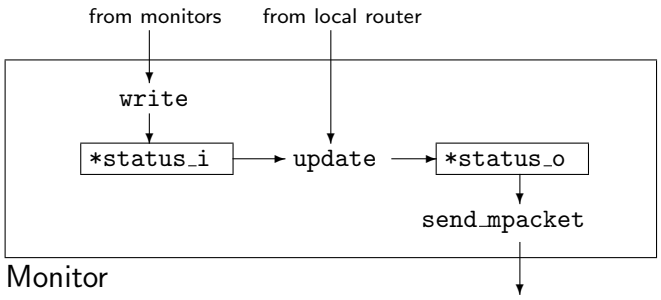


Figure 5.8: Structure of a monitor including the functions other than the report functions.

processed values to the `status_o` array before they are utilized to optimize the routing in the local router. There are also arrays `corestatus_i` and `corestatus_o`, for the core load values. The variable `m_counter` is used to adjust the monitoring packet send interval. The functionality of monitor components for different monitoring structures are presented below. The simulation environment includes three separate models of monitors to be used in distributed, centralized and clustered monitoring structures. The core load monitoring features are implemented only in the distributed monitor component. The functions of a monitor component, other than the report functions, and their relationships are represented in Figure 5.8.

Distributed Monitor

In a distributed monitoring structure a monitor is connected to every router in the network and all the monitors have similar functionality including both monitor and probe features. A distributed monitor receives link and traffic status information from the local router and core load information from the local core. It provides the collected information to the neighbor monitors. That way the information spreads over the network.

`write` A `write` function is executed when a neighbor monitor sends a monitor packet to the monitor. The function reads the data from the packet and stores data to the `status_i`, `corestatus_i` and `linkstatus` arrays.

`update` An `update` function reads the statuses of the local router and core to the arrays `status_i` and `corestatus_i`. The function also calculates processed status data from the `status_i` and `corestatus_i` values and stores them to the arrays `status_o` and `corestatus_o`. The status data processing is done following the guidelines of an utilized status data processing algorithm. The algorithm can, for instance, calculate a status value of a component as a weighted average of the component's own and its neighbors' statuses. These status values are written to the output ports `out#` and `coreout#`. The `update` function also writes the link status values from the `linkstatus` array to `lstout#` ports. These values are not processed.

`send_mpacket` A `send_mpacket` function generates a monitoring packet once in a status data update interval. The length of the interval can be defined as a static duration with the variable `m_counter`. The length of the interval can be also based on the variation in the status values. The current status value is compared to the previous status value and to a threshold value. This threshold defines the required difference between current and previous status until a new monitoring packet is generated and sent. The status update intervals are discussed later in Section 6.1. The `send_mpacket` function stores the neighbor status data and link status data to the packet and sends it to the router which sends it to all its neighbors.

Report Functions The report functions of a monitor component include `get_sentmpackets` and `reset_counters`. Former function returns the number of sent monitoring packets and the latter function resets the monitoring packet counter.

Central Monitor

The central monitor module is used in centralized monitoring structures. There is one central monitor in the system where the statuses of all the links and routers are stored. Monitors, which are attached to each router in the network, operate as probes. These probes send their local router's status to the central monitor which processes the collected information and delivers it to the other routers. Simple probe modules send, receive and store the monitoring data at the routers but do not do any processing. In the simulation environment the central monitor and the probes are modeled using the same monitor component which is configured for different tasks. This operation mode is selected using variable textttmode (0: local monitor, 1: central monitor). The address of the central monitor is stored to variable `m_addr` in every probe. This is required when sending observed status data to the central monitor.

`update` In the local monitors working as probes the `update` function reads the current statuses from local network components so that they can be sent to the central monitor. In the central monitor mode this function also does the status data processing. It reads the received raw status data values from the `status_i` array, does the status data processing and stores the processed values to the `status_o` array. The status data processing is based on the utilized monitoring algorithm. In both modes the function also writes the network and link status data from the arrays to the output ports `out#` and `lstout#`.

`send_mpacket` In the local monitor mode the `send_mpacket` generates a monitoring packet and stores the traffic status of the local router and the statuses of the outgoing links to the packet and sends it to the central monitor. In the central monitor mode this function sends a monitoring packet to a router and its local monitor one after another. After the sending the function calculates the address of the next router and uses this address during the following cycle. During every cycle the function takes the

corresponding network and link status values from the arrays and does the required calculations before storing them to a packet which is sent to a router.

The functionalities of the `write` and the report functions are identical to the corresponding functions in a distributed monitor.

Cluster Monitor

The cluster monitor is used in clustered monitoring structures. There is a cluster monitor in every cluster and simple monitors attached to all the other routers to send, receive and store network information. The simple monitors operate as probes. They send the statuses from the routers to the cluster monitors which store them, exchange the information with other cluster monitors and deliver the data to the simple monitors in its own cluster.

A variable `m_addr` defines the address of the cluster monitor while variables `nm#` are used to store the addresses of the cluster monitors of the neighboring clusters. `sendingmode` variable defines if the monitor is 1) sending monitoring data to the routers in its own cluster or 2) sending monitoring data to the cluster monitors of the neighboring clusters.

`update` In a local monitor the `update` function reads the status values from local components so that they can be sent to the cluster monitor. In the cluster monitor the `update` function works similarly as in the central monitor. When defining the statuses of routers on the borders of the clusters the algorithm takes into account the status of the neighbor cluster behind the border. The `update` function also takes care of putting status data to the output ports `out#` and `lstout#` so that they can be read and utilized by other local components.

`write` When a monitoring packet is sent to a cluster monitor the `write` function is executed and the network and link status data are written from the packet to the corresponding variables and arrays in the monitor. The status values of the neighboring clusters are also stored to the `status_i` array.

`send_mpacket` The `send_mpacket` function in a cluster monitor has five different sending modes (defined with variable `sendingmode`). Mode 1 is for sending monitoring packets to the routers in its own cluster. Modes 2-5 are for sending monitoring packets to the cluster monitors in the neighboring clusters in west, north, east and south. In mode 1 the cluster monitor operates similarly as a centralized monitor. In a simple local monitor the `send_mpacket` function sends the router status and the nearest link statuses to the cluster monitor.

5.3 Simulator Components

Simulator components in the simulation environment are not actually implemented on real Network-on-Chip realizations but necessary to model the system on a simulation environment. They can be abstract concepts or components which are implemented for debugging and monitoring the simulation environment.

5.3.1 Packet

In real implementations a packet is a concept how the data have been structured when it is transferred in a network. In the simulation environment a packet is a simple C++ class which represents the data packets that are moved in the network. Main information stored in a packet includes source and destination addresses and data payload. This information is stored in variables `source` and `dest`, and `data#` arrays, respectively. There is also a variable `packet_type`, which indicates the type of the packet. The type can be data packet (0), monitoring packet (1), reply packet (2) or mapping packet (3). The reply packets are used when the traffic pattern utilizes automatic replying to received packets. The packet type can be read using function `get_type`.

Initialization The initialization function `initialize` is executed when a packet is created. The function sets the current time as the send time of the packet (variable `sendtime`) and resets the hop count calculator. The hop count calculator counts the transfer distance of a packet and it is incremented every time the packet is routed by calling function `add_hop`.

Address Functions Functions `get_source` and `get_dest` return the addresses of the sender and destination of the packet respectively. Every time a packet goes through a link the link sets its direction using function `set_sourcedir` (possible directions: towards north, south, east, or west). A router can read this value using function `get_sourcedir` when it needs to know from which direction a packet has arrived to a router.

Data Functions The packet can have several arrays which are used to store data (`data#`). These vectors can be written and read using functions `set_data#` and `get_data#`.

Mapping Functions When a packet is used as a mapping packet, there are specific variables including variables for caused load, mapper's address and id, and mapping distance. The `caused_load` variable represents the size of task which is mapped with the mapping packet. The unit of the task size is left undefined at this point. Mapping distance is the distance in hops from the original mapper. A packet includes functions to read and manipulate these values.

Report Functions When a packet arrives at its destination the receiver calls function `set_rectime`. This function sets the current time as a receiving time of the packet. The time of sending and receiving of a packet can be read using functions `get_sendtime`, `get_rectime`. There is also function to read the total latency of a packet, `get_latency`. To read the total hop count of a packet, there is a function `get_hopcount`.

5.3.2 Terminator

A terminator is a component which is implemented only for debug purposes. Terminators are connected to the output ports of the routers on the borders of the network so that packets which are going outside of the network end to the terminator components. The terminator is not implemented to real NoC implementations but it can be used detect misrouted packets in the borders of the network.

5.4 Simulator Functionality

The simulation flow starts from NoC architecture configuration. The simulated Network-on-Chip is structured from separate components which are connected together. The components are configured with desired parameters and the simulation is started. After the simulation is finished the simulation environment executes result collection functions and the results are stored for analysis.

The architecture of an NoC is defined in top level class `noc`. It includes representation of the NoC architecture, its components and how they are connected. The `noc` class is controlled through a `main` class. All the simulation parameters are defined in the `main` class.

5.4.1 Architecture

The `noc` class generates vectors where the components of the NoC are stored. The components and the NoC itself are C++ objects. The NoC class connects its components to each other. At this time the simulation environment includes only the mesh topology but the implementation of other topologies is not limited.

The NoC class has functions to generate errors to the system, to control task mapping simulations and to collect results from the components. For the cases when a static packet sending interval is used, there is a function `set_interval` which is sensitive to the report clock and changes the packet interval based on an interval plan which defines the intervals used in different phases of the simulation.

`generate_errors` This function generates link faults to the system in random locations. The function puts the chosen links in faulty state so that they are not usable. The randomization can be overridden so that the faults can be put to certain locations without randomization. The number of errors is defined with the variable `e` in the NoC parameters. The function is sensitive to the reconfiguration clock (`conf_clk`). Each

time the `generate_errors` function is executed it resets the links and randomizes new locations for the link faults in the system.

`mapping_control` This function initiates task mapping process during a simulation. The function is sensitive to the router clock and the designer can choose at which clock cycle a task mapping process is initiated. Two parameters in the process are 1) the address of the mapper core and 2) the size of the task.

Reporting Functions The NoC class includes the reporting functions to return key figures as total latency, average latency, processed and received packets, packets tried to sent, packets actually sent, sent monitoring packets, total hop count, average hop count and dropped packets. The statistics are printed out using functions `init_report` and `report`. `init_report` is called from inside the `report` and it calls the report functions of the components. The initialization function collects the results from the NoC components and the report function prints them out or saves in a file.

5.4.2 Top Level Control

The top level control of the simulation environment is implemented in `main` class which is a mandatory part of every C++ implementation. The class has all the simulation parameters as variables and uses these when creating NoC objects from the `noc` class. The `main` class includes a loop where multiple NoCs can be generated with varying parameters. The top level class initiates the simulation and when it has been completed, collects overall results from the simulated system.

5.4.3 Traffic Pattern

The traffic on a simulated NoC can be based on core load values or a traffic pattern. The NoC simulation environment has three traffic patterns which are based on randomization. The simplest pattern is a fully random traffic pattern which randomizes the packet destinations among all the cores in the network. A new destination is randomized for every sent packet. A weighted random traffic pattern is adjusted so that a one third of traffic is between neighboring cores, another third between neighbor's neighbors and the last part between all the other cores in the network. This pattern roughly imitates a traffic pattern in a real NoC implementation where most of the traffic takes place between cores near each other.

The third implemented traffic pattern is a two-level pattern which includes uniform random traffic and varying hot spots each of which sends a relatively large number of packets to a single receiver during a certain time interval. A relatively small number of cores operate as hot spots simultaneously and send packets to a statically chosen receiver cores. At the same time, other cores are sending relatively smaller amount of traffic to random destinations. This two-level traffic pattern imitates real applications where most of the traffic takes place between certain cores at a time. It is aimed for even

more realistic performance simulations. The traffic management simulations, presented in this thesis, are done using this two-level traffic pattern.

5.4.4 Fault Injection

During simulation the network links can be set faulty. In Network-on-Chip simulations the fault information can be simplified by using only the information on faulty links and representing other faulty components by marking the links around these components to be faulty. In the simulation environment the number of faults is defined by the user and the simulator places the faults randomly in the network. The simulation is executed several times with different fault patterns and the results are averaged from the original simulation results. This procedure gives overall insight of the operation of the system when parts of the network are faulty.

5.4.5 Task Mapping

To enable run-time task mapping the simulation model of a monitoring packet was improved to contain required core load status information and a counter for mapping decision attempts. The simulation environment is able to report the load of separate cores in any report point (defined by the report clock) during the simulation. It also reports the successful task mapping including the IDs of the mapping initiator, assignee and the number of performed mapping decision attempts.

The simulation environment makes it possible to adjust initial mapping schemes, simulate task mappings and collect results concerning the task mapping processes. Simulation reports include detailed descriptions of each mapping initialization, decision attempt and executed mappings. The details include identifiers of the original mapper and the core where the mapping decision attempt has been carried out. Number of decision attempts is also presented. The simulation environment is able to illustrate how the core load is distributed over the system.

5.5 Summary

The SystemC based Network-on-Chip simulation environment was presented in this chapter. The simulation environment includes models of different NoC components and it can be configured to simulate features of distinct NoC architectures.

In the following chapters the simulation environment is utilized to model an NoC with the presented DCM structure (see Chapter 4). In Chapter 6 the simulation environment is used to model task management features of the DCM structure while in Chapter 7 a case study concerning distributed lightweight task mapping is examined.

The simulation environment is implemented modularly so that in future works it can be utilized to model Network-on-Chip architectures with new and varying features comprehensively.

Chapter 6

Traffic Management

The dynamically clustered Network-on-Chip monitoring structure, which was presented in Chapter 4, is mainly targeted and designed for traffic management purposes. Traffic management aims to balance the utilization of network resources so that maximum performance can be achieved. Essentially, traffic management tries to avoid the development of highly loaded traffic hot spots in the network. Traffic management should also keep the network functioning even though if there are some faulty components. Traffic management related issues are studied in this chapter.

The results presented in this chapter are based on simulations which were carried out using the simulation environment presented in Chapter 5. Exception to this is the basic analysis of network status data diffusion in the beginning of Section 6.2 (Figure 6.6 on page 75) which is based on conceptual analysis and is carried out similarly as the analysis presented in Chapter 4.

6.1 Status Update Interval

A status update interval defines how often or in which circumstances a monitor sends the updated status data to its neighbors. There are two different approaches: *static* and *dynamic*. When a static update interval is used every monitor sends the updated status data to its neighbors after a certain time interval regardless of the changes in the data after the previous update. The only parameter in the static update is the time between the updates ($Interval_{Static}$). The unit the interval is the minimum time between two sent packets from one source. The time interval parameter should be adjusted in a way that the network components have up to date status information but the update traffic does not strain the communication resources too much.

A dynamic update interval is based on the variation of the status values. The monitor sends the up to date status values to the neighbors when the difference between current and previous values is more than a pre-defined update threshold ($Threshold_{Dynamic}$). The unit of the threshold is the same as the unit of the network status values. This threshold is the essential parameter of the dynamic update which is adjusted corre-

spondingly as the time interval parameter in the static update procedure. Static and dynamic status update intervals have certain weaknesses. When the network status is changing rapidly, the static status update misses a fraction of the changes. Surely it can be configured to be short enough to detect every change but then it would cause futile overhead during changeless periods. However, after a predefined time interval the status data will be exactly up-to-date for a moment. The weaknesses of the dynamic status update interval are opposite to that of the static update interval. When the network status is changing slowly, the status values can be slightly out of date for a relatively long time before the status value variation reaches the update threshold and will be updated.

The weaknesses can be compensated by combining these two procedures to an *enhanced dynamic* status update interval. This interval type has two parameters which are familiar from the static and dynamic intervals: time interval ($Interval_{Enhanced}$) and threshold ($Threshold_{Enhanced}$). This method works similarly as the dynamic status update interval but there is also a time interval parameter which defines the maximum

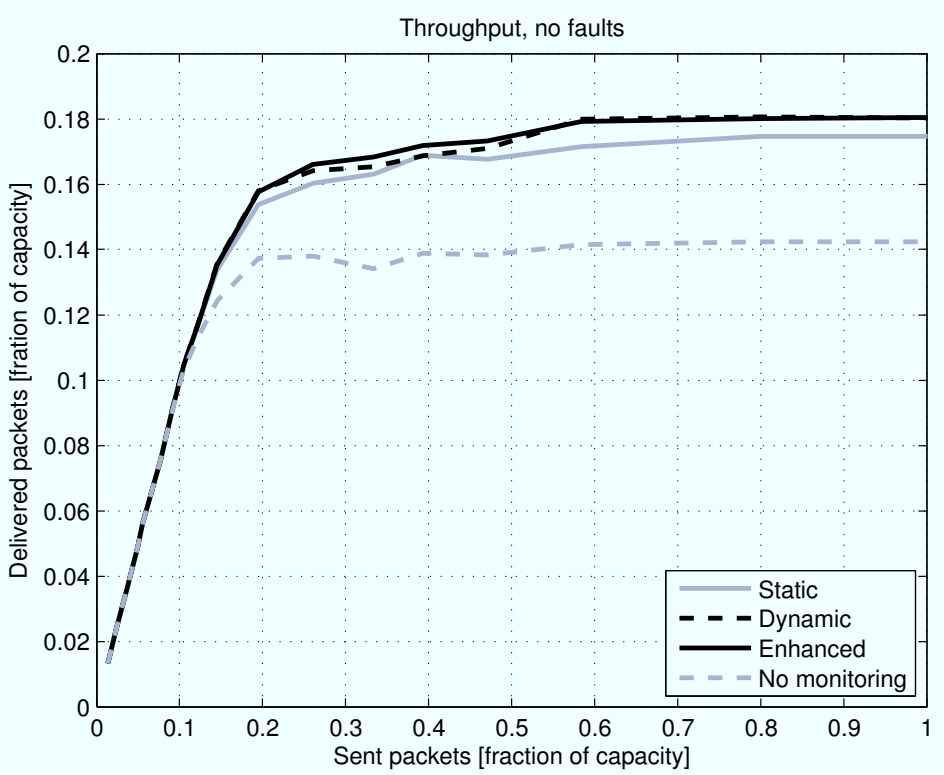


Figure 6.1: Throughput with different status update interval procedures and identical traffic pattern. No faults. $Interval_{Static} = 23$, $Threshold_{Dynamic} = 3$, $Interval_{Enhanced} = 50$, $Threshold_{Enhanced} = 5$

time between two status updates regardless of the variation of these values. When the enhanced dynamic status update interval is used, larger threshold and time interval parameters can be applied than in static or a dynamic update interval. This can be used to decrease the number of monitoring packets while performance is improved.

6.1.1 Status Update Interval Analysis

The influence of the different status update intervals on the network throughput were compared. Figure 6.1 and Figure 6.2 present the throughput of a 64-router NoC with static, dynamic and enhanced dynamic status update interval protocols in faultless network and when 10% of links in the network are faulty, respectively. The throughput of a corresponding NoC without monitoring and routing adaptivity is also presented. The update interval parameters are adjusted so that the numbers of sent monitoring packets during the simulations are at the same range with each other. Status data granularity is 32 which means that the network status values are in range of 0–31.

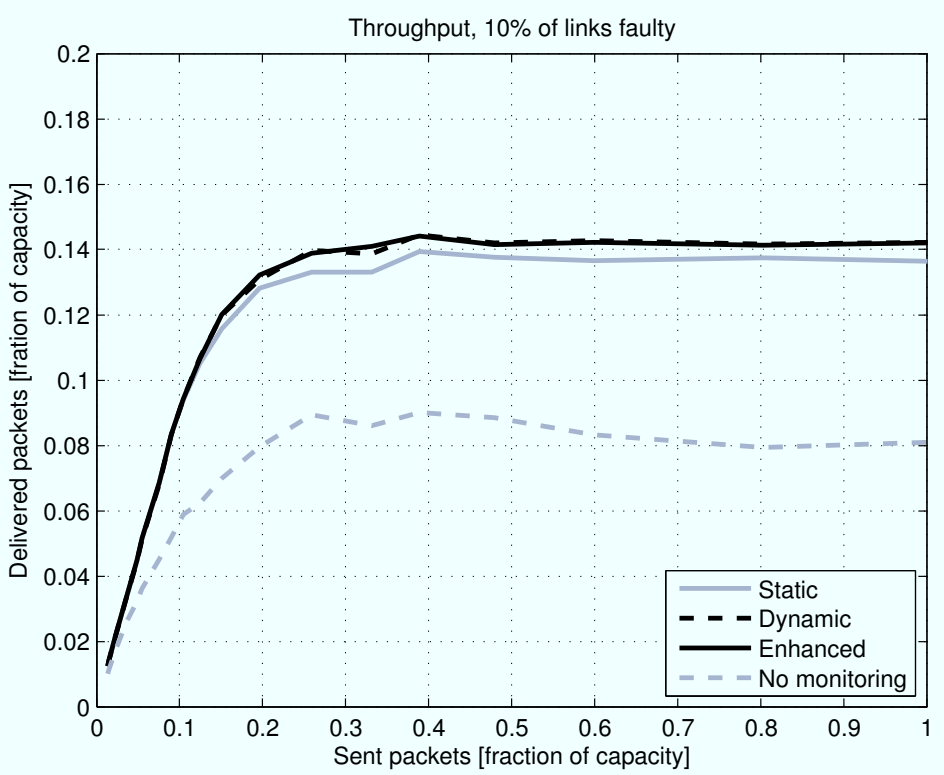


Figure 6.2: Throughput with different status update interval procedures and identical traffic pattern. 10% of the links are faulty. $Interval_{Static} = 23$, $Threshold_{Dynamic} = 3$, $Interval_{Enhanced} = 50$, $Threshold_{Enhanced} = 5$

The obtained performance values with different status update intervals were compared in a point where the throughput has been saturated. The point, where 60% of the maximum capacity of packets were sent during a routing cycle, was used for the measurements. As can be seen in Figure 6.1 and Figure 6.2, the dynamic and enhanced dynamic update intervals improve network performance significantly while the improvement with the static status update interval is slightly lower. In a faultless network the static interval causes 21% improvement when compared to a system without any monitoring. At the same time the dynamic and the enhanced update intervals reach 29% performance improvement. In a faulty network the throughput increase of 70% has been reached using the dynamic and the enhanced dynamic status update intervals. The corresponding performance improvement with the static update interval is 63%. These proposed methods make it possible to gain major improvements in the overall performance especially when parts of the network resources are faulty or unusable. When the results from the faultless and the faulty network are compared, one can note that in the system without monitoring the performance drops 40% when faults are added to the network. The corresponding drop when monitoring with static update interval is used

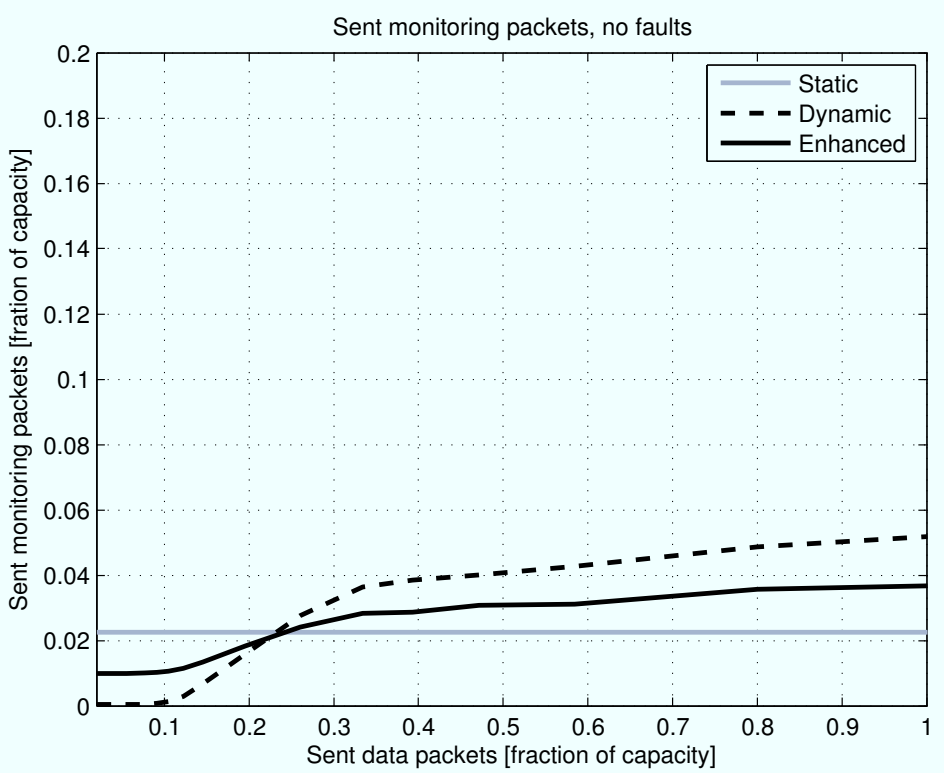


Figure 6.3: Number of sent monitoring packets. No faults. $Interval_{Static} = 23$, $Threshold_{Dynamic} = 3$, $Interval_{Enhanced} = 50$, $Threshold_{Enhanced} = 5$

is 20% and with dynamic or enhanced update interval 21%.

The numbers of sent monitoring packets in these faultless and faulty networks are illustrated in Figure 6.3 and Figure 6.4. As can be noted the number of packets increases with the amount of traffic when dynamic or enhanced dynamic update intervals are used.

The number of monitoring packets is high related to the number of data packets. However, each monitoring packet has the hop count (the distance the packet is transferred in a network) of 1 and they do not require routing decisions but only a simple routing packet recognition when they are moved in the network. Thus, the amount of monitoring packets is negligible in comparison with the obtained performance improvement. One should also note that the number of monitoring packets in a highly loaded network with the enhanced dynamic status update interval is smaller than with the dynamic status update interval although the throughput is equal. This difference implies lower energy consumption in a highly loaded system when the enhanced status update interval is utilized and also shows that the enhanced interval works well even though it uses less communication resources in a highly loaded network.

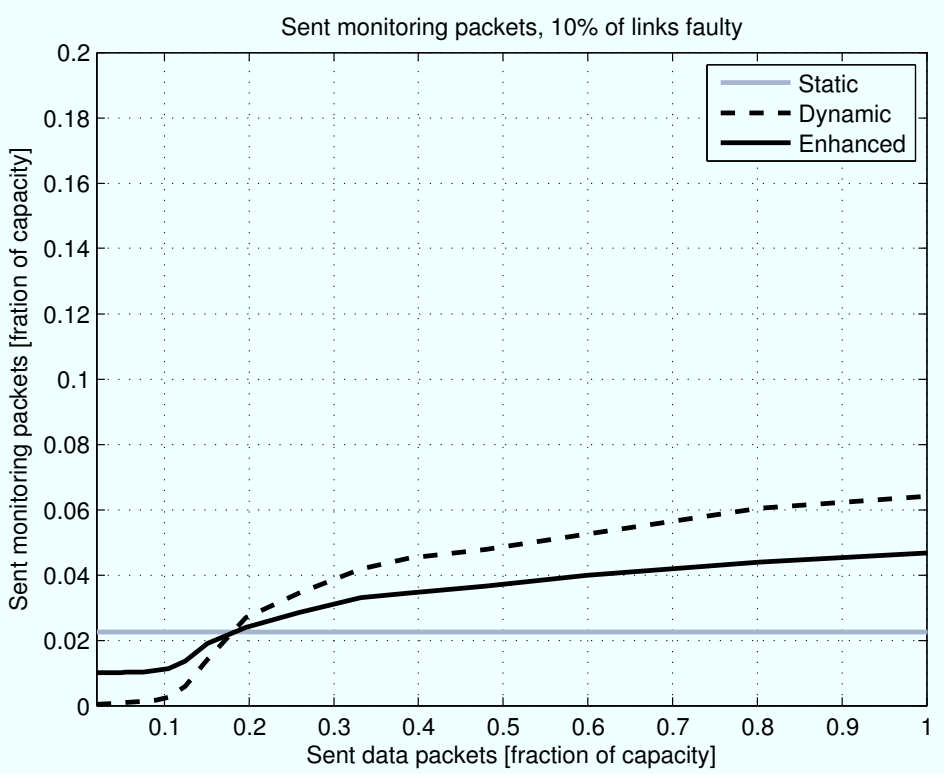


Figure 6.4: Number of sent monitoring packets. 10% of the links are faulty. $Interval_{Static} = 23$, $Threshold_{Dynamic} = 3$, $Interval_{Enhanced} = 50$, $Threshold_{Enhanced} = 5$

Table 6.1: Status update implementation complexity.

	Static	Dynamic	Enhanced dynamic
Cells	99	461 (+366%)	581 (+487%)
Area	1362 μm^2	3780 μm^2 (+178%)	5081 μm^2 (+273%)

6.1.2 Cost of Implementations

The complexities of the different status update interval implementations were analyzed using VHDL models. The status update interval control units were modeled with static, dynamic and enhanced dynamic mechanisms. These VHDL models were synthesized to 90 nm technology and the results are presented in Table 6.1. The modeled part is just a small piece of a monitor and a very small part of the whole system which means that the presented size differences may not be prominent. However, in some implementations a designer could choose a dynamic status update interval to do a compromise between complexity and performance.

The cost of a monitoring system implementation is related to the number and size of required probes and monitor logic and registers in the monitors. In every system, there is at least one probe for each router and registers for their own and their neighbors' statuses. The number of registers in a router is not related to the used monitoring structure so they are not taken into account in this analysis. The probes in the distributed monitoring system are simpler than in the clustered and centralized versions because they do not have to communicate over the network with the monitoring unit. The simple probes could be used also in clustered and centralized systems if there are dedicated communication resources between the probes and the monitors, and if the probe is connected to the router to which the monitor is attached. Furthermore, monitors in distributed systems are simpler because they handle smaller amounts of data.

A monitor in a centralized structure needs registers to store the statuses of all the routers in the network. In clustered systems there have to be registers in monitors to store the statuses of the routers in a local cluster and the overall statuses of the neighboring clusters. A monitoring unit of the distributed monitoring system requires registers for the status of the local router and the statuses of the neighbors. In clustered and distributed systems the number of neighbor statuses to be stored is defined during the design process. Here we assume that the statuses of 12 neighbors are stored (the closest neighbors and their neighbors). If the routing algorithm uses raw (i.e. unprocessed) neighbor status data, a monitor in distributed and clustered structures may use these data from the router's memory and that way the amount of required registers in monitors is decreased.

A cost comparison is presented in Table 6.2. Autonomous probes are probes which can communicate with a monitoring unit over the NoC while simple probes are directly

Table 6.2: Monitoring system cost comparison.

	Distributed	**Clustered**	**Centralized**
Autonomous probes	0	$n - \lceil n/m \rceil$	$n - 1$
Simple probes	n	$\lceil n/m \rceil$	1
Cluster monitors	0	$\lceil n/m \rceil$	1
Simple monitors	n	0	0
Registers in a monitor	$1 \dots (q+1)$	$m \dots (m+q)$	n

n = routers in the network

m = routers in a cluster

q = neighboring monitors/clusters whose overall statuses are stored in a monitor

connected to a monitor with dedicated communication resources. Cluster monitors are monitors that take care of several probes and routers. Those are used in centralized and clustered systems while simple monitors, which just take care of a single probe and communicate with their neighboring monitors, are used in distributed monitoring systems. The required amount of memory in the monitors is defined as the number of registers where one register is able to store the status of one router. The number of required registers is listed on the table. The number of registers in distributed and clustered structures depends on how efficiently the registers of the local router can be utilized for the purposes of the monitor.

Exact cost comparison is impossible without circuit implementations but some estimations of the component sizes and especially of their mutual ratios can be made. At first we define the units; the size of a simple probe: S_P, and the unit cost for monitors: S_M. An autonomous probe is a simple probe with capability to communicate over an NoC. The probe and the communication part of a probe are assumed to be roughly the same size which leads to an assumption that the size of an autonomous probe is $2S_P$. Another assumption is the ratio of the different monitor sizes. The approximated monitor sizes are related to the number of direct connections from a monitor to probes, routers and other monitors. S_M is equivalent to the cost of logic required for one connection, for instance the size of a monitor in the middle of a distributed monitoring system is $6S_M$ because the monitor is connected to a probe, a router and four neighbor monitors. In addition we denote the size of a register with S_R.

By using the above assumptions, we can carry out a cost comparison of monitoring systems for an NoC with 100 routers (Table 6.3). Note that there are no assumptions on the relations between units S_P, S_M and S_R which means that they are not comparable with each other.

The approximations in Table 6.3 show that the distributed implementation is the least complex in terms of probes. The number of registers depends on the used moni-

Table 6.3: Cost of 100-router NoC with different monitoring structures.

	Distributed	**Centralized**
Probes	$100S_P$	$199S_P$
Monitors	$560S_M$	$200S_M$
Registers	$100S_R$...$1096S_R$	$100S_R$

	Clustered (4)	**Clustered (16)**	**Clustered (25)**
Probes	$175S_P$	$191S_P$	$196S_P$
Monitors	$280S_M$	$224S_M$	$208S_M$
Registers	$100S_R$...$284S_R$	$100S_R$...$152S_R$	$100S_R$...$112S_R$

Size of a cluster in parentheses.

toring algorithm but is at least equal to the number of routers and increases when the size of a cluster decreases. The analysis shows that when using distributed monitoring structure it is strongly recommended to integrate the router and the monitor and share the status registers among them. This can be done also when the cluster size is relatively small and raw monitoring data are stored to the status registers of the routers. In large clusters, the neighbor cluster statuses are stored to the routers far away from the monitor, which makes the register sharing difficult. Table 6.3 also shows that the total cost of monitors decreases when the size of a cluster increases.

It has to be noted that in distributed implementations the computation is done in smaller parts by simple components working at the same speed with the larger monitors. That way the distributed system may reach significantly higher performance because the amount of data to be processed in a monitor is substantially smaller, or they can be optimized for low power consumption and area. Furthermore, the distributed monitoring system can be more fault tolerant because the computation is divided into the small monitoring units and each of them is much less critical in terms of overall system functionality than the larger monitors in clustered and centralized systems.

6.2 Status Data Diffusion

The network status data diffusion defines how far the status of a network component (in our analysis router) spreads in the network. A wider diffusion area makes it possible to react to problems early and avoid routing packets to the worst hot spots or faulty areas.

Network information diffusion in distributed and statically clustered network monitoring is illustrated in Figure 6.5. Diffusion of information in a monitoring system is divided into phases so that a phase consists of traffic information collection inside a

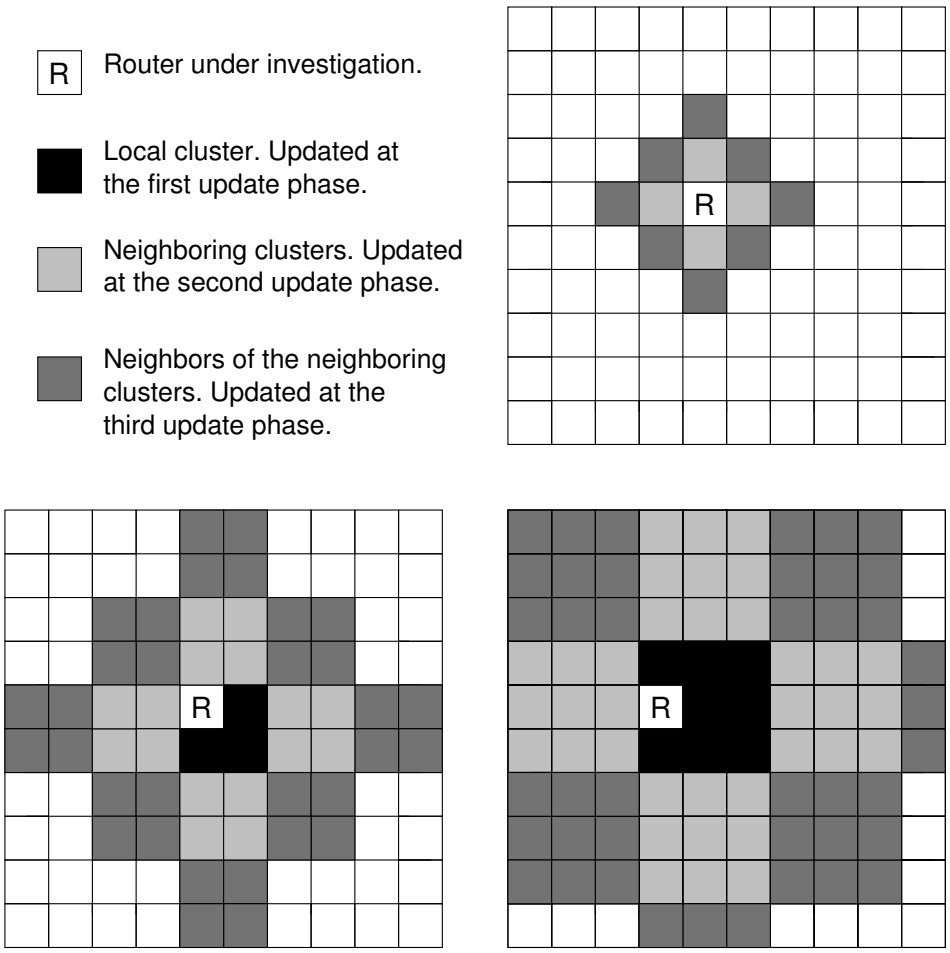


Figure 6.5: Diffusion of network status information.

monitoring cluster, information exchange between neighboring clusters and information delivery to the routers in the local cluster. The local cluster is reached during the first phase, neighbor clusters during the second phase and during the third phase the information spreads to the neighbors of the neighboring clusters. Still, it has to be noted that the durations of the update phases are proportional to the cluster size. The figure shows that a larger cluster size makes diffusion coarser because the information reaches one cluster at a time.

As Figure 6.5 shows, the shape of the monitoring cluster should be the same as the shape of the whole network. This guarantees that the data spread evenly over the network and approaches each side of the network with the same rate. If the cluster size is two, for example, the data diffuse to the other dimension with half the rate. The area or the number of routers where the network status data diffuse during a certain time frame is represented with equation

$$D_{area} = \begin{cases} 0 & \text{when } p = 0, \\ C_{size} & \text{when } p = 1, \\ C_{size} + \sum_{i=2}^{p} 4(i-1)C_{size} & \text{when } p > 1 \end{cases} \tag{6.1}$$

where p is the number of update phases (see Figure 6.5). At the first update phase ($p = 1$) the information diffuses to the local cluster. During the next phases, the data diffuse to the neighboring clusters of the cluster or clusters which were updated at the previous phase. This additional area increases by the size of four clusters during every phase. This is represented with the sum term. The duration of an update process can be calculated using equation

$$t = (p-1)\sqrt{C_{size}} + 4\left\lfloor \frac{\sqrt{C_{size}}}{2} \right\rfloor \qquad \text{when } p > 0 \tag{6.2}$$

where the first term represents the time consumed to move data from one monitor to another while the second term depicts the time used to collect the data from routers and finally to deliver the data to routers on another cluster. The floor function gives the maximum hop count between a router and a monitor in a cluster.

Figure 6.6 shows how fast the information diffuses in the network. The distributed system outperforms other candidates. We can note that the structure with the cluster size of nine is faster than the structure with the cluster size of four. Respectively, the structure with the cluster size of 25 spreads the data faster than the system with the cluster size of 16. This is due to the more optimal structure of the clusters with the sizes of 9 and 25. When C_{width} and C_{height} of a cluster are odd numbers the cluster monitor can be placed exactly in the middle of the cluster which minimized the theoretical average hop counts (H_{avg}) in a cluster. Figure 6.6 also shows that when a message needs to be delivered to the whole system immediately, the distributed monitoring system accomplishes this with the highest rate. This may be useful if immediate system reconfiguration is required.

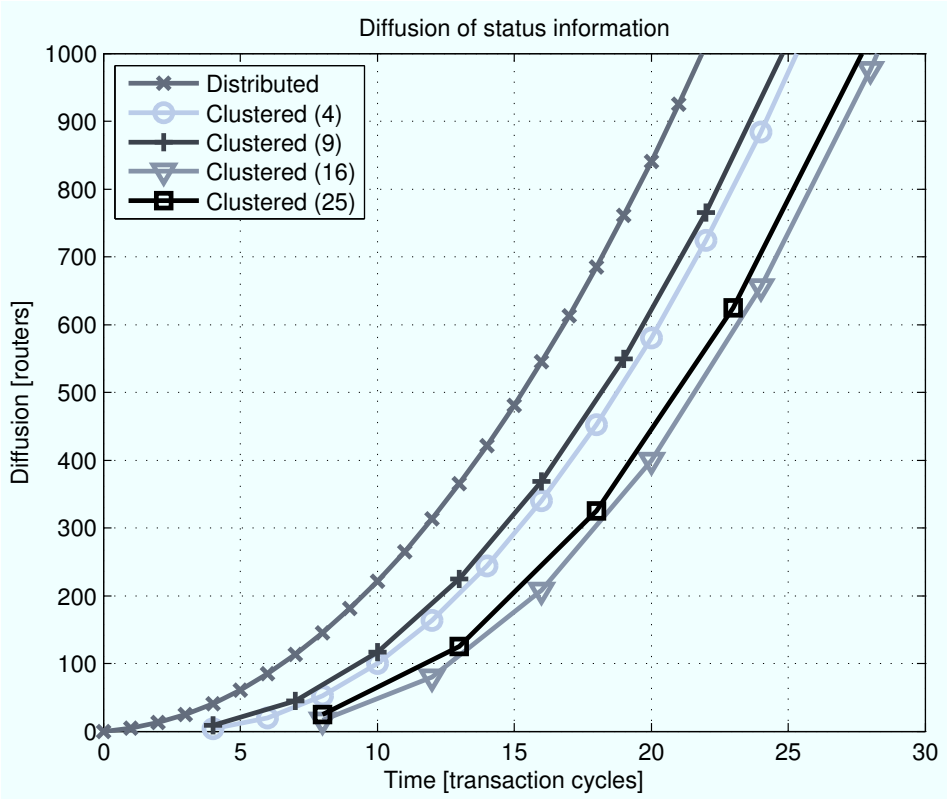


Figure 6.6: Diffusion of network status information as a function of time (size of a cluster in parentheses).

There are two factors which affect the status data diffusion: the size of a dynamic cluster and status data processing. The cluster size defines how far the status data diffuse from a router and the amount of neighbor status data a router has. The cluster size has an effect on the quantity of neighbor status data to be transferred in a monitoring packet. If the size of a dynamic cluster is 5, it contains the neighbors of a router and the router itself. In this case it is enough to send only the router's own status to the neighbors. If the size of a cluster is 13, it also includes the neighbors of the neighbors and then four neighbor statuses should be included in every monitoring packet. In larger dynamic clusters the amount of monitoring data which has to be included in a monitoring packet increases to a level which is not practical. Therefore, we have limited our analysis to the dynamic cluster sizes of 5 and 13.

6.2.1 Additional Status Data Processing

In DCM structure the network traffic status values are based on the load of the routers. When additional status data processing is used, each router status value is based on

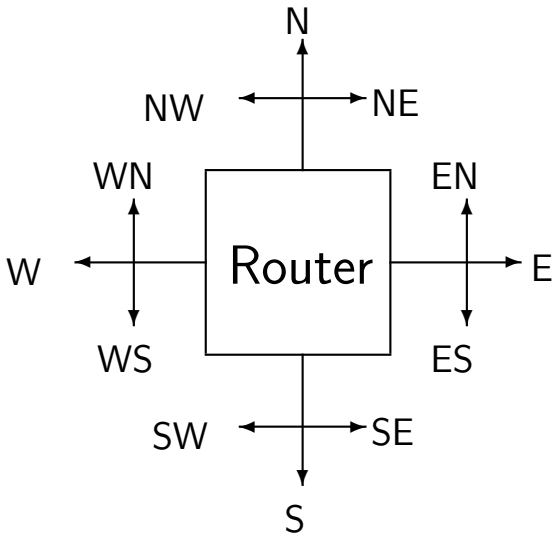


Figure 6.7: Routing directions. The abbreviations N, S, E and W stand for directions north, south, east and west, correspondingly.

the state of the router itself and the state of its neighbors. When the neighbor routers' status is defined using also its own neighbors, the status data diffuse over the network.

A processed status of a router (S), which represents the traffic load in a router and its surroundings, is calculated using Equation 6.3 where $S_x, x \in \{L, N, S, E, W\}$ is the status of a neighboring router in a specific direction (Local router, or a neighbor in North, South, East or West direction).

$$S = \alpha S_L + \frac{1-\alpha}{4}(S_N + S_S + S_E + S_W) \tag{6.3}$$

where α is a factor which defines which part of the router status is related to its own utilization level. When the status data of twelve neighbors is used the statuses of different routing directions are calculated using for example Equation 6.5 and Equation 6.6, where

$$D_x, x \in \{N, S, E, W, NW, WN, NE, EN, SW, WS, SE, ES\}, \tag{6.4}$$

represents the status of a routing direction and N_i where $1 \leq i \leq C_{Size} - 1$ is the status of a neighboring router in a specific index. Other ten statuses are calculated using corresponding equations. The routing directions are represented in Figure 6.7 and the used indexes of the neighbor routers in Figure 6.8.

$$D_N = \beta N_2 + (1-\beta) N_7 \tag{6.5}$$

$$D_{ES} = \gamma N_3 + \frac{1-\gamma}{2}(N_9 + N_{10}) \tag{6.6}$$

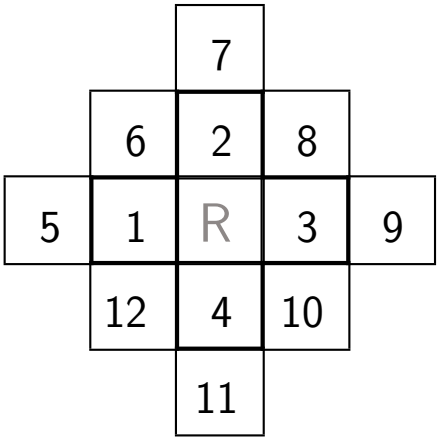


Figure 6.8: Indexes of the neighbor routers. R denotes the local router of which neighbors are indexed.

where β and γ are factors which define at which intensity the status of a neighboring router affects to the status of a direction. Coefficients α, β and γ ($0 \leq \alpha, \beta, \gamma \leq 1$) are in here defined experimentally, but can be individually adjusted to avoid or favor routing packets to certain routing directions. $\beta, \gamma = 1$ when the size of a dynamic cluster is four. Therefore, Equation 6.5 and Equation 6.6 are simplified in that case and the complexity of the system is reduced.

6.2.2 Status Data Diffusion Analysis

The analysis of network status diffusion is presented in Figure 6.9, Figure 6.10, Figure 6.11 and Figure 6.12. The same simulation was executed with two different dynamic cluster sizes without faults and with 10% of network links being faulty. The analysis was not done with larger cluster sizes because the amount of transferred monitoring data then increases to intolerable level, as was described in Section 6.2. If the cluster size is increased another time further the size of it would became 25 and each monitoring packet would include 13 different router statuses. The additional status data processing was used in the cases presented in Figure 6.9 and Figure 6.11. The processing coefficients were experimentally chosen to obtain maximal throughput so that $\alpha = 0.5$, $\beta = 0.6$ and $\gamma = 0.6$. When the additional processing was not utilized (Figure 6.10 and Figure 6.12) the raw monitored status data were used and the statuses of neighbor routers did not have influence on the router status values.

The differences in network performance appear when the throughput has been fully or nearly saturated. The figures show that in a faultless network the performance differences are notable. The proportional differences were measured at the point where 60% (0.6 on the x-axis) of the maximum capacity of packets were sent during a routing cycle. The 60%-point was chosen because it is clearly after the saturation point but still far from

the maximum load.

All the following performance increment percentages are in proportion to the performance of an NoC with similar fault pattern, deterministic routing algorithm and without network monitoring. In a faultless network (see Figure 6.9 and Figure 6.10) the performance increase is 19% when status data processing is used, regardless of the monitoring cluster size. Surprisingly, when status data processing is turned off the throughput increases 23% and 21% in systems with monitoring cluster sizes 5 and 13, respectively. Simulations were also executed in faulty networks where 10% of the links are set to unusable state (see Figure 6.11 and Figure 6.12). These links were randomly chosen and simulations were run with several different random fault patterns. When status data processing is used, the performance increases are 78% and 74% in networks with cluster sizes 5 and 13, respectively. Without status data processing, the corresponding values are 78% and 72%.

The analysis shows that DCM with small cluster size improves network performance significantly. An especially notable feature is its ability to maintain the network throughput in a faulty network. Without network monitoring, the throughput decreases 41% when 10% of the links become faulty. However, if the presented monitoring is used the decrement is only 11%. Furthermore, the throughput in a faulty network with monitor-

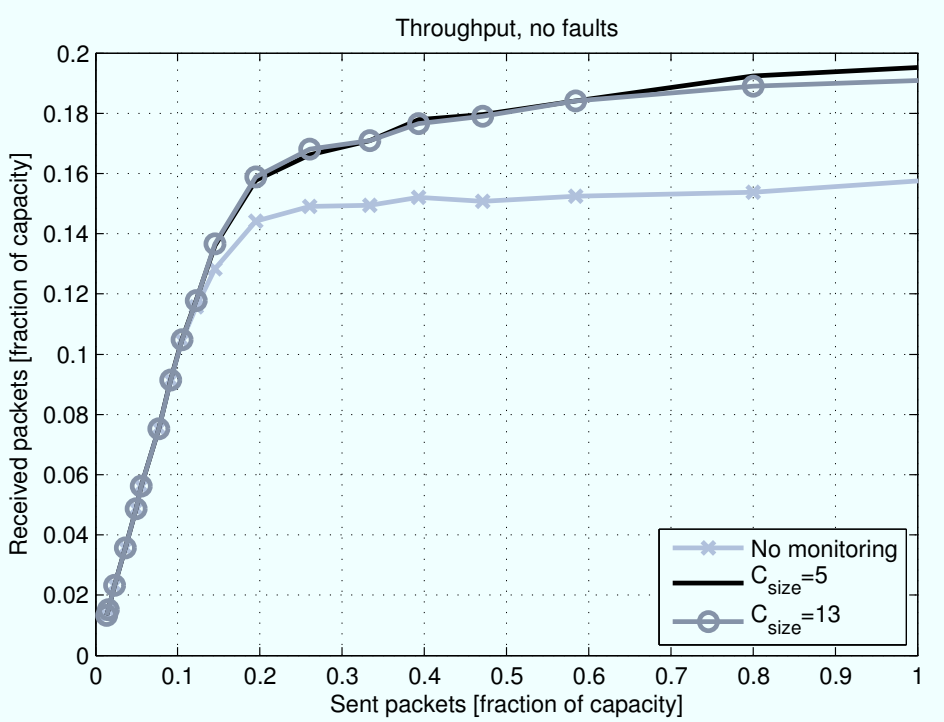


Figure 6.9: Throughput with different sized clusters (C_{size}) without network faults and with additional status data processing.

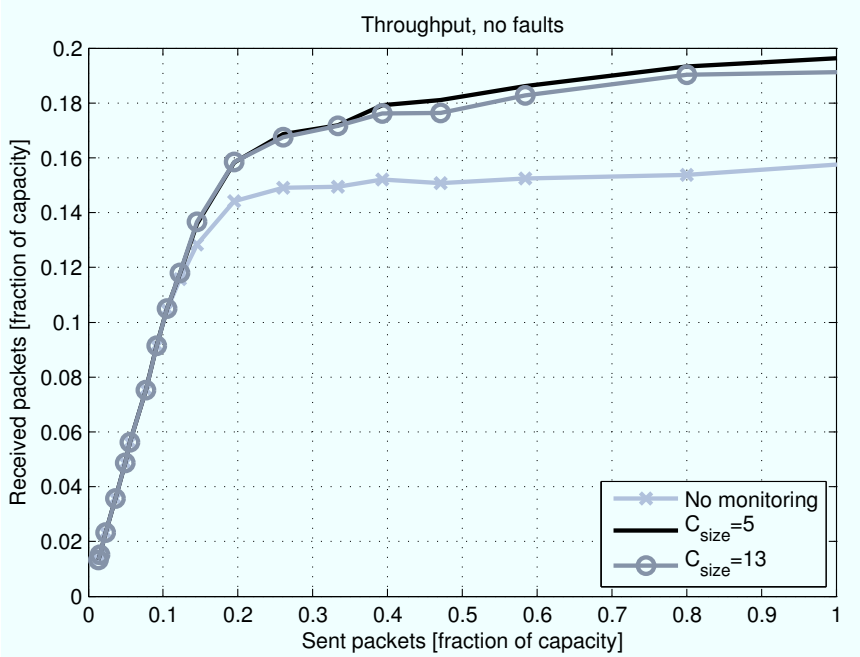


Figure 6.10: Throughput with different sized clusters (C_{size}) without network faults and without additional status data processing.

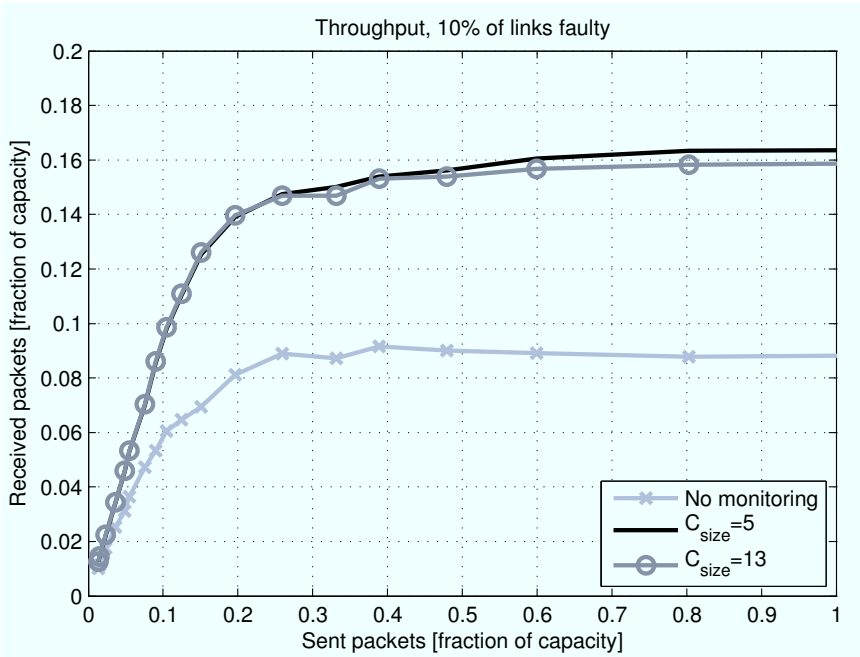


Figure 6.11: Throughput with different sized clusters (C_{size}). 10% of links are faulty and additional status data processing is enabled.

ing is 6% higher than it of a faultless network without monitoring.

A noteworthy observation is that a larger cluster size does not have positive impact on the performance but actually reduces it. This phenomenon can have multiple reasons. One reason for the inefficiency can be that too much data processing leads to inaccurate status data. The network status values are based on the status of the current router as well as the statuses of its neighbor routers. Too much of averaging between the status values equalizes the status values over the network and dissolves the differences between them. Another reason could be the latency in the status data propagation which makes it outdated before it is utilized. The monitored status data are transferred using the shared network resources in the actual data network. Even though the monitoring data are prioritized and passes all the other data in the router buffers, the transfer latency especially between the current router and the neighbors of the neighbors is present. When the traffic pattern in the network is rapidly changing the network with large monitoring cluster size possibly cannot react to the change quick enough. A large cluster could work in networks with more static traffic pattern. However, the need for routing reconfiguration and extensive network monitoring is lower in these kind of systems. Therefore, rapid reacting to change in the traffic pattern is necessary.

The influence of the additional status data processing is small or even non-existent.

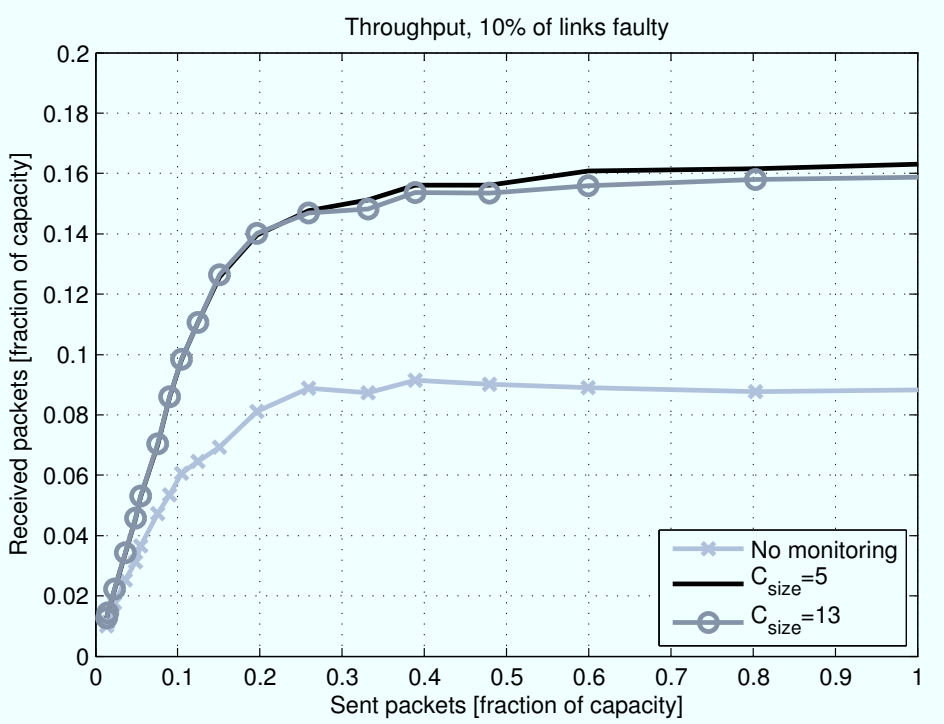


Figure 6.12: Throughput with different sized clusters (C_{size}). 10% of links are faulty and additional status data processing is disabled.

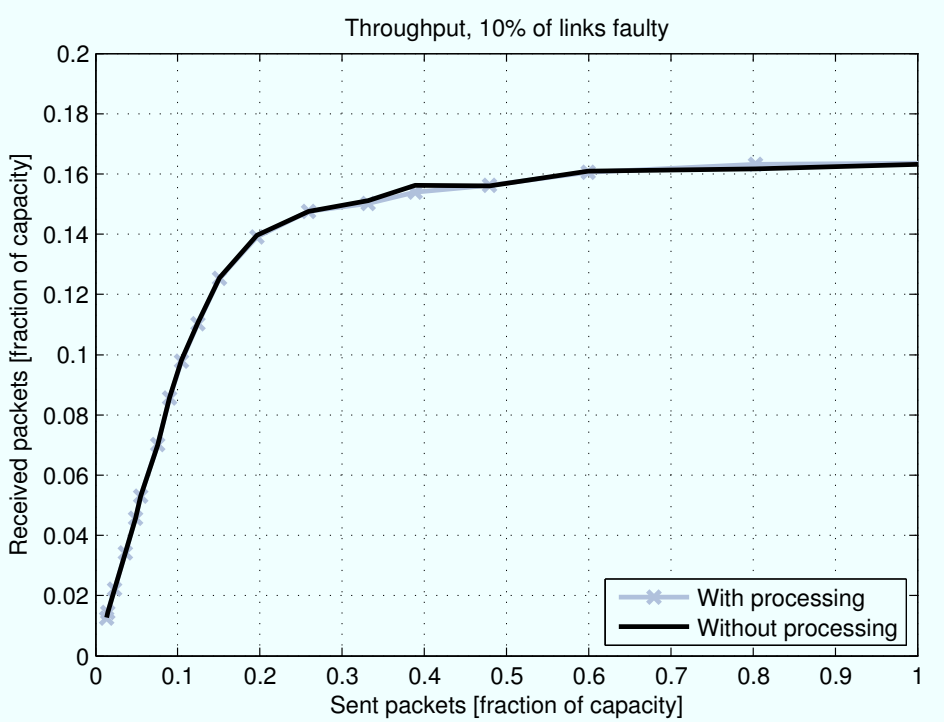


Figure 6.13: Throughput with and without additional status data processing. $C_{size} = 5$, 10% of links are faulty.

In a faulty network there is a very small increase in the throughput. However, in a faultless network the impact is even negative. For example, Figure 6.13 shows the difference in throughput in a faulty network with $C_{Size} = 5$. As can be seen in this specific comparison the difference is negligible.

The inefficiency of the status data processing may stem from the same factors as that of the large cluster size. The differences between status values dissolve and are not on display so that the routing algorithm could make right decisions.

6.3 Format of Network Status Data

The network status data are used to deliver the information of the state of the network and it can be used to different purposes. When the main purpose of use is traffic management the data typically include information concerning network load and faults. Faults are simply denoted using binary values which indicate if a component of the network is usable or faulty. In more complex systems multi-level fault indicators could be considered. The network load is denoted using a scale where different values represent different amounts of load on a network component. In our simulations, the network load

representation is linear. Different scales can be considered in some specific applications.

6.3.1 Granularity of the Router Status Values

The status data granularity defines the resolution of the status data values which is how many different values there are on the scale which is used to represent the load on a network component. The smallest used value indicates that the load of a network component is very low and the highest value represents a high load of the component. The rest of the values indicate component load linearly between the extreme values. An example of the status data granularity was given in Section 4.5. In physical implementations, the status data values are represented as binary numbers which means that finer granularity requires more bits and that way increases the size of monitoring packet payload. The status data granularity impacts on to the amount of the monitoring data to be transferred as well as to the required computational resources in the monitoring components. The granularity should be chosen so that the required data transfer and status data processing resources are adequate in the framework of the NoC implementation in question.

The 64-core NoC has been simulated with different granularity alternatives. 32 was defined to the maximum possible granularity because of the limited size of payload in the monitoring packets. A status value with 32-level granularity can be indicated with 5 bits. When monitoring cluster size is 13 a monitoring packet should include information on router's status and statuses of its four neighbors. With the granularity of 32, this takes 25 bits which can be considered a realistic amount of data in a monitoring packet. The same data granularity is also used between probes and monitors.

The throughput of a 64-core NoC with diverse status granularity is presented in Figure 6.14, Figure 6.15, Figure 6.16 and Figure 6.17. The simulations were carried out in a faultless network and in a network where 10% of the links were faulty. The results show that in a fully functional network the performance is only slightly improved when granularity is larger than eight. It can be also noted that in a faulty network the granularity should be at least 16. In both cases 4-level granularity leads to significantly lower performance which means 7% decrement in the faultless and 4% in the faulty network compared with the 16-level granularity. This supports the use of at least 16-level granularity. The performance of the system without monitoring is illustrated as a reference.

6.3.2 Combining Router and Link Statuses

Unification of traffic and fault information is a method to simplify monitoring status data. In the original status data format (see Figure 4.8) there is a binary number to represent the traffic load and a bit to indicate the resource faultiness. To decrease the monitoring complexity and the size of monitoring data payload we analyzed two approaches where the monitoring data are combined to hybrid forms. These two hybrid data formats are presented below.

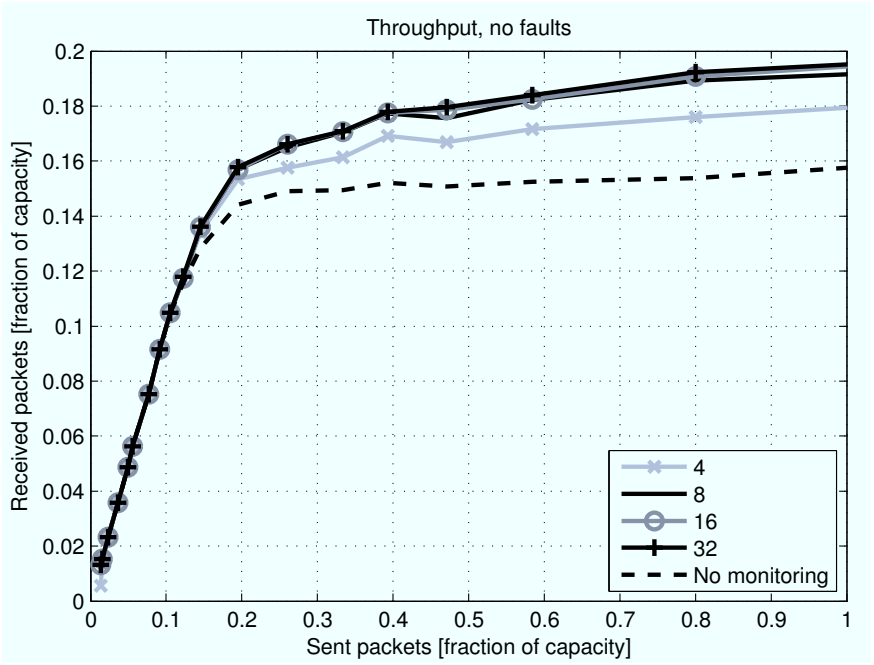


Figure 6.14: Throughput with different traffic status granularity alternatives. $C_{Size} = 5$ and data processing is enabled. No faults.

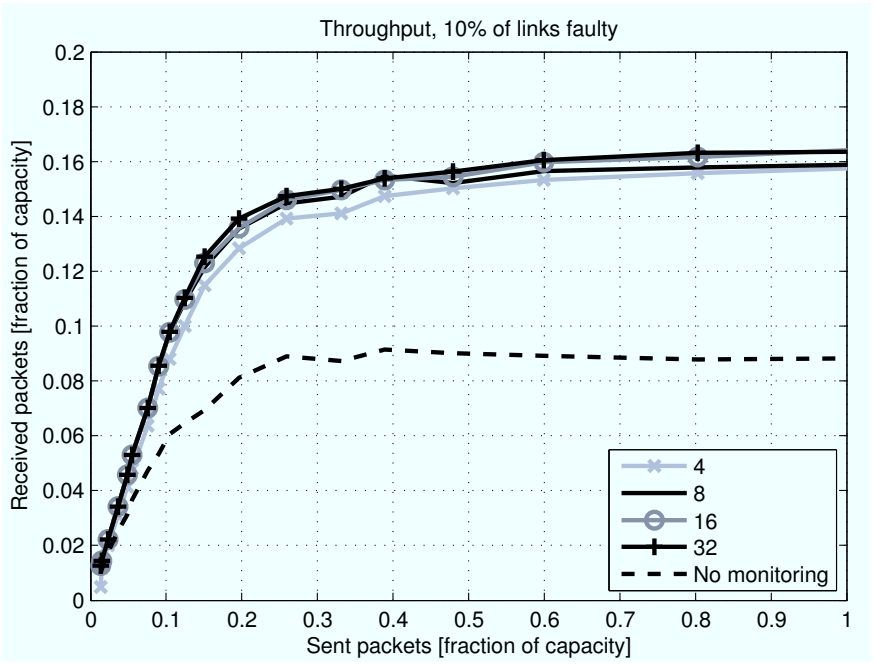


Figure 6.15: Throughput with different traffic status granularity alternatives. $C_{Size} = 5$ and data processing is enabled. 10% of links are faulty.

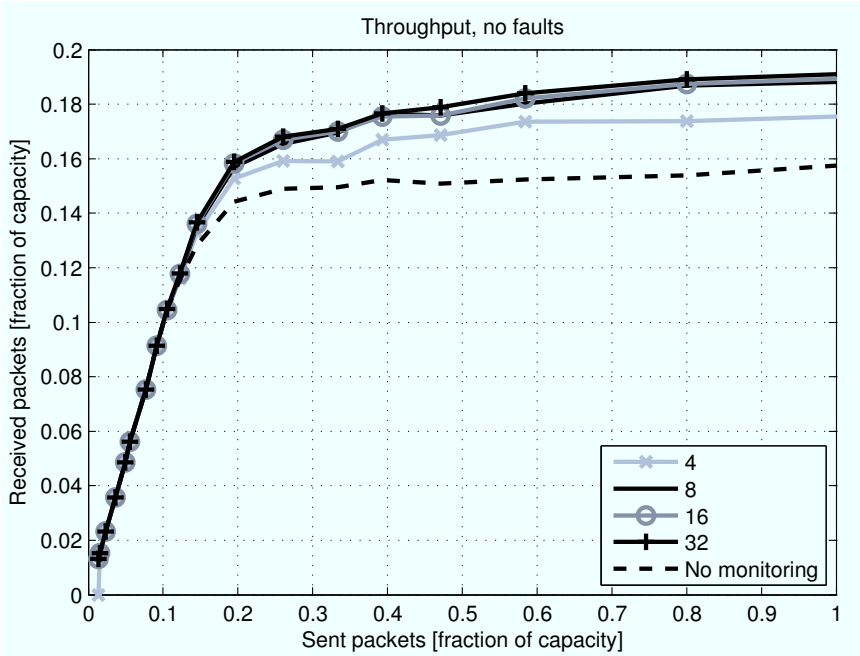


Figure 6.16: Throughput with different traffic status granularity alternatives. $C_{Size} = 13$ and data processing is enabled. No faults.

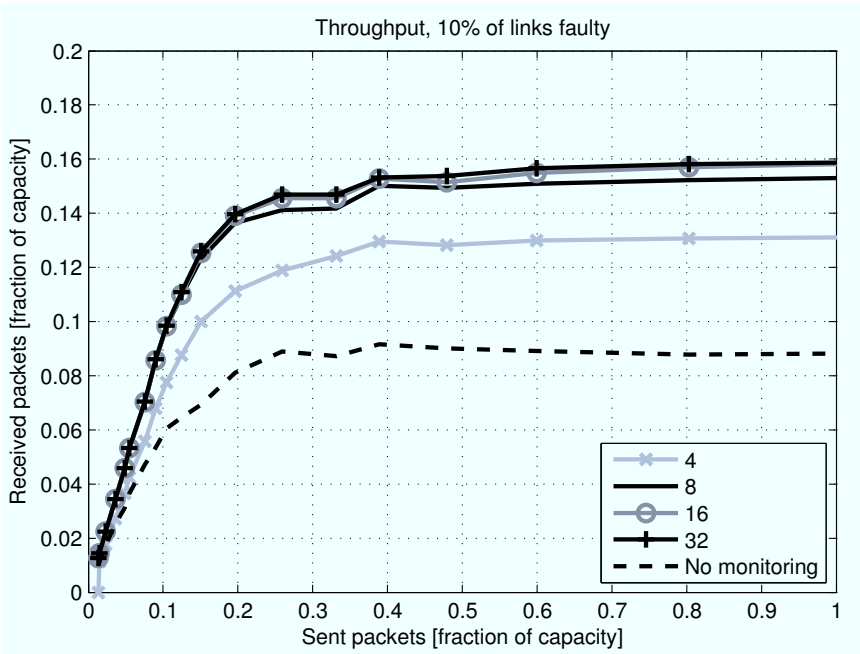


Figure 6.17: Throughput with different traffic status granularity alternatives. $C_{Size} = 13$ and data processing is enabled. 10% of links are faulty.

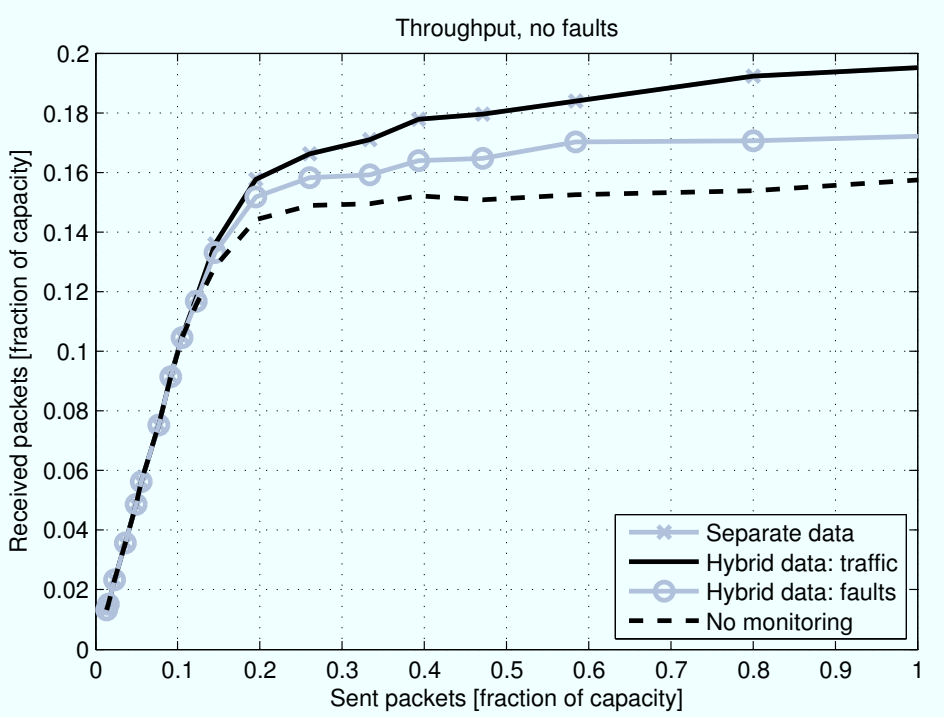


Figure 6.18: Throughput with separate traffic and fault data as well as with the hybrid formats. $C_{Size} = 5$ and data processing is enabled. No faults.

Hybrid Status Data Using Status Values

Traffic status values can be used to indicate faults by defining that the maximum status value does not only indicate high traffic load but also faulty resources. If there is a faulty component in some direction, the traffic status value of that direction is set to its maximum value. In this case the payload of a monitoring packet (see Figure 4.8) is reduced by 4 bits because the fault values are not included. When this format is utilized, the routing algorithm has to be configured to totally avoid the routing directions with maximum traffic values.

Hybrid Status Data Fault Indicator Values

The monitoring data are simplified even further when all the status data are combined to the boolean fault indicator values. In this approach, a routing direction is marked as faulty when there is high traffic load. The status can be restored when the traffic load decreases. This way packets are not routed in highly loaded directions. A drawback in this approach is the loss of knowledge about differences between routing directions with low and medium traffic load. Because the traffic status values are not used, the reduction of the monitoring packet payload is n bits if $C_{Size} = 5$ and $5n$ bits if $C_{Size} = 13$.

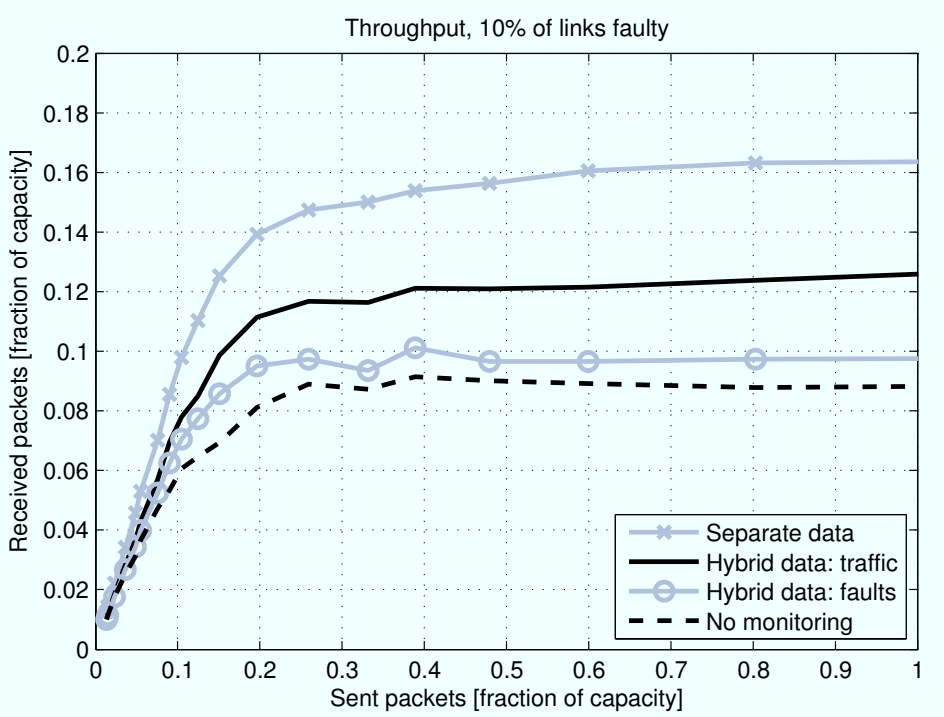


Figure 6.19: Throughput with separate traffic and fault data as well as with the hybrid formats. $C_{Size} = 5$ and data processing is enabled. 10% of links are faulty.

The monitoring data combination approaches were simulated with the NoC model and the results are presented in Figure 6.18 and Figure 6.19. Obviously, in a faultless network the traffic status based combination works similarly as separate data. When the traffic data have been integrated into the fault statuses, the decrease in throughput is 8%. However, the performance is still 12% better than without monitoring. In a faulty network separated status data are notably the best solution. The traffic data based hybrid format causes 24% performance loss which is even larger with the fault data based format, 40%. Nevertheless, these hybrid formats increase the performance by 36% and 8%, correspondingly, compared with the system without traffic monitoring.

The presented analysis leads to a resolution that both monitoring data classes are necessary in a system where faults are a realistic threat. In less vital applications, the hybrid formats could be a good compromise.

6.4 Serial Monitor Communication

In the DCM structure the monitoring data are transferred in the same network which is used by the original data packets. It is a straightforward solution which minimizes the requirement of additional resources. However, a shared-resource structure is always at

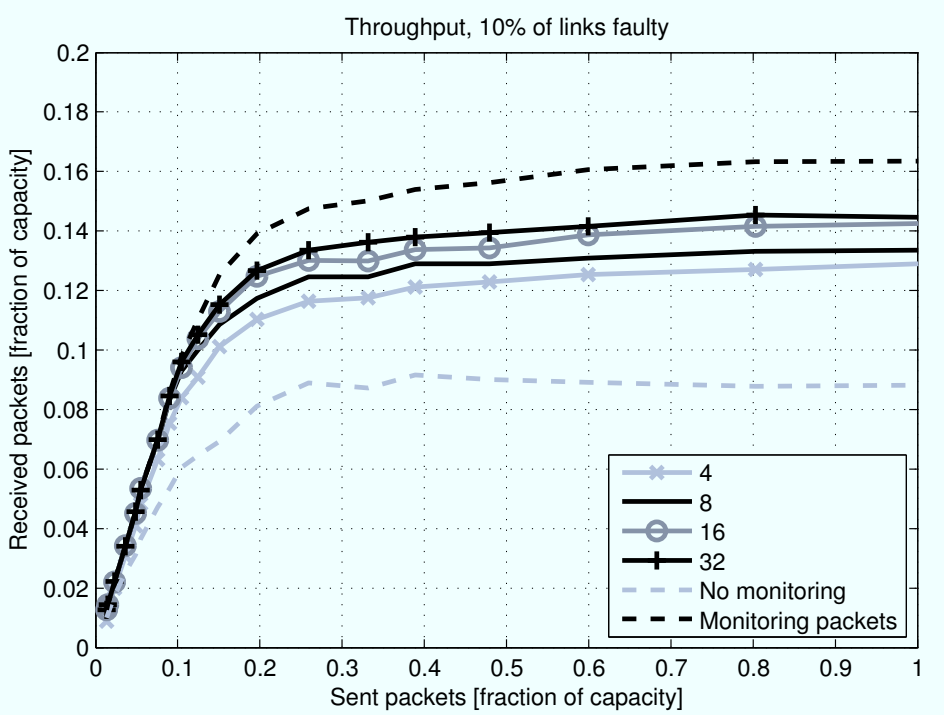


Figure 6.20: Throughput with different granularity alternatives using serial communication. 10% of links are faulty and data processing is enabled. $C_{Size} = 5$.

least somewhat intrusive and it consumes the network resources which otherwise could be used by the actual data packets.

An alternative solution to the inter-monitor communication is serial communication which is implemented with dedicated channels. It can be realized with relatively small amount of additional resources. A drawback in the serial communication is the increased transfer delay. However, because the serial communication resources are dedicated to the monitoring communication there can be a non-stop status update without paying attention to update intervals. [56]

Serial monitor communication was simulated with the SystemC based NoC simulation model. Throughput with different status data granularities and serial communication is presented in Figure 6.20 and Figure 6.21. The serial transmitter operates at the same clock frequency as the maximum frequency of the monitoring packet transmitter. However, the latter rarely works on its maximum frequency because of the status update interval conditions.

Essentially serial communication is slower than the earlier discussed parallel, packet based communication and the theoretical delays of the serial communication are even more increased when there is large amount of data to be transferred for example in systems with relatively large monitoring clusters. However, in contrast the serial com-

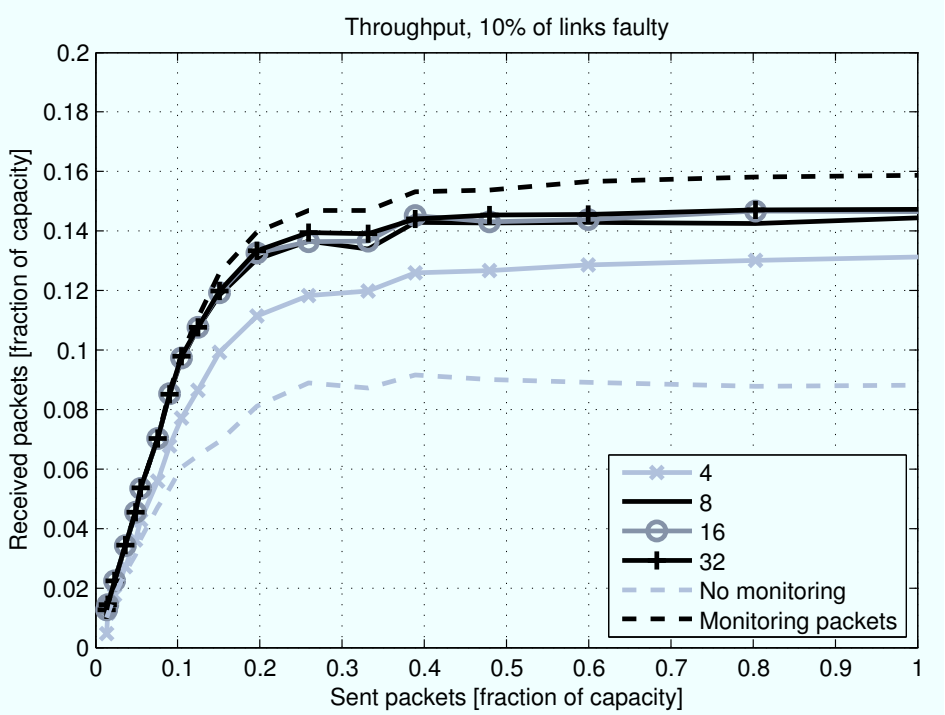


Figure 6.21: Throughput with different granularity alternatives using serial communication. 10% of links are faulty and data processing is enabled. $C_{Size} = 13$.

munication is operating in dedicated communication resources which can be used only to this purpose all the time. This way the status values can be updated actually more often than when the monitoring packets are transferred in the shared resources. Somewhat surprisingly the system with $C_{Size} = 13$ works well also for coarser status granularities when serial communication is utilized. This is a result of a shorter traffic status update interval even though in this case the amount of serially transferred data is quite large. In this case the granularity of 4 clearly stands out. Possibly the granularity of 4 is simply too rough to be used with the data amount of a system with large monitoring clusters. Figure 6.20 shows that when serial communication and small cluster size are used the performance differences are more notable also between granularities 8, 16 and 32. When serial communication is used, in a system with $C_{Size} = 5$ the performance with the granularity of 32 is 11% less than with a corresponding system using monitoring packets. Respectively, the performance of a system with serial communication and the granularity of 4 is 39% better than the performance of a system without monitoring. The corresponding percentages for system with $C_{Size} = 13$ are 6% and 42%.

The serial communication could be a useful option when a designer wants to keep the communication resources of the actual data and the control system separately. The serial approach guarantees that the monitoring communication does not disturb the

actual data which are transferred in the network. It may be possible to increase the clock frequency of the serial transmitter from what was used in the presented analysis. In this case, the performance differences should shrink.

6.5 Reduction of Monitors

The DCM system is based on a structure where there is an identical monitor attached to each router. These monitors include both monitoring and probing components. One potential way to reduce monitoring structure complexity is to decrease the number of monitors systematically by removing every nth monitor. In this approach there is a probe attached to every router but a monitor is attached only to a limited number of routers. This means that there is still complete knowledge of the network state in the monitors because there are probes attached to every router. Two monitor removal patterns are illustrated in Figure 6.22. The monitors receive probed data, process it and deliver it to the local router. The routers, that do not have own monitor, should utilize a deterministic routing algorithm because they do not have access to the probed status data from the neighboring routers. An adaptive algorithm is utilized in the other routers [14]. The simplified deterministic routers forward packets based on a deterministic routing algorithm. In problematic traffic or fault cases a simplified router could route packets randomly just directing them to some of its neighbors. In any case the neighbors of a deterministic router are adaptive routers which can route the packet forward adaptively.

In addition to performance, the reduction of monitors affects the complexity of the NoC implementation. Routers which do not have own monitoring component could have less complex routing logic which decreases the router area. This is due to the deterministic routing algorithm which substitutes the adaptive routing algorithm in the

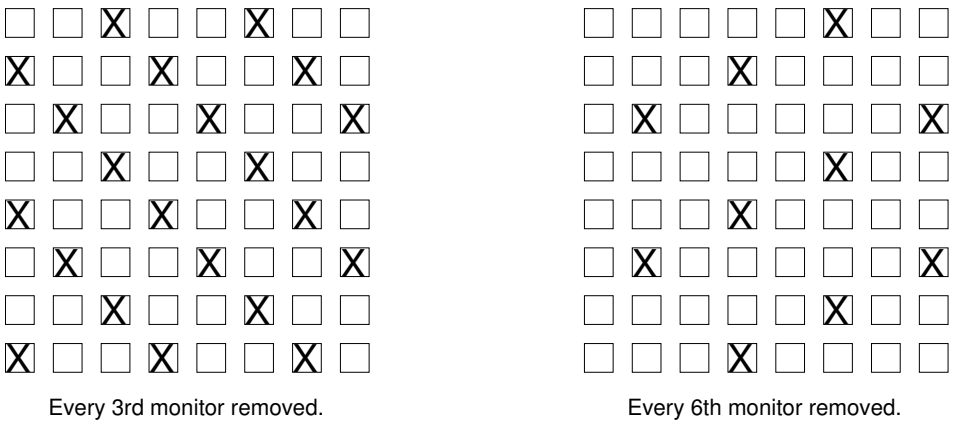


Figure 6.22: Two patterns of removed monitors. Removed monitors are marked with X.

routers which do not have a monitoring component. However, the probing components cannot be simplified because they have still have to offer status data to other monitors.

This approach was analyzed using our SystemC based NoC simulation model and the results are presented in Figure 6.23 and Figure 6.24. The simulation cases were chosen so that the un-monitored routers are placed as evenly as possible in the network. This way we defined four simulation cases where every second, every third, every sixth and every twelfth monitor was removed from the network.

The figure shows that the removal of a monitor, even if it is just every 12th, has notable influence on network throughput and the influence is even more remarkable when there are faults in the network. Removal of every second monitor causes 18% performance decrement in faultless network and 43% in the network with 10% of faulty links. In a faultless network the performance is equal with the performance of a system without traffic monitoring. In a faulty network there is 2% performance increase compared with the unmonitored system. If just every 12th monitor is removed, the performance decreases by 3% and 10%, respectively.

The removal of monitors has positive impact on area and traffic overheads caused by the monitoring system. However, the total area of the monitoring system is almost negligible in comparison with the area of a 64-core NoC. This way the removal of monitors

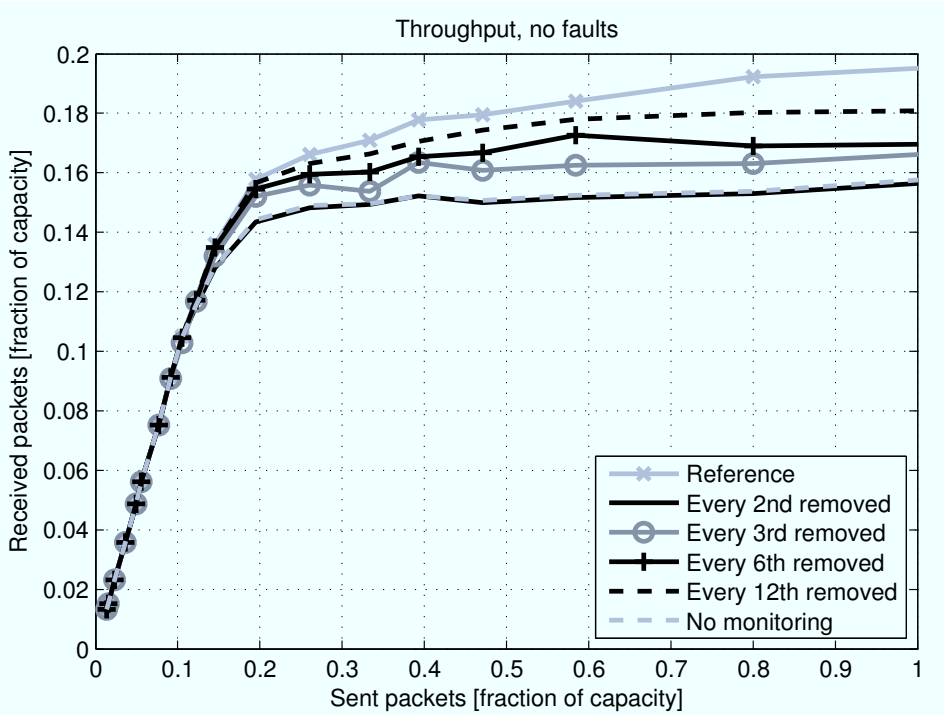


Figure 6.23: Throughput with fewer monitors. $C_{size} = 5$ and data processing is enabled. No faults.

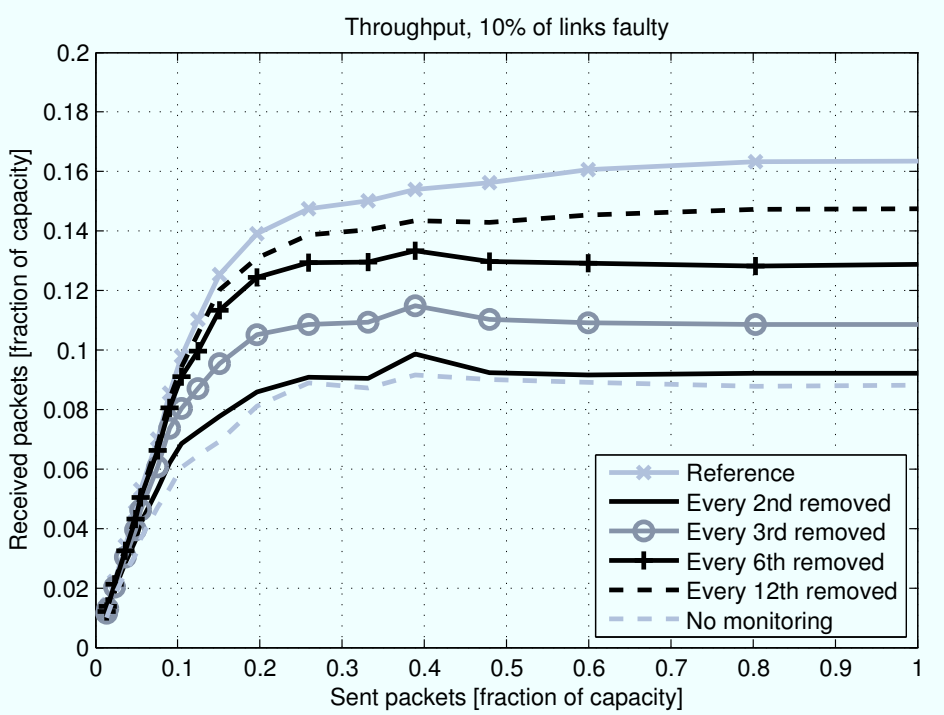


Figure 6.24: Throughput with fewer monitors. $C_{size} = 5$ and data processing is enabled. 10% of links are faulty.

cannot be justified with the reduced complexity when the performance decrement is as large as presented here. In application specific NoCs it could be reasonable to remove monitors from areas where traffic is predictable so that the resources can be sized properly during the design phase and adaptivity is not necessary. However, in our work the focus is on homogeneous general purpose NoCs so the monitors are placed evenly over the network.

6.6 Summary

This chapter discussed and analyzed traffic management related issues in the dynamically clustered distributed NoC monitoring structure. At first, different status data update interval approaches were presented and their impact to overall performance and cost were analyzed. Then, the diffusion of status data was studied and the effects of additional status data processing were analyzed. Format of status data was studied and usage of serial communication between monitors was analyzed. Finally, the effects of reduction of the number of monitors was analyzed.

Chapter 7

Lightweight Task Mapping

Task mapping is an essential part of management of a multiprocessor Network-on-Chip. In conventional processor systems tasks can be assigned statically to processor cores during the design phase of the system. However, when there are tens or hundreds of cores static mapping cannot exploit the full potential of the NoC in terms of flexibility, scalability and performance. A large NoC is able to accommodate a lot of different kind of tasks in various sizes. This capability is strongly limited if mapping has to be carried out once and it cannot be reconfigured run time. An obvious solution to this is dynamic task mapping which customizes task mapping during operation of the system. The goal of task mapping is to find the trade-off between maximal processor utilization and minimum communication resource utilization [74].

The task mapping methods can be roughly categorized in terms of three factors. The first factor is the used system metrics that affect the mapping decisions. These metrics are basically computational load of cores and traffic load of the network. The load metrics show how much there are free computational and communication resources that can be assigned for new tasks. These metrics can be used individually or in conjunction. The second factor is the type of mapping implementation which can be centralized or distributed. In the centralized approach there is a mapping processor which has complete overall knowledge of the status of the system and makes all the mapping decisions. In a distributed approach the mapping process is distributed in several units around the system and it is carried out without centralized control. Finally, the third categorization factor is the type of the mapping itself. The mapping can be static or dynamic. Static mapping is done once and it remains unchanged during the operation of the system. Dynamic mapping has a starting point when the tasks are mapped during the system initialization, but the mapping can change and new tasks may be mapped during run time.

Centralized mapping shares the same strengths and weaknesses as any centralized system. They are relatively easy to implement and control because all the information is collected into one place in the system. It is also good in terms of area overhead because only one mapping processor is required. On the other hand, a centralized mapping is

fragile for faults and causes high monitoring and mapping traffic load especially around the central controller.

Some characteristics, used in distributed run-time task mapping, are previously presented in different research articles. A run-time agent-based distributed application mapping for on-chip communication has been presented in [4]. It is designed for heterogeneous multiprocessor systems. The approach presented in [4] uses fixed virtual clusters and cluster agents which take care of task mapping in a cluster. There are also global agents which map applications to different clusters.

Communication-aware heuristics for run-time task mapping on NoC-based MPSoC (multiprocessor System-on-Chip) has been presented in [61]. Four run-time mapping heuristics have been presented including mapping strategies which map a new task for example to their nearest neighbor systematically or by choosing the best neighbor available. The mapping can be also communication-aware in a way that the distance between two communicating tasks is tried to be minimized.

A decentralized task mapping approach for homogeneous multiprocessor NoCs has been presented in [74]. The mapping is based on local information on processor workload, task size, communication requirements and link contention. This information is collected only from a close vicinity of a core. The presented task mapping algorithm tries to find better mapping for current task scheme in the system.

Our goal is to extend the previously presented dynamically clustered network monitoring structure to be used as a part of a lightweight task mapping mechanism. The cores of the system should be able to independently assign tasks to other cores. The proposed task mapping mechanism is based on the idea of mapping tasks to the best available neighbor core. The idea behind our solution is that a core initiates task mapping by sending a mapping packet to another core which seems to be the best from its own point of view. Mapping packets are discussed in Section 7.2. Finally, the task ends up mapped to a core. The task can be mapped to a core where the initiator core sent the mapping packet at the beginning or the task can be remapped further to a less loaded core. The initiator core is only interested about the fact that the task will be executed properly with a reasonable performance, without regarding which core does it.

7.1 Lightweight Distributed Task Mapping

Our lightweight distributed task mapping is based on the dynamically clustered monitoring (DCM) structure (see Chapter 4). The DCM structure [55, 58] is originally designed for traffic management purposes to collect traffic status and fault data from a network. In this chapter the monitoring structure is enhanced and applied for dynamic run-time task mapping purposes.

The task mapping mechanism is designed to be as lightweight as possible so that acceptable performance can be reached with minimal area overhead. Intrusiveness of run-time task mapping is also taken into account because any run-time management operation should not interfere with the main operations of the system. The lightweight

task mapping is distributed to the monitors all around the network, which makes it flexible and highly scalable. As the system is distributed, the added energy consumption and heat generation are also distributed over the whole system. The communication overhead is also moderate due to the lack of centralized control and long-distance control related communication. Task mapping functionality is allocated into monitors and cores so that the monitors take care of the neighbor core evaluation and the cores make the mapping decisions. The system does not try to pursue optimal task mapping schemes but good mapping schemes which lead to acceptable overall performance while maintaining the other goals including flexibility, scalability and low intrusiveness.

Monitoring and mapping processes are distributed to cores and to separate monitoring components which are directly connected to the cores. The monitors collect network and computational load information from their neighborhood and offer this data, combined with their own statuses, to their own neighbors. This way the cores do not only have knowledge of their neighbors' statuses but also wider overall knowledge of network and computational load in different directions in the network. The neighbor data collection and the monitoring principles were presented in Chapter 3.

Typically, tasks are mapped to the processors in the initialization phase of the system and the execution of these initial tasks begins. This initial mapping can be done statically during the design phase of the system or it can be left to the responsibility of an operating system. The idea of the dynamic run-time task mapping is that while the initial task mapping has been carried out at the initialization phase, the computational cores are able to map new tasks at run-time independently without centralized control. This way the operation of the system is flexible and the run-time reconfiguration does not cause significant load to the system.

The status data in the distributed monitoring are collected from the neighbors and the received data can also include data from the neighbors' neighbors. Therefore, a monitor does not have complete knowledge of the network but only the status of the cores and routers in its own dynamic cluster (see Chapter 4). Complete knowledge is not essential because one of the main principles of the lightweight task mapping is to minimize the distance between tasks which are communicating with each other. Lower computing performance could be beneficial if higher computing performance causes a lot of long-distance communication, because overall performance is the joint effect of computational and communication performance. Mapping to a remote core on the other side of the network is possible if necessary. This has been made possible by a task remapping mechanism which can forward a mapping packet until a suitable core has been found. The remapping is discussed later in this chapter.

One essential principle in the lightweight task mapping mechanism is that the initial mapper does not know in the beginning where its task will finally be mapped. This knowledge is irrelevant. All that a core has to know is that the task will end up to a core where it can be executed with acceptable performance. When the task has been mapped, the original mapper is informed where the task resides. This information is not needed prior to completion of the mapping.

7.1.1 Core Load Monitoring and Reporting

The lightweight task mapping mechanism is designed as a part of the dynamically clustered monitoring structure where the cluster size of 13 is utilized. In a dynamically clustered structure each core has knowledge of the cores in its own dynamic cluster which comprises the 12 nearest neighbors around the core. The data collection is carried out similarly as in the network status data collection (see Chapters 4 and 6). Each monitor has a table where it stores values which illustrate free computational capacity in the neighboring cores. The table is kept up to date as new monitoring packets arrive. The monitoring packet arrival frequency is determined by the used update interval, which has been discussed in Section 6.1.

The monitor reports the index of the currently best neighbor core to the local core. The selection of the best core is discussed in Section 7.2.1. The absolute value of the free capacity in the best core can be reported with the index of the core if it is required by the used mapping decision strategy. Mapping decision strategies are discussed in Section 7.2.2.

7.2 Mapping Decision Process

The mapping decision process is implemented in cores and monitors and the process is divided into four steps which are illustrated in Figure 7.1. The process starts when a master core decides to map a task. It generates a *task mapping packet* which is notably similar to a monitoring packet (see Chapter 3) but the packet type indicator indicates that it is a task mapping related control packet. A mapping packet includes the definition of the task, the size of the task and the address of the original mapper. The monitor defines which of the cores in the know neighborhood is currently the best candidate and the core sends the mapping packet to this core. The phases, described above, are executed in the *initialization* phase. When the chosen destination core receives a mapping packet, it has two options how to deal with the packet. The core can decide to execute the task by itself or it can remap the task to some other core. When the core decides to remap, it can choose the best neighbor from its neighborhood or return the task to the original mapper.

7.2.1 Neighbor Evaluation

The monitor decides which core in its known neighborhood is currently the best one to receive new tasks to be executed. In our task mapping method, the evaluation is based on computational load (core load), communication resource load or a combination of these two load metrics. Maximum core load is the core capacity which in here is an abstractly defined concept. It represents a size of a typical task which can be assigned at a time to a core. The core capacity can be overloaded which basically means that the overloading task waits for the previous task to be completed. The communication

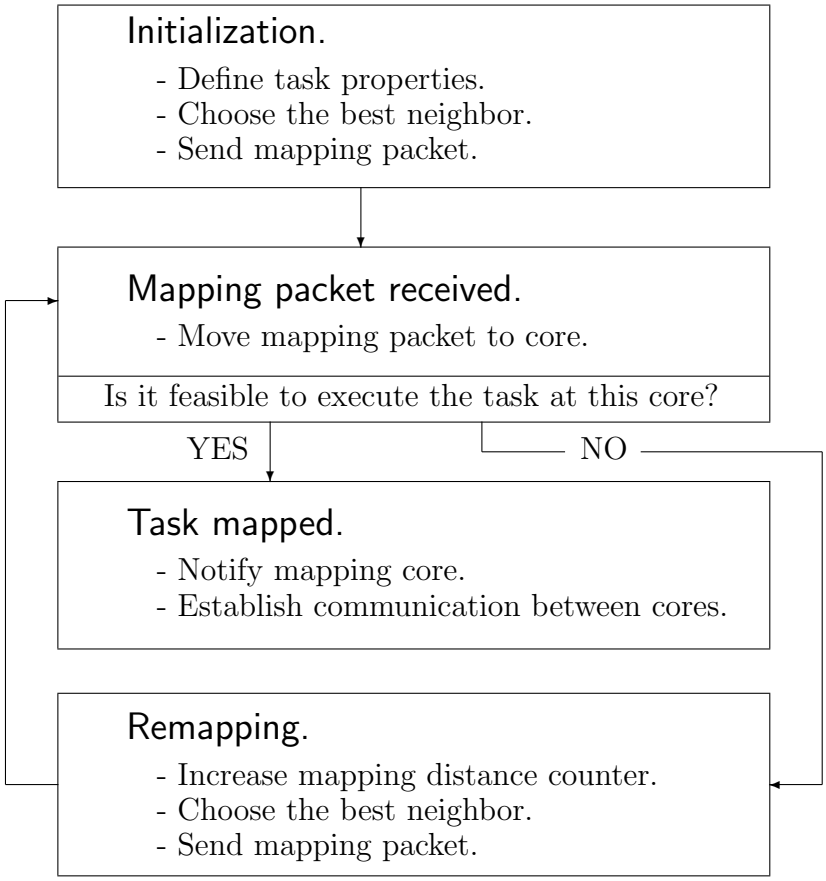


Figure 7.1: Mapping decision process.

resource load is the same metric which is used in traffic management to reconfigure adaptive routing at run-time (see Chapter 6).

Core-Load-Aware Evaluation

Core-load-aware neighbor evaluation is the simplest evaluation option. The monitor compares the core load values of its neighbors and chooses the one with the highest free capacity as the currently best neighbor core. The core load values are defined so that a core measures its current load (C_{used}) and subtracts it from its maximum computational capacity (C_{max}). This result is the current free capacity (C_{free}) of the core and it is used as a core status value in the core-load-aware evaluation.

$$C_{free} = C_{max} - C_{used} \tag{7.1}$$

Communication-Aware Evaluation

Communication in the network is mainly caused by the tasks which communicate with each other. If we define that memory accesses and I/O communication are also essentially tasks, all communication takes place between tasks. Therefore, there should be a relation between the computational core load and communication resource load. A simple communication-aware neighbor evaluation uses only the traffic monitoring data and chooses the best neighbor core based on this traffic information. The traffic status ($S_{traffic}$) is calculated using Equation 6.3 (see Section 6.2.1). This is a very lightweight mapping solution because there is no need for the core load monitoring at all. However, one can assume that the accuracy suffers when the decisions are based on secondary information, the communication load information in this case.

Core-Load- and Communication-Aware Evaluation

The third neighbor evaluation approach is a combination of the above evaluation approaches. It is both core-load- and communication-aware. The core load and communication statuses are combined to a single status value which represents both the free computational capacity and the status of the communication resources. The core load status values show the amount of free capacity so that a larger value represents smaller core load and that way a larger amount of free capacity. The traffic status values, which are used to measure the load of communication resources, represent the traffic status so that a larger value corresponds to a larger load in the network. A combined core load and communication status value ($C_{combined}$) is calculated using equation

$$C_{combined} = C_{free} - S_{traffic}, \tag{7.2}$$

where C_{free} is calculated using Equation 7.1 and $S_{traffic}$ is the traffic status value in the direction of the core. Finally, the selection of the best neighbor core is based on the ranking of these combined core load and communication values. An additional weight

factor can be utilized to adjust the significance of the traffic status in Equation 7.2. In the presented communication-aware evaluation approaches the available fault information is utilized so that tasks are not mapped in directions where a fault has been observed.

7.2.2 Mapping Decision Strategies

The task mapping decision strategy defines what a core does when it receives a task mapping packet. There are three procedures to select from: the core can take over the task and the mapping is completed, the core can do a remapping and map the task forward to a core which seems the best from its point of view, or the core can return the mapping packet to the original mapper. This study includes four different task mapping decision strategies which execute these three procedures based on the defined strategy. These strategies are presented below.

1. Remapping forward if there is not enough free capacity for the task in the current core. The task is remapped as long as it ends up to a suitable core. The disadvantage of this decision strategy is that the mapping distance is unbounded. The tasks can be mapped far away from the mapper and if there is no suitable core in the network, the mapping packet is finally dropped similarly as un-routable data packets. If the mapping packet is dropped the task will not be completed and the orginator of the task has to initiate another instance of the same task mapping.

2. Core capacity overloading if mapping distance exceeds a statically defined distance threshold even if there is not enough free capacity in the current core. There is a counter in a mapping packet which shows how many times the mapping has been tried. Naturally this has a negative impact on the system performance. This decision strategy eliminates the disadvantage of the mapping strategy 1. However, the core overloading has a negative influence to the system performance. The mapping distance is bounded with the distance threshold and thus it can be kept moderate.

3. Capacity overloading if there is not enough free capacity in the known neighborhood. This strategy requires knowledge of the amount of free capacity in the neighboring cores so that the monitor has to report not only the ID of the best neighbor but also the amount of its free capacity. Performance-wise this is a good strategy because when a core is overloaded the distance between the mapper core and the mapped tasks remains moderate. The mapping distances can be kept even shorter than in decision strategy 2. However, this strategy requires higher monitoring complexity due to detailed capacity reporting.

4. Remapping backwards if not enough free capacity in the known neighborhood or in the current core. This strategy is similar to the previous (3) strategy. The difference is that when enough free capacity cannot be found, the current core is not overloaded but the mapping packet is returned to the mapper. In this

case the original mapper core recognizes that the packet is returned and overloads itself. In terms of performance there is a compromise that when core overloading is required the communication between cores is eliminated. The elimination of inter-core communication increases performance by lowering the communication resource load. However, packet returning and identification of returned mapping packets requires even more complex logic than the decision strategy 3.

7.3 Enhanced Monitoring System

Several modifications to the monitoring system are made to enable the task mapping in the dynamically clustered Network-on-Chip. The main improvements concern core and monitor modules while there are also smaller modifications in the router attributes and packet structure.

Each core has two new parameters, *capacity* and *load*. Capacity represents the computational capability of a core while load indicates how many units of the capacity is currently used. Capacity in this framework is an abstract concept (see Section 7.2.1). It can be seen as a typical maximum size of the task queue in a core. However, there is a possibility to overload a core by extending the size of the task queue. The core also has mechanisms to initiate task mapping, to assign a task to itself and to remap a task further.

Monitor components have been developed further to be able to collect also the core load data in addition to the network status information. An enhanced monitor also includes neighbor evaluation mechanisms which rank the neighbors and chooses the currently best neighbor core based on the used evaluation approach.

The routers are modified so that the task mapping packets have the second highest priority after monitoring packets. The prioritization is required to prevent mapping packets to get stuck in the network. However, the priority is not seen as important as the priority of the monitoring packets because in a highly loaded network the mapping of new tasks is arguable. The implementation of task mapping features in the simulation environment has been discussed in Section 5.4.5.

7.4 Case Study: Mapping Tasks

The functionality of the lightweight distributed task mapping algorithm was demonstrated as a case study. The demonstration was carried out using the presented NoC simulation environment (see Chapter 5). A mesh-shaped NoC containing 64 cores and routers (8 x 8) was modeled. The details to be examined concern the number of mapping decision attempts, distances between the mappers and the mapped tasks as well as the amount of failed mappings. The distribution of the computational core load was also examined. The used neighbor evaluation approaches are denoted as in Table 7.1. The notation of mapping decision strategies was presented in Section 7.2.

Table 7.1: Neighbor evaluation approaches.

	Description
a	**Core load** based evaluation.
b	**Communication** based evaluation.
c	**Combined** core load and communication based evaluation.

The purpose of this demonstration is not to find an absolute answer to which of these presented mapping strategies and evaluation approaches are the best in any given application. The case study is based on comparison of the mapping approaches and it is realized using examples showing how their functionality differs in a specific situation. The demonstration scheme has been chosen so that it would show the essential properties of different mapping approaches when the system is loaded with new tasks. The mapper cores are located in a large area in the middle of the network so that balanced task mapping is possible. To imitate mapping schemes of real systems the mapper area is not symmetric.

The starting point for the simulations is a 64-core NoC without any traffic or computational load on any core. During the simulation 14 cores initiate a task mapping one after another. These 14 cores are shown in Figure 7.2. This is repeated 4 times so that each of these cores initiates and sends 4 mapping packets totalling 56 tasks to be mapped. The example system consists of identical multipurposes cores so that each task

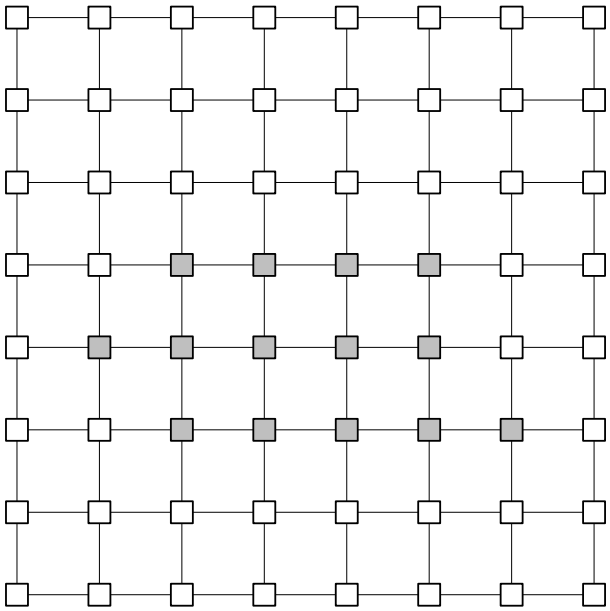


Figure 7.2: Task mapping initiator cores marked with gray.

Table 7.2: Task mapping results.

MDS	NEA	Task distance	Mapping decisions	Drop-%
1	a	1.74	1.23	16.07
1	b	2.00	2.10	64.29
1	c	1.87	1,38	16.07
2	a	1.91	1.42	0.00
2	b	1.91	2.11	0.00
2	c	1.98	1.43	0.00
3	a	1.71	1.13	0.00
3	b	1.09	1.07	0.00
3	c	1.79	1.18	0.00
4	a	1.48	1.23	0.00
4	b	0.32	1.75	0.00
4	c	1.34	1.25	0.00

MDS: Mapping decision strategy.
NEA: Neighbor evaluation approach.
Task distances and mapping decisions are average values.

can be executed in any of the cores. Communication in the network is modelled so that there is continuous communication between a mapper and the related core in which the task is mapped.

The results of the case study demonstration are collected into Table 7.2. Notice, that when using the decision strategy 1 the mapping packet drop percentage is very large especially when the communication-aware neighbor evaluation is used (64,29 %). High drop rate indicates that simple communication-aware neighbor evaluation does not reflect the computational status well enough. Because of the core overload mechanism of the other decision strategies, the mapping packet drop does not exist when those strategies are used.

The differences in the task distances and mapping decision attempt counts are not that prominent. The minimum of the task distances is reached using the decision strategy 4. The short task distances are explained by the mapping returning mechanism which eliminates the communication completely in some of the cases.

The computational core load at the end of each simulation is illustrated in Figure 7.3, Figure 7.4, Figure 7.5 and Figure 7.6. The color of the core shows the amount of the load. White cores are not loaded at all while black cores are highly overloaded. As can be noticed the purely communication-aware evaluation approach causes significantly uneven core load distribution. As was mentioned previously, there is a correlation between core load and communication resource load even though the relation is not perfect. An obvious uncertainty is caused by the fact that communication between two cores distributes the communication load evenly on the network path between these cores.

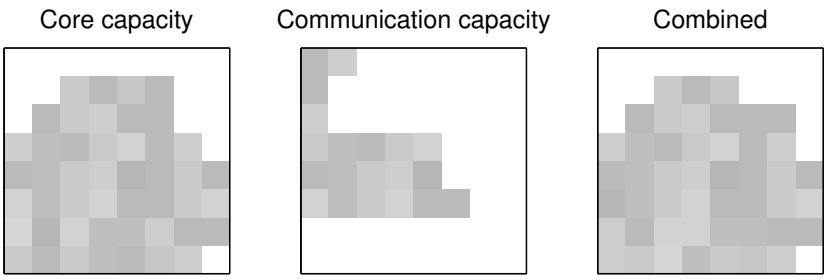


Figure 7.3: Core load with mapping decision strategy 1.

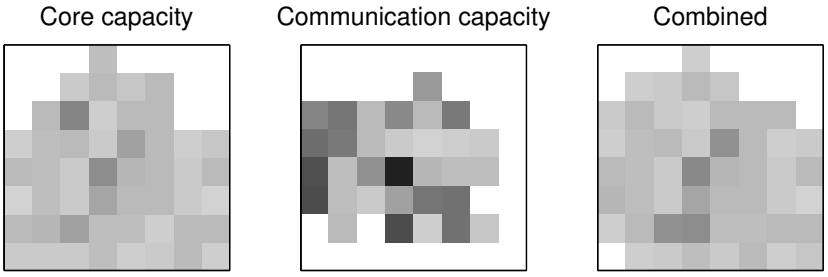


Figure 7.4: Core load with mapping decision strategy 2.

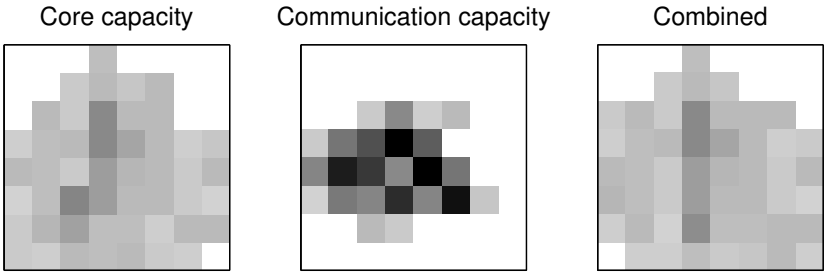


Figure 7.5: Core load with mapping decision strategy 3.

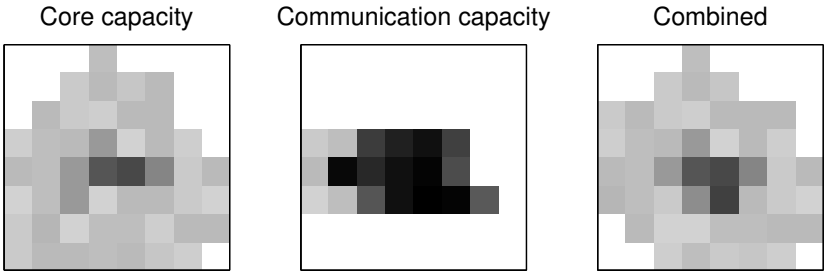


Figure 7.6: Core load with mapping decision strategy 4.

External observer cannot separate routers which are connected to a highly loaded core from routers which are just situated along a highly loaded network path. Network status data are also partly inaccurate because it shows the load of a router and not the exact load of a link. This inaccuracy has been discussed in Chapter 3.

Other evaluation approaches lead to moderately even load distribution. One can note that when decision strategy 4 is applied the number of overloaded cores is somewhat higher than with the other strategies. These overloaded cores are mostly situated in the middle of the network and middle of the area of mapper cores (see Figure 7.2). The tasks which are initialized from the middle of the loaded area end up more probably in situation where the task cannot be mapped. In this kind of situation the strategy 4 returns the mapping packet to the mapper core. This causes concentrated overloading in the middle of the highly loaded area.

While the strategy 1 causes unacceptable mapping packet dropping it seems that the decision strategies 2 and 3 with core-load-aware or combined neighbor evaluation approach (a or c) are the considerable alternatives. In this case study, the usage of communication status information in the combined neighbor evaluation approach does not cause notable load distribution differences in one direction or another. However, the overhead of combined evaluation related to the core-load-aware evaluation is minimal because the communication status information is anyway available in monitors.

7.5 Summary and Future Work

A task mapping approach applying the previously presented DCM structure was presented and its functionality demonstrated in this chapter. The presented demonstration shows that an enhanced DCM structure is suitable for lightweight task mapping purposes. Lightweight task mapping does not aim for perfect mapping results but schemes with acceptable performance so that run-time task mapping is enabled. The mechanism is lightweight so that reasonable mapping performance can be reached with minimal complexity, energy and area overhead. Four different mapping decision processes and three neighbor evaluation approaches were studied. The demonstration shows that core load monitoring is essential in task mapping and it cannot be fully replaced with communication monitoring even though core load and communication load are related to each other.

The core load monitoring protocol steers tasks to the best cores in the known neighborhood. The goal is to make the process as simple and lightweight as possible. The optimal solution would choose not the best but the most suitable core, taking into account that it is not always profitable to accommodate small tasks to the cores with the largest free capacity. Then, for instance, small task would be mapped on a core which has free capacity just for a small task and it cannot accommodate larger tasks. However, if the task sizes are equalized the difference between lightweight and optimal solutions decreases.

In this chapter the basic functionality of the presented task mapping algorithm was

demonstrated in a form of a case study. In future works a more comprehensive analysis should be carried out to generalize the results and adjust the features of the presented algorithm.

Chapter 8

Conclusions

Dynamic monitoring methods for Networks-on-Chip were studied and a dynamically clustered distributed monitoring approach was presented and analyzed in this thesis. Essential background information concerning principles of Networks-on-Chip were studied including different topologies, routing algorithms, network flow control as well as common problems on NoCs. Monitoring of Networks-on-Chip was discussed and different monitoring structures were studied. Finally, issues concerning the presented dynamically clustered monitoring structure (DCM) were discussed and analyzed using an in-house simulation environment.

Our analysis shows that centralized monitoring is slow and causes significant communication overhead to the system while there are tasks, especially related to traffic management, which can be implemented without centralized control. A selection between dedicated and shared resources is an essential decision in terms of intrusiveness, performance and area overhead of a monitoring system. If the designer decides to use dedicated resources, it could be reasonable to use centralized monitoring. Otherwise, if shared resources are used, at least network management should be distributed to minimize the need for centralized control and the complexity of centralized monitors. In a large, complex system it may also be reasonable to use hybrid monitoring structures. A hybrid monitoring structure has both distributed and centralized monitoring services, possibly also clustered services, in the same system. Relatively simple and frequently repeated network monitoring tasks are carried out using distributed monitoring which optimizes the network resource utilization. The centralized monitoring is typically needed for system diagnostic purposes and for tasks which cannot be executed without centralized control. Traffic management is basically quite regional which means that most of the traffic related problems can be solved using the resources near the problematic areas. In most of the cases there is no need to know the situation at the other side of the network but the status of the neighbors, their neighbors and so on as far as required. This decreases the need for centralized control and releases resources for other tasks, for instance load balancing, optimization and reconfiguration.

The work, presented in this thesis, includes extensive analysis of the dynamically

clustered distributed monitoring approach. To make the analysis possible an in-house SystemC based NoC simulation environment was designed and implemented. It has been proved to be an efficient tool for analyzing and simulating different aspects in Networks-on-Chip. The simulation environment can be easily configured for analysis of different features which makes it a flexible tool for early phase analysis. The simulation environment was used to analyze monitoring algorithms, monitoring data diffusion areas, the format of monitoring data and communication as well as the number of monitors. The presented research shows that in most cases simple monitoring algorithms and small monitoring cluster areas perform at least as well as more complex implementations. In this thesis the added value of more complex monitoring structures was found small or negligible in terms of performance. Thus, acceptable monitoring performance can be reached using relatively simple monitoring structures. Another observation is that even small adjustments in the system parameters, such as monitoring data update frequencies or numbers of used monitors, can have a significant influence to the overall performance. Therefore the parameters should be chosen carefully while designing complex distributed monitoring structures.

The presented DCM structure was applied for task management to analyze its flexibility for purposes other than traffic management. The presented distributed task mapping method was analyzed with the NoC simulation environment using different mapping decision strategies and various neighbor evaluation approaches. The analysis showed that the dynamically clustered monitoring structure can be utilized for autonomous distributed task mapping. It was also noted that proper functionality of the distributed task mapping requires accurate information about computational load of cores. Even though the communication resource load is related to computational load, it is not accurate enough to be applied for task mapping purpose.

The proposed distributed monitoring system is flexible and scalable, which makes is suitable for Networks-on-Chip because flexibility and scalability are natural characteristics of these on-chip networks. Distributing the functionality and partitioning the system into smaller parts improves system level fault tolerance because then a faulty component cannot have severe influence on the overall functionality of the system. The presented analysis shows that distributed monitoring systems load the network less than typical centralized systems. Fault tolerance, communication overhead and diffusion of monitoring data can be improved by clustering a system into smaller parts. However, clustered structures do not reach the performance of totally distributed monitoring systems. The main drawback of distributed monitoring is the lack of global status information which may be required in reconfiguration tasks such as system-level energy consumption optimization. The outcome is that in large and complex systems it can be beneficial to have both centralized and distributed structures.

Performance of the DCM structure could be adjusted by using different sized monitoring clusters in different areas in the network. Areas with low traffic load may work at reasonable performance using very simple deterministic routing algorithms. At the same time in the same system, there could be performance critical areas with high traffic loads and strict quality of service requirements. Larger monitoring clusters and

adaptive routing algorithms are suitable for these critical areas. The cluster sizes and routing algorithms can be made run time reconfigurable to enable extensive optimization capability.

In design of complex multiprocessor systems, the focus should be kept on the computational features while the implementation of the communication infrastructure should be straightforward without time consuming design of essential structures. Reusable and fully scalable distributed monitoring structures can be utilized in systems of different sizes without complicated customization processes. The structures should be developed towards general communication platforms which conform to different systems with varying performance and operational requirements.

The work presented in this thesis has notable potential for further development. The presented NoC simulation environment can be extended and developed for analysis of distinct new systems and features. The foundations of dynamically clustered network monitoring have been presented and analyzed in this thesis. Future work should lead the DCM structure towards physical implementations as parts of high performance multiprocessor systems.

Bibliography

[1] SystemC. http://www.accellera.org/.

[2] M.A. Al Faruque, T. Ebi, and J. Henkel. ROAdNoC: Runtime observability for an adaptive Network on Chip architecture. In *IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2008*, pages 543–548, November 2008.

[3] M.A. Al Faruque, T. Ebi, and J. Henkel. AdNoC: Runtime adaptive network-on-chip architecture. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 20(2):257–269, February 2012.

[4] M.A. Al Faruque, R. Krist, and J. Henkel. Adam: Run-time agent-based distributed application mapping for on-chip communication. In *45th ACM/IEEE Design Automation Conference, DAC 2008*, pages 760–765, June 2008.

[5] M. Ali, M. Welzl, and S. Hellebrand. A dynamic routing mechanism for network on chip. In *23rd NORCHIP Conference*, pages 70–73, November 2005.

[6] S. Amstutz. A new store and forward message switching system. *IEEE Transactions on Communication Technology*, 19(4):528 –529, August 1971.

[7] D. Andreasson and S. Kumar. Slack-time aware routing in noc systems. In *IEEE International Symposium on Circuits and Systems*, pages 2353–2356, May 2005.

[8] N. Bansal, A. Blum, S. Chawla, and A. Meyerson. Online oblivious routing. In *Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 44–49, 2003.

[9] L. Benini and G. De Micheli. Networks on Chip: a new paradigm for Systems on Chip design. In *Design, Automation & Test in Europe, DATE '02*, pages 418–419, 2002.

[10] D. Bertozzi, L. Benini, and G. De Micheli. Error control schemes for on-chip communication links: the energy-reliability tradeoff. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(6):818–831, June 2005.

[11] A. Borella, G. Cancellieri, F. Chiaraluce, and F. Meschini. Tree topologies in atm networks. In *Singapore ICCS/ISITA '92. 'Communications on the Move'*, volume 1, pages 162–166, November 1992.

[12] C. Ciordas, T. Basten, A. Radulescu, K. Goossens, and J. Van Meerbergen. An event-based monitoring service for Networks on Chip. *ACM Transactions on Design Automation of Electronic Systems*, 10(4):702–723, 2005.

[13] C. Ciordas, K. Goossens, A. Radulescu, and T. Basten. NoC monitoring: impact on the design flow. In *IEEE International Symposium on Circuits and Systems, ISCAS 2006*, pages 1981–1984, 2006.

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

[15] W.J. Dally. Virtual-channel flow control. *IEEE Transactions on Parallel and Distributed Systems*, 3(2):194 –205, March 1992.

[16] W.J. Dally and B. Towles. Route packets, not wires: On-chip interconnection networks. In *Design Automation Conference*, pages 684–689, 2001.

[17] M. Dehyadgari, M. Nickray, A. Afzali-kusha, and Z. Navabi. Evaluation of pseudo adaptive xy routing using an object oriented model for noc. In *The 17th International Conference on Microelectronics*, December 2005.

[18] J. Duato, S. Yalamanchili, and L. Ni. *Interconnection Networks, an Engineering Approach.* Morgan Kaufmann, 2003.

[19] T. Dumitras and R. Marculescu. On-chip stochastic communication. In *Design, Automation and Test in Europe Conference and Exhibition, 2003*, pages 790 – 795, 2003.

[20] R. Farivar, M. Fazeli, and S.G. Miremadi. Directed flooding: a fault-tolerant routing protocol for wireless sensor networks. In *Proceedings of Systems Communications*, pages 395 – 399, August 2005.

[21] U. Feige and P. Raghavan. Exact analysis of hot-potato routing. In *33rd Annual Symposium on Foundations of Computer Science*, pages 553–562, October 1992.

[22] D. Fick, A. DeOrio, Jin Hu, V. Bertacco, D. Blaauw, and D. Sylvester. Vicis: A reliable network for unreliable silicon. In *46th ACM/IEEE Design Automation Conference, DAC '09*, pages 812–817, July 2009.

[23] L. Fiorin, G. Palermo, and C. Silvano. MPSoCs run-time monitoring through Networks-on-Chip. In *Design, Automation & Test in Europe, DATE '09*, pages 558–561, April 2009.

[24] K. Goossens, J. Dielissen, and A. Radulescu. Æthereal Network on Chip: Concepts, architectures and implementations. In *IEEE Design & Test of Computers*, volume 22, pages 414–421, 2005.

[25] P. Gratz, B. Grot, and S.W. Keckler. Regional congestion awareness for load balance in networks-on-chip. In *IEEE 14th International Symposium on High Performance Computer Architecture, HPCA 2008*, pages 203–214, February 2008.

[26] C. Grecu, A. Ivanov, R. Saleh, and P.P. Pande. Testing network-on-chip communication fabrics. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(12):2201–2214, December 2007.

[27] C. Grecu, P. Pande, A. Ivanov, and R. Saleh. BIST for network-on-chip interconnect infrastructures. In *24th IEEE VLSI Test Symposium*, pages 30–35, May 2006.

[28] K. Hamwi and O. Hammami. Design and implementation of mpsoc single chip with butterfly network. In *18th IEEE/IFIP VLSI System on Chip Conference (VLSI-SoC)*, pages 143–148, September 2010.

[29] J. Hu and R. Marculescu. DyAD – smart routing for networks-on-chip. In *Proceedings, 41st Design Automation Conference*, pages 260–263, 2004.

[30] P. Ituero, M. Lopez-Vallejo, M.A.S. Marcos, and C.G. Osuna. On-chip monitoring: A light-weight interconnection network approach. In *14th Euromicro Conference on Digital System Design (DSD)*, pages 619–625, September 2011.

[31] B. W. Johnson. *Design and Analysis of Fault-Tolerant Digital Systems*. Addison-Wesley, 1989.

[32] H. Kariniemi and J. Nurmi. Arbitration and routing schemes for on-chip packet networks. In J. Nurmi, H. Tenhunen, J. Isoaho, and A. Jantsch, editors, *Interconnect-Centric Design for Advanced SoC and NoC*, pages 253–282. Kluwer Academic Publishers, 2004.

[33] W. Karl, M. Leberecht, and M. Oberhuber. SCI monitoring hardware and software: Supporting performance evaluation and debugging. In Hermann Hellwagner and Alexander Reinefeld, editors, *SCI: Scalable Coherent Interface*, volume 1734, pages 417–432. Springer Berlin, Heidelberg, 1999.

[34] Parviz Kermani and Leonard Kleinrock. Virtual cut-through: A new computer communication switching technique. *Computer Networks*, 3:267–286, 1979.

[35] J. Kim, D. Park, T. Theocharides, N. Vijaykrishnan, and C.R. Das. A low latency router supporting adaptivity for on-chip interconnects. In *Proceedings, 42. Design Automation Conference*, pages 559–564, June 2005.

[36] T. Lehtonen. *On Fault Tolerant Methods for Networks-on-Chip*. PhD thesis, Turku Centre for Computer Science (TUCS), Turku, Finland, October 2009.

[37] T. Lehtonen, D. Wolpert, P. Liljeberg, J. Plosila, and P. Ampadu. Self-adaptive system for addressing permanent errors in on-chip interconnects. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 18(4):527–540, April 2010.

[38] H. Li and M. Maresca. Polymorphic-torus network. *Computers, IEEE Transactions on*, 38(9):1345–1351, September 1989.

[39] Zhonghai Lu and Axel Jantsch. Tdm virtual-circuit configuration for network-on-chip. *IEEE Transactions on Very Large Scale Integration Systesms*, 16(8):1021–1034, August 2008.

[40] M. Majer, C. Bobda, A. Ahmadinia, and J. Teich. Packet routing in dynamically changing networks on chip. In *Proceedings, 19th IEEE International Parallel and Distributed Processing Symposium*, April 2005.

[41] T. Marescaux, A. Rångevall, V. Nollet, A. Bartic, and H. Corporaal. Distributed congestion control for packet switched Networks on Chip. In *International Conference ParCo 2005*, pages 761–768, 2005.

[42] G. De Micheli and L. Benini. *Networks on Chips*. Morgan Kaufmann, 2006.

[43] J.M. Montanana, M. Koibuchi, H. Matsutani, and H. Amano. Balanced dimension-order routing for k-ary n-cubes. In *International Conference on Parallel Processing Workshops, ICPPW '09.*, pages 499 –506, September 2009.

[44] R.B. Mouhoub and O. Hammami. NoC monitoring hardware support for fast NoC design space exploration and potential NoC partial dynamic reconfiguration. In *International Symposium on Industrial Embedded Systems, IES '06*, pages 1–10, October 2006.

[45] S. Murali, T. Theocharides, N. Vijaykrishnan, M. Irwin, L. Benini, and G. De Micheli. Analysis of error recovery schemes for networks on chips. *IEEE Design & Test of Computers*, 22(5):434–442, September-October 2005.

[46] E. Nilsson, M. Millberg, J. Oberg, and A. Jantsch. Load distribution with the proximity congestion awareness in a Network on Chip. In *Design, Automation & Test in Europe, DATE '03*, 2003.

[47] V. Nollet, T. Marescaux, and D. Verkest. Operating-system controlled Network on Chip. In *41st Design Automation Conference*, pages 256–259, 2004.

[48] J. Nurmi. Network-on-chip: A new paradigm for system-on-chip design. In *Proceedings 2005 International Symposium on System-on-Chip*, pages 2–6, August 2005.

[49] J. Nurmi, H. Tenhunen, J. Isoaho, and A. Jantsch, editors. *Interconnect-centric Design for Advanced SoC and NoC*. Kluwer Academic Publishers, Dordrecht, Netherlands, 2004.

[50] K. Oommen and D. Harle. Hardware emulation of a network on chip architecture based on a clockwork routed manhattan street network. In *International Conference on Field Programmable Logic and Applications*, pages 727–728, August 2005.

[51] S. Pasricha and N. Dutt. *On-Chip Communication Architectures: System on Chip Interconnect*. Morgan Kaufmann, 2008.

[52] R. J. Perlman. Fault-tolerant broadcast of routing information. In *INFOCOM*, pages 93–102, 1983.

[53] M. Pirretti, G.M. Link, R.R. Brooks, N. Vijaykrishnan, M. Kandemir, and M.J. Irwin. Fault tolerant algorithms for network-on-chip interconnect. In *Proceedings of IEEE Computer society Annual Symposium on VLSI, 2004.*, pages 46 – 51, February 2004.

[54] J. M. Rabaey. *Digital Integrated Circuits: A Design Perspective*. Prentice Hall, Inc, Upper Saddle River, NJ, 1996.

[55] V. Rantala, T. Lehtonen, P Liljeberg, and J. Plosila. Analysis of monitoring structures for network-on-chip – a distributed approach. *IGI International Journal of Embedded and Real-Time Communication Systems (IJERTCS)*, 2(1):49–67, 2011.

[56] V. Rantala, T. Lehtonen, P. Liljeberg, and J. Plosila. Analysis of status data update in dynamically clustered network-on-chip monitoring. In *1st International Conference on Pervasive and Embedded Computing and Communication Systems, PECCS 2011*, March 2011.

[57] V. Rantala, T. Lehtonen, and J. Plosila. Network on chip routing algorithms. Technical Report 779, TUCS Technical Report, August 2006.

[58] V. Rantala, P. Liljeberg, and J. Plosila. Status data and communication aspects in dynamically clustered network-on-chip monitoring. *Journal of Electrical and Computer Engineering*, 2012(2012), 2012.

[59] E. Rijpkema, K. Goossens, and P. Wielage. A router architecture for networks on silicon. In *Proceedings of Progress 2001, 2nd Workshop on Embedded Systems*, 2001.

[60] L. G. Roberts and B. D. Wessler. Computer network development to achieve resource sharing. In *Proceedings of the Spring Joint Computer Conference*, AFIPS '70 (Spring), pages 543–549, 1970.

[61] A.K. Singh, T. Srikanthan, A. Kumar, and W. Jigang. Communication-aware heuristics for run-time task mapping on noc-based mpsoc platforms. *Journal of Systems Architecture*, 56:242–255, July 2010.

[62] J. Srinivasan, S.V. Adve, P. Bose, and J.A. Rivers. The case for lifetime reliability-aware microprocessors. In *31st Annual International Symposium on Computer Architecture*, pages 276 – 287, June 2004.

[63] Bjarne Stroustrup. *The C++ Programming Language.* Addison-Wesley, 1997.

[64] M. Tagel, P. Ellervee, and G. Jervan. Scheduling framework for real-time dependable noc-based systems. In *Proceedings of the International Symposium on System-on-Chip (SOC 2009)*, pages 95–99, October 2009.

[65] K.-C. Tai. Definitions and detection of deadlock, livelock, and starvation in concurrent programs. In *International Conference on Parallel Processing, ICPP 1994.*, volume 2, pages 69 –72, August 1994.

[66] Z. Tianxu and D. Xuchao. Reliability estimation model of ic's interconnect based on uniform distribution of defects on a chip. In *Proceedings of the 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pages 11–17, November 2003.

[67] B. Towles, W. J. Dally, and S. Boyd. Throughput-centric routing algorithm design. In *Proceedings, 15th ACM symposium on Parallel algorithms and architectures*, pages 200–209, 2003.

[68] L. G. Valiant and G. J. Brebner. Universal schemes for parallel communication. In *Proceedings of the thirteenth annual ACM symposium on Theory of computing*, STOC '81, pages 263–277, New York, NY, USA, 1981. ACM.

[69] J.W. van den Brand, C. Ciordas, K. Goossens, and T. Basten. Congestion-controlled best-effort communication for Networks-on-Chip. In *Design, Automation & Test in Europe, DATE '07*, pages 1–6, April 2007.

[70] H.-S. P. Wong, D. J. Frank, P. M. Solomon, C. H. J. Wann, and J. J. Welser. *Emerging Nanoelectronics: Life with and after CMOS (Vol. 1)*, chapter Nanoscale CMOS, pages 46–83. Kluwer Academic Publishers, Norwell, MA, 2005.

[71] M. Yang, T. Li, Y. Jiang, and Y. Yang. Fault-tolerant routing schemes in rdt(2,2,1)/α-based interconnection network for networks-on-chip designs. In *Proceedings, 8th International Symposium on Parallel Architectures, Algorithms and Networks*, December 2005.

[72] Zhang Ying, Wu Ning, Wan Yu Peng, Ge Fen, and Zhou Fang. Fault-tolerant schemes for NoC with a network monitor. In *International Symposium on Communications and Information Technologies (ISCIT)*, pages 1083 –1086, October 2010.

[73] Jia Zhao, S. Madduri, R. Vadlamani, W. Burleson, and R. Tessier. A dedicated monitoring infrastructure for multicore processors. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 19(6):1011–1022, June 2011.

[74] P. Zipf, G. Sassatelli, N. Utlu, N. Saint-Jean, P. Benoit, and M. Glesner. A decentralized task mapping approach for homogeneous multiprocessor network-on-chips. *International Journal of Reconfigurable Computing*, 2009, 2009.

Turku Centre for Computer Science
TUCS Dissertations

1. **Marjo Lipponen**, On Primitive Solutions of the Post Correspondence Problem
2. **Timo Käkölä**, Dual Information Systems in Hyperknowledge Organizations
3. **Ville Leppänen**, Studies on the Realization of PRAM
4. **Cunsheng Ding**, Cryptographic Counter Generators
5. **Sami Viitanen**, Some New Global Optimization Algorithms
6. **Tapio Salakoski**, Representative Classification of Protein Structures
7. **Thomas Långbacka**, An Interactive Environment Supporting the Development of Formally Correct Programs
8. **Thomas Finne**, A Decision Support System for Improving Information Security
9. **Valeria Mihalache**, Cooperation, Communication, Control. Investigations on Grammar Systems.
10. **Marina Waldén**, Formal Reasoning About Distributed Algorithms
11. **Tero Laihonen**, Estimates on the Covering Radius When the Dual Distance is Known
12. **Lucian Ilie**, Decision Problems on Orders of Words
13. **Jukkapekka Hekanaho**, An Evolutionary Approach to Concept Learning
14. **Jouni Järvinen**, Knowledge Representation and Rough Sets
15. **Tomi Pasanen**, In-Place Algorithms for Sorting Problems
16. **Mika Johnsson**, Operational and Tactical Level Optimization in Printed Circuit Board Assembly
17. **Mats Aspnäs**, Multiprocessor Architecture and Programming: The Hathi-2 System
18. **Anna Mikhajlova**, Ensuring Correctness of Object and Component Systems
19. **Vesa Torvinen**, Construction and Evaluation of the Labour Game Method
20. **Jorma Boberg**, Cluster Analysis. A Mathematical Approach with Applications to Protein Structures
21. **Leonid Mikhajlov**, Software Reuse Mechanisms and Techniques: Safety Versus Flexibility
22. **Timo Kaukoranta**, Iterative and Hierarchical Methods for Codebook Generation in Vector Quantization
23. **Gábor Magyar**, On Solution Approaches for Some Industrially Motivated Combinatorial Optimization Problems
24. **Linas Laibinis**, Mechanised Formal Reasoning About Modular Programs
25. **Shuhua Liu**, Improving Executive Support in Strategic Scanning with Software Agent Systems
26. **Jaakko Järvi**, New Techniques in Generic Programming – C++ is more Intentional than Intended
27. **Jan-Christian Lehtinen**, Reproducing Kernel Splines in the Analysis of Medical Data
28. **Martin Büchi**, Safe Language Mechanisms for Modularization and Concurrency
29. **Elena Troubitsyna**, Stepwise Development of Dependable Systems
30. **Janne Näppi**, Computer-Assisted Diagnosis of Breast Calcifications
31. **Jianming Liang**, Dynamic Chest Images Analysis
32. **Tiberiu Seceleanu**, Systematic Design of Synchronous Digital Circuits
33. **Tero Aittokallio**, Characterization and Modelling of the Cardiorespiratory System in Sleep-Disordered Breathing
34. **Ivan Porres**, Modeling and Analyzing Software Behavior in UML
35. **Mauno Rönkkö**, Stepwise Development of Hybrid Systems
36. **Jouni Smed**, Production Planning in Printed Circuit Board Assembly
37. **Vesa Halava**, The Post Correspondence Problem for Market Morphisms
38. **Ion Petre**, Commutation Problems on Sets of Words and Formal Power Series
39. **Vladimir Kvassov**, Information Technology and the Productivity of Managerial Work
40. **Frank Tétard**, Managers, Fragmentation of Working Time, and Information Systems

41. **Jan Manuch**, Defect Theorems and Infinite Words
42. **Kalle Ranto**, Z_4-Goethals Codes, Decoding and Designs
43. **Arto Lepistö**, On Relations Between Local and Global Periodicity
44. **Mika Hirvensalo**, Studies on Boolean Functions Related to Quantum Computing
45. **Pentti Virtanen**, Measuring and Improving Component-Based Software Development
46. **Adekunle Okunoye**, Knowledge Management and Global Diversity – A Framework to Support Organisations in Developing Countries
47. **Antonina Kloptchenko**, Text Mining Based on the Prototype Matching Method
48. **Juha Kivijärvi**, Optimization Methods for Clustering
49. **Rimvydas Rukšėnas**, Formal Development of Concurrent Components
50. **Dirk Nowotka**, Periodicity and Unbordered Factors of Words
51. **Attila Gyenesei**, Discovering Frequent Fuzzy Patterns in Relations of Quantitative Attributes
52. **Petteri Kaitovaara**, Packaging of IT Services – Conceptual and Empirical Studies
53. **Petri Rosendahl**, Niho Type Cross-Correlation Functions and Related Equations
54. **Péter Majlender**, A Normative Approach to Possibility Theory and Soft Decision Support
55. **Seppo Virtanen**, A Framework for Rapid Design and Evaluation of Protocol Processors
56. **Tomas Eklund**, The Self-Organizing Map in Financial Benchmarking
57. **Mikael Collan**, Giga-Investments: Modelling the Valuation of Very Large Industrial Real Investments
58. **Dag Björklund**, A Kernel Language for Unified Code Synthesis
59. **Shengnan Han**, Understanding User Adoption of Mobile Technology: Focusing on Physicians in Finland
60. **Irina Georgescu**, Rational Choice and Revealed Preference: A Fuzzy Approach
61. **Ping Yan**, Limit Cycles for Generalized Liénard-Type and Lotka-Volterra Systems
62. **Joonas Lehtinen**, Coding of Wavelet-Transformed Images
63. **Tommi Meskanen,** On the NTRU Cryptosystem
64. **Saeed Salehi**, Varieties of Tree Languages
65. **Jukka Arvo**, Efficient Algorithms for Hardware-Accelerated Shadow Computation
66. **Mika Hirvikorpi**, On the Tactical Level Production Planning in Flexible Manufacturing Systems
67. **Adrian Costea**, Computational Intelligence Methods for Quantitative Data Mining
68. **Cristina Seceleanu**, A Methodology for Constructing Correct Reactive Systems
69. **Luigia Petre**, Modeling with Action Systems
70. **Lu Yan**, Systematic Design of Ubiquitous Systems
71. **Mehran Gomari**, On the Generalization Ability of Bayesian Neural Networks
72. **Ville Harkke**, Knowledge Freedom for Medical Professionals – An Evaluation Study of a Mobile Information System for Physicians in Finland
73. **Marius Cosmin Codrea**, Pattern Analysis of Chlorophyll Fluorescence Signals
74. **Aiying Rong**, Cogeneration Planning Under the Deregulated Power Market and Emissions Trading Scheme
75. **Chihab BenMoussa**, Supporting the Sales Force through Mobile Information and Communication Technologies: Focusing on the Pharmaceutical Sales Force
76. **Jussi Salmi**, Improving Data Analysis in Proteomics
77. **Orieta Celiku**, Mechanized Reasoning for Dually-Nondeterministic and Probabilistic Programs
78. **Kaj-Mikael Björk**, Supply Chain Efficiency with Some Forest Industry Improvements
79. **Viorel Preoteasa**, Program Variables – The Core of Mechanical Reasoning about Imperative Programs
80. **Jonne Poikonen**, Absolute Value Extraction and Order Statistic Filtering for a Mixed-Mode Array Image Processor
81. **Luka Milovanov**, Agile Software Development in an Academic Environment
82. **Francisco Augusto Alcaraz Garcia**, Real Options, Default Risk and Soft Applications
83. **Kai K. Kimppa**, Problems with the Justification of Intellectual Property Rights in Relation to Software and Other Digitally Distributable Media
84. **Dragoş Truşcan**, Model Driven Development of Programmable Architectures
85. **Eugen Czeizler**, The Inverse Neighborhood Problem and Applications of Welch Sets in Automata Theory

86. **Sanna Ranto**, Identifying and Locating-Dominating Codes in Binary Hamming Spaces
87. **Tuomas Hakkarainen**, On the Computation of the Class Numbers of Real Abelian Fields
88. **Elena Czeizler**, Intricacies of Word Equations
89. **Marcus Alanen**, A Metamodeling Framework for Software Engineering
90. **Filip Ginter**, Towards Information Extraction in the Biomedical Domain: Methods and Resources
91. **Jarkko Paavola**, Signature Ensembles and Receiver Structures for Oversaturated Synchronous DS-CDMA Systems
92. **Arho Virkki**, The Human Respiratory System: Modelling, Analysis and Control
93. **Olli Luoma**, Efficient Methods for Storing and Querying XML Data with Relational Databases
94. **Dubravka Ilić**, Formal Reasoning about Dependability in Model-Driven Development
95. **Kim Solin**, Abstract Algebra of Program Refinement
96. **Tomi Westerlund**, Time Aware Modelling and Analysis of Systems-on-Chip
97. **Kalle Saari**, On the Frequency and Periodicity of Infinite Words
98. **Tomi Kärki**, Similarity Relations on Words: Relational Codes and Periods
99. **Markus M. Mäkelä**, Essays on Software Product Development: A Strategic Management Viewpoint
100. **Roope Vehkalahti**, Class Field Theoretic Methods in the Design of Lattice Signal Constellations
101. **Anne-Maria Ernvall-Hytönen**, On Short Exponential Sums Involving Fourier Coefficients of Holomorphic Cusp Forms
102. **Chang Li**, Parallelism and Complexity in Gene Assembly
103. **Tapio Pahikkala**, New Kernel Functions and Learning Methods for Text and Data Mining
104. **Denis Shestakov**, Search Interfaces on the Web: Querying and Characterizing
105. **Sampo Pyysalo**, A Dependency Parsing Approach to Biomedical Text Mining
106. **Anna Sell**, Mobile Digital Calendars in Knowledge Work
107. **Dorina Marghescu**, Evaluating Multidimensional Visualization Techniques in Data Mining Tasks
108. **Tero Säntti**, A Co-Processor Approach for Efficient Java Execution in Embedded Systems
109. **Kari Salonen**, Setup Optimization in High-Mix Surface Mount PCB Assembly
110. **Pontus Boström**, Formal Design and Verification of Systems Using Domain-Specific Languages
111. **Camilla J. Hollanti**, Order-Theoretic Mehtods for Space-Time Coding: Symmetric and Asymmetric Designs
112. **Heidi Himmanen**, On Transmission System Design for Wireless Broadcasting
113. **Sébastien Lafond**, Simulation of Embedded Systems for Energy Consumption Estimation
114. **Evgeni Tsivtsivadze**, Learning Preferences with Kernel-Based Methods
115. **Petri Salmela**, On Commutation and Conjugacy of Rational Languages and the Fixed Point Method
116. **Siamak Taati**, Conservation Laws in Cellular Automata
117. **Vladimir Rogojin**, Gene Assembly in Stichotrichous Ciliates: Elementary Operations, Parallelism and Computation
118. **Alexey Dudkov**, Chip and Signature Interleaving in DS CDMA Systems
119. **Janne Savela**, Role of Selected Spectral Attributes in the Perception of Synthetic Vowels
120. **Kristian Nybom**, Low-Density Parity-Check Codes for Wireless Datacast Networks
121. **Johanna Tuominen**, Formal Power Analysis of Systems-on-Chip
122. **Teijo Lehtonen**, On Fault Tolerance Methods for Networks-on-Chip
123. **Eeva Suvitie**, On Inner Products Involving Holomorphic Cusp Forms and Maass Forms
124. **Linda Mannila**, Teaching Mathematics and Programming – New Approaches with Empirical Evaluation
125. **Hanna Suominen**, Machine Learning and Clinical Text: Supporting Health Information Flow
126. **Tuomo Saarni**, Segmental Durations of Speech
127. **Johannes Eriksson**, Tool-Supported Invariant-Based Programming

128. **Tero Jokela**, Design and Analysis of Forward Error Control Coding and Signaling for Guaranteeing QoS in Wireless Broadcast Systems
129. **Ville Lukkarila**, On Undecidable Dynamical Properties of Reversible One-Dimensional Cellular Automata
130. **Qaisar Ahmad Malik**, Combining Model-Based Testing and Stepwise Formal Development
131. **Mikko-Jussi Laakso**, Promoting Programming Learning: Engagement, Automatic Assessment with Immediate Feedback in Visualizations
132. **Riikka Vuokko**, A Practice Perspective on Organizational Implementation of Information Technology
133. **Jeanette Heidenberg**, Towards Increased Productivity and Quality in Software Development Using Agile, Lean and Collaborative Approaches
134. **Yong Liu**, Solving the Puzzle of Mobile Learning Adoption
135. **Stina Ojala**, Towards an Integrative Information Society: Studies on Individuality in Speech and Sign
136. **Matteo Brunelli**, Some Advances in Mathematical Models for Preference Relations
137. **Ville Junnila**, On Identifying and Locating-Dominating Codes
138. **Andrzej Mizera**, Methods for Construction and Analysis of Computational Models in Systems Biology. Applications to the Modelling of the Heat Shock Response and the Self-Assembly of Intermediate Filaments.
139. **Csaba Ráduly-Baka**, Algorithmic Solutions for Combinatorial Problems in Resource Management of Manufacturing Environments
140. **Jari Kyngäs**, Solving Challenging Real-World Scheduling Problems
141. **Arho Suominen**, Notes on Emerging Technologies
142. **József Mezei**, A Quantitative View on Fuzzy Numbers
143. **Marta Olszewska**, On the Impact of Rigorous Approaches on the Quality of Development
144. **Antti Airola**, Kernel-Based Ranking: Methods for Learning and Performace Estimation
145. **Aleksi Saarela**, Word Equations and Related Topics: Independence, Decidability and Characterizations
146. **Lasse Bergroth**, Kahden merkkijonon pisimmän yhteisen alijonon ongelma ja sen ratkaiseminen
147. **Thomas Canhao Xu**, Hardware/Software Co-Design for Multicore Architectures
148. **Tuomas Mäkilä**, Software Development Process Modeling – Developers Perspective to Contemporary Modeling Techniques
149. **Shahrokh Nikou**, Opening the Black-Box of IT Artifacts: Looking into Mobile Service Characteristics and Individual Perception
150. **Alessandro Buoni**, Fraud Detection in the Banking Sector: A Multi-Agent Approach
151. **Mats Neovius**, Trustworthy Context Dependency in Ubiquitous Systems
152. **Fredrik Degerlund**, Scheduling of Guarded Command Based Models
153. **Amir-Mohammad Rahmani-Sane**, Exploration and Design of Power-Efficient Networked Many-Core Systems
154. **Ville Rantala**, On Dynamic Monitoring Methods for Networks-on-Chip

Turku Centre *for* Computer Science

Joukahaisenkatu 3-5 B, 20520 Turku, Finland | www. tucs.fi

University of Turku
Faculty of Mathematics and Natural Sciences
- Department of Information Technology
- Department of Mathematics and Statistics

Turku School of Economics
- Institute of Information Systems Science

Åbo Akademi University
Division for Natural Sciences and Technology
- Department of Information Technologies

ISBN 978-952-12-2823-0
ISSN 1239-1883

Ville Rantala

On Dynamic Monitoring Methods for Networks-on-Chip

Ville Rantala

On Dynamic Monitoring Methods for Networks-on-Chip