MASTER

Runtime networks-on-chip performance monitoring

van den Brand, J.W.M

Award date:
2005

Link to publication

Disclaimer
This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
RUNTIME NETWORKS-ON-CHIP PERFORMANCE MONITORING.

J.M.W. van den Brand

Supervisor: prof.dr.ir. J.L. van Meerbergen (Philips Research)
Coach: dr.ir. A.A. Basten
Ir. C. Ciordas
Date: August 2005
Runtime Networks-on-Chip Performance Monitoring
Concepts, design and implementation
J.W.M. van den Brand

Final report for Master of science project conducted from September 2004 - August 2005

Department of Electrical Engineering
Information and Communication Systems/Electronic Systems (ICS/ES)
Technische Universiteit Eindhoven

and

Embedded Systems Architectures on Silicon (ESAS)
Philips Research

Professor:
Prof. dr. ir. J.L. van Meerbergen (TU/e, Philips Research)

Supervisors:
Dr. ir. A.A. Basten (TU/e)
ir. Calin Ciordas (TU/e)
Abstract

Networks-on-Chip (NoC) are scalable interconnects for Systems-on-Chip (SoC). An event-based NoC monitoring service has been developed within the DEMONS project which supports runtime observability of NoC behavior. Runtime NoC performance monitoring is an instance of this monitoring service and enables observation of performance measures such as connection latency and link utilization, while a system is actually running. This can for instance be used to obtain network resource information for Quality-of-Service (QoS) management or to support communication centric debugging. This thesis investigates whether runtime NoC performance monitoring is feasible and how it can be used.

A framework is presented that identifies NoC performance measures. In order to prove feasibility, runtime NoC performance monitoring is designed and implemented for the Æthereal NoC. For most of the investigated performance measures, generated load heavily depends on the sampling period. Generated load is presented for each of the investigated performance measures.

Experiments are conducted for an MPEG application to obtain realistic communication cost figures. These experiments show that the costs for runtime NoC performance monitoring can be low compared to existing traffic, even for very small sampling periods. For instance, link utilization for all links in a 3x1 mesh using a sampling period of 600 ns introduces 1.46% extra traffic and requires 0.76% extra energy compared to the communication costs of the MPEG application.

This thesis presents NoC congestion control as part of QoS management as an application of runtime NoC performance monitoring. It is well known that an increase in network utilization results in an exponential increase of connection latencies. The goal of NoC congestion control is to bound connection latency by bounding NoC utilization. The suggested control scheme, model predictive control (MPC), combines model based predictions with runtime performance measurements. The control scheme is applied to an Æthereal NoC setup and is implemented to show how the control scheme performs in a realistic environment. For the presented example, latency is reduced from 200 ns per message to 110 ns per message. For the same example we observe a reaction speed of several microseconds when an MPC controller is used that measures link utilization and takes control actions each 600 ns. Experiments with the required link utilization measure have shown 0.19% of additional traffic and 0.15% of additional energy for communication, compared to the communication costs of the MPEG application.

The main conclusion of this thesis is that runtime NoC performance monitoring is feasible at reasonable costs and that it is usable for, for instance, resource monitoring for QoS management.
I hereby thank the people from both the Philips ESAS department and the TU/e ICS/ES department for giving me this opportunity. Special gratitude goes out to Calin Ciordas and Twan Basten for their support and guidance throughout the project and Professor Jef van Meerbergen for giving me the opportunity to fulfill my project in such an inspiring environment. Finally, I'd like to thank ESAS members Kees Goossens and Santiago Gonzalez Pestana for their support.
# Contents

Abstract ................................................................. ii
Acknowledgements ....................................................... iii
List of Figures ........................................................... vii
Glossary ................................................................. viii

1 Introduction 2
1.1 Problem definition .............................................. 2
1.2 Project environment ........................................... 3
1.3 Objectives ......................................................... 3
1.4 Main contributions .............................................. 3
1.5 Report organization .............................................. 4

2 Concepts and related work 5
2.1 Networks-on-Chip .................................................. 5
2.1.1 Basic concept ................................................ 5
2.1.2 IBtheral ...................................................... 5
2.2 NoC monitoring service ......................................... 7
2.3 Runtime NoC performance monitoring ....................... 9
2.3.1 Introduction .................................................. 9
2.3.2 Measurements and event generation ..................... 9
2.3.3 Transport of events from probe to MSA ................. 10
2.3.4 Interpretation of data ....................................... 10
2.3.5 Applications for runtime NoC performance monitoring 11
2.4 Runtime performance monitoring for the internet .......... 12

3 Performance measures 14
3.1 Framework .......................................................... 14
3.1.1 Introduction .................................................. 14
3.1.2 Performance measure targets ............................... 14
3.1.3 Performance measures per performance metric ........... 15
3.2 Summary of performance measures ............................. 18
3.3 Conclusions ........................................................ 20

4 Design and implementation 21
4.1 Measurements ...................................................... 21
4.1.1 Link utilization and link throughput ....................... 21
4.1.2 Router utilization ........................................... 22
4.1.3 Buffer utilization ........................................... 22
4.1.4 Connection latency ......................................... 22
4.2 Event generation .................................................. 24
## CONTENTS

4.2.1 Event format optimization ............................................ 24  
4.2.2 Link utilization and link throughput .............................. 24  
4.2.3 Router utilization .................................................... 25  
4.2.4 Buffer utilization ..................................................... 26  
4.2.5 Connection latency ................................................... 27  
4.3 Measurement non-intrusiveness ........................................ 28  
4.3.1 Introduction ........................................................... 28  
4.3.2 Detect monitoring connection by observing packet headers .... 29  
4.3.3 Add flag to flit sideband information ............................ 29  
4.4 Computation of performance measure from measurement data .... 29  
4.5 Support for multiple performance measures at a single probe ... 30  
4.6 Conclusions .................................................................... 30  

5 Experimental cost quantification 31  
5.1 Environment ................................................................. 31  
5.2 Cost measures .............................................................. 31  
5.3 Approach .................................................................... 31  
5.4 Experimental setups ...................................................... 32  
5.5 Link utilization ............................................................. 34  
5.6 Router utilization .......................................................... 36  
5.7 Connection latency ......................................................... 37  
5.8 Combining multiple performance measures ......................... 37  
5.9 Conclusions .................................................................... 38  

6 NoC congestion control 40  
6.1 Introduction ................................................................. 40  
6.1.1 Goal for NoC congestion control ................................. 40  
6.1.2 NoC congestion control as part of QoS .......................... 40  
6.1.3 Control theory ........................................................... 40  
6.1.4 Point of control ........................................................ 41  
6.1.5 Measured value ........................................................... 42  
6.2 NoC congestion control for Æthereal ............................... 42  
6.2.1 Introduction .............................................................. 42  
6.2.2 Characteristics of the control problem ......................... 43  
6.2.3 Model predictive control .............................................. 43  
6.2.4 Link utilization model for multiple sources .................... 44  
6.3 Experiments for Æthereal NoC congestion control ............... 45  
6.3.1 Introduction ............................................................ 45  
6.3.2 Matlab simulation ....................................................... 45  
6.3.3 Æthereal simulation results ........................................ 47  
6.4 Conclusions .................................................................... 47  

7 Conclusions and recommendations 55  
7.1 Conclusions .................................................................... 55  
7.2 Recommendations and future research ............................. 56  

References ............................................................................ 57  

A Verification of an SDF model of the Æthereal NoC 59
List of Figures

2.1 NoC with 3x3 mesh topology ........................................... 6
2.2 Probe connected to a router [1] ....................................... 8
2.3 NoC with several probes and a single MSA, one of the monitor connections is indicated with arrows ................. 8
2.4 Runtime NoC performance monitoring .................................. 9
2.5 Statistical interpretation of performance measurement showing sample average and sample standard deviation ......... 11
2.6 Performance measurement with $\Delta T$ chosen too high to observe sine wave shaped disturbance .................. 12

3.1 Performance measure targets framework ............................ 15
3.2 Target relationships in UML notation ................................. 16
3.3 Latency measures .......................................................... 17

4.1 Flit format for $\beta$thereal .............................................. 21
4.2 Connection latency .......................................................... 23
4.3 Generated load for single link utilization/throughput measure as a function of $\Delta T$ .............................................. 25
4.4 Generated load for router link utilization measure as a function of $\Delta T$ ......................................................... 26
4.5 Generated load for buffer utilization measure as a function of $\Delta T$ .............................................................. 27
4.6 Generated load for connection latency measure as a function of load$_C$ .......................................................... 28
4.7 $\beta$thereal packet header .................................................... 29

5.1 MPEG application using 3x2 NoC ......................................... 32
5.2 MPEG application using 3x1 NoC ......................................... 33
5.3 Link utilization of a link in the MPEG 3x2 mesh ....................... 34
5.4 Link utilization of a link in the MPEG 3x1 mesh ....................... 34
5.5 Router utilization for all routers in 3x2 mesh with $\Delta T = 1000$ (flitclocks) ..................................................... 36
5.6 Router utilization for all routers in 3x1 mesh with $\Delta T = 100$ (flitclocks) ..................................................... 37
5.7 Connection latency for single connection for connection from audio IP to memory ........................................... 39

6.1 Feedback control system .................................................... 41
6.2 Conceptual view of link utilization with and without controlled BE ......................................................... 43
6.3 MPC for NoC congestion control .......................................... 44
6.4 Model used for MPC .......................................................... 45
6.5 NoC configuration for (3x1 mesh with two connections) ............. 46
6.6 3x1 mesh with control for a single link ................................. 46
6.7 Different inputs for Matlab simulation .................................... 48
6.8 Open loop responses for different inputs ............................... 49
6.9 Responses for controller with control interval of 100 flit clocks ....... 50
LIST OF FIGURES

6.10 Responses for controller with control interval of 1 flit clock . . . 51
6.11 Sine wave faster than round-trip latency . . . . . . . . . . . . . 52
6.12 Link utilization (a) and latency (b) of a connection using the link
without NoC congestion control . . . . . . . . . . . . . . . . . . . . 53
6.13 Link utilization (a) and latency (b) of a connection using the link
with NoC congestion control . . . . . . . . . . . . . . . . . . . . . . . 54

A.1 Setup for SDF verification experiment . . . . . . . . . . . . . . . . . 59
A.2 SDF model for verification experiment, 1 flit clock (fc) equals 6 ns 60
A.3 Connection latency without critical end-to-end flow control . . . 61
A.4 SDF model for verification experiment with critical end-to-end flow
control, 1 flit clock (fc) equals 6 ns . . . . . . . . . . . . . . . . . . . . 62
A.5 Connection latency with critical end-to-end flow control . . . . . 62
## Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Abbr.</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arity</td>
<td></td>
<td>Number of inputs/outputs of a router, arity n means n inputs and n outputs</td>
</tr>
<tr>
<td>Best effort</td>
<td>BE</td>
<td>Connection property indicating a connection with no guarantees concerning bandwidth and latency</td>
</tr>
<tr>
<td>Channel</td>
<td></td>
<td>A channel is a pair of NIPs that communicate packets in one direction (Network layer communication service)</td>
</tr>
<tr>
<td>Connection</td>
<td></td>
<td>Transport layer communication service which transports messages between a pair of NIPs, consists of one or more channels</td>
</tr>
<tr>
<td>Flit</td>
<td></td>
<td>Smallest conceptual unit of communication, communicated inside network when using wormhole routing</td>
</tr>
<tr>
<td>Guaranteed throughput</td>
<td>GT</td>
<td>Connection property indicating guarantees for both throughput and latency; Æthereal GT uses slot reservations in a TDMA slot table to give guarantees</td>
</tr>
<tr>
<td>Intellectual Property</td>
<td>IP</td>
<td>Host that is connected to interconnect (e.g. processing unit or memory unit)</td>
</tr>
<tr>
<td>Interconnect</td>
<td></td>
<td>Communication subnet</td>
</tr>
<tr>
<td>Link</td>
<td></td>
<td>Physical means of communication from router/NI to another router/NI</td>
</tr>
<tr>
<td>Message</td>
<td></td>
<td>Transport layer communication unit. Messages are commonly split into smaller units called packets for communication over the network</td>
</tr>
<tr>
<td>Network Interface</td>
<td>NI</td>
<td>Interface between IPs and router network</td>
</tr>
<tr>
<td>Network Interface Port</td>
<td>NIP</td>
<td>Port for communication link between IP and NI</td>
</tr>
<tr>
<td>Network-on-Chip</td>
<td>NoC</td>
<td>On-chip interconnect in the form of a network, comprised of routers and network interfaces</td>
</tr>
<tr>
<td>Packet</td>
<td></td>
<td>Network layer communication unit for packet switched networks; when wormhole routing is used a packet is further split into flits</td>
</tr>
</tbody>
</table>
## GLOSSARY

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance measure</td>
<td>Quantity of performance for a certain part of the network defined by performance measure target and metric</td>
</tr>
<tr>
<td>Performance measure target</td>
<td>Entity for which performance measure is taken (e.g. link and connection)</td>
</tr>
<tr>
<td>Performance metric</td>
<td>Fundamental performance quantity (e.g. throughput and latency)</td>
</tr>
<tr>
<td>Port</td>
<td>Communication exit or entrance, the 'Æ'thereal NoC has network interface ports (NIP) at the IP side of a NI and router ports in the router network</td>
</tr>
<tr>
<td>Router</td>
<td>Node that switches the communicated data units, a network of routers switch data from one NI to another</td>
</tr>
<tr>
<td>Router port</td>
<td>Port for communication at routers and at the router network side of network interfaces</td>
</tr>
<tr>
<td>System-on-Chip</td>
<td>Combination of IPs and interconnect on a single chip</td>
</tr>
<tr>
<td>Time division multiple access</td>
<td>Scheme where physical resources are shared over time by multiple users</td>
</tr>
<tr>
<td>Transaction</td>
<td>A group of operations that must all succeed or fail. 'Æ'thereal uses a transaction based model with masters and slaves. Masters issue requests with a certain command (e.g. read or write). One or more slaves receive and execute each transaction. A transaction can also include a response that is issued by a slave. Such a response can include a transaction acknowledgement or data (e.g. requested by master).</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Problem definition

Modern multimedia applications require extensive computation facilities and embedded systems are widely used to fulfill this requirement. The ever-increasing demand for these complex systems in combination with the increased potential of on-chip complexity gave rise to a new breed of chips containing multiple communicating processing units. Networks-on-Chip (NoCs) provide these so-called Multiprocessor Systems-on-Chip (MP-SoCs) with a scalable and flexible interconnect [2]. Examples of NoCs currently under development are Æthereal [3; 4], Nostrum [5] and Xpipes [6].

Monitoring is the process of collecting information about system behavior. A runtime monitoring system collects information while a system is actually running, allowing a more profound understanding of the system's actual behavior. Hardware monitoring techniques have been little explored because of the huge amount of low-level data they generate. Nevertheless, they proved to be low intrusive.

Performance monitoring is a type of monitoring that focuses on performance measures such as throughput and latency. Runtime NoC performance information can for example be used to support debugging of performance problems and to supply a Quality-of-Service (QoS) system with actual NoC performance information.

An efficient NoC monitoring mechanism capable of adapting to NoC monitoring demands (driven for example by debugging) on the fly is developed within the DEMONS (Debugging and MOnitoring Networks on Silicon) project [1; 7]. It addresses the data explosion problem by introducing on-chip abstraction of low-level data into events.

The current NoC monitoring solution uses the monitored NoC to transport monitor data. The load that is inserted in the NoC by the monitoring service should be low compared to existing traffic in order to make NoC monitoring appealing. Furthermore, introduction of data on a system can change the system's behavior causing communication intrusiveness. Transport of runtime NoC performance monitor data over the NoC that is being monitored also introduces the risk of including this data in the measurements causing measurement intrusiveness.

Communication should be non-intrusive or low-intrusive and measurement intrusiveness must be removed to make runtime NoC performance monitoring usable. The runtime NoC performance monitoring project only investigates the latter.

The research question that is to be answered is stated as follows:
Is runtime NoC performance monitoring feasible and how can it be used?

1.2 Project environment

The project is part of DEMONS ([1] and [7]) and is performed at Philips Research. Scientists at Philips Research are developing a NoC architecture called Åthereal ([3] and [4]). This architecture is used as a test bench for runtime NoC performance monitoring. The Åthereal simulator [8] is used for experiments.

1.3 Objectives

These are the objectives that should be met in order to answer the previously stated research question:

1. **Show feasibility by design of a runtime NoC performance monitoring system.** Åthereal is used for this purpose. Besides showing feasibility, the design of the runtime NoC performance monitoring system for a real NoC helps us in theorizing about the costs for both the required communication and computation.

   (a) **Obtain overview of performance measures.** There are numerous NoC performance measures. An overview of NoC performance measures is of key importance for designing and understanding runtime NoC performance monitoring.

   (b) **Avoid measurement intrusiveness.** Because the runtime NoC performance monitoring system uses the NoC to transport monitor data, the danger exists that the measurements influence themselves. This so-called measurement intrusiveness must be solved to make the system usable.

   (c) **Give cost effective solutions.** As the main purpose of the runtime NoC performance monitoring system is that it is to be used for on-chip runtime measurements, traffic, area and energy must be as low as possible.

2. **Obtain realistic communication cost figures by implementing runtime NoC performance monitoring for a real NoC (Åthereal).** By implementing runtime NoC performance monitoring for the Åthereal simulator we can obtain realistic figures for communication costs (traffic, energy).

3. **Show usability of runtime NoC performance monitoring with realistic applications.**

1.4 Main contributions

During this project I have:

- Proved feasibility of runtime NoC performance monitoring

  - Designed and implemented a runtime NoC performance monitoring system based on the NoC monitoring service for the Åthereal NoC which supports the following performance measures:
1.5. Report organization

- Link throughput
- Link utilization
- Router utilization
- Network interface buffer utilization
- Router buffer utilization
- Connection latency

- Shown communication cost of the aforementioned performance measures both theoretically and experimentally

- Presented a framework for runtime NoC performance monitoring performance measures

- Presented techniques for removing measurement intrusiveness and applied one of these techniques in the Æthereal NoC runtime NoC performance monitoring implementation

- Proved usability of runtime NoC performance monitoring
  - Presented network resource monitoring for QoS management as an application of runtime NoC performance monitoring
  - Introduced a NoC congestion control scheme using model predictive control (MPC) as part of QoS management
  - Implemented this control scheme for a small setup for the Æthereal NoC

1.5 Report organization

The preamble of this document presents a list of frequently used terms and abbreviations. The second chapter describes the concepts of runtime NoC performance monitoring as well as related work. In the third chapter, all investigated performance measures are presented. The fourth chapter shows the design and implementation of the runtime NoC performance monitoring system for Æthereal for some of the measures from chapter three. Chapter five describes the conducted experiments and presents the results derived from these experiments. In the sixth chapter, NoC feedback control is presented as an application of runtime NoC performance monitoring. The results are compared with the expectations from chapter four. The most important conclusions and recommendations are presented in the final chapter. In appendix A, the verification of an SDF model is presented.
Chapter 2

Concepts and related work

This chapter describes, besides the concepts of runtime NoC performance monitoring also the basics of NoCs, the NoC monitoring service [1; 7] and Æthereal [3; 4]. Finally, this chapter discusses internet performance monitoring as related work.

2.1 Networks-on-Chip

2.1.1 Basic concept

A System-on-Chip (SoC) combines multiple IP components on a single chip. Buses can provide a high performance interconnect for SoCs; however, they are not scalable. For instance, when the number of components increases the bandwidth per component decreases. This makes buses unsuitable for SoCs with more than 10 bus master components [2]. NoCs are switched networks for SoC interconnect. As opposed to bus based interconnects they are scalable; bandwidth can be increased by increasing the number of switching components. NoC concepts are comparable to the concepts of other switched networks like local area networks and wide area networks (LAN and WAN). There are however eminent differences. Area and energy efficiency is crucial for embedded systems whereas for LANs and WANs these characteristics are far less important. Therefore, resources for NoCs are scarce compared to those for LANs and WANs.

A typical NoC consists of a switching network composed of multiple routers (R) and network interfaces (NI) which connect processing units (IP) to the switching network. Figure 2.1 shows a NoC with a 3x3 mesh topology. One NI can connect multiple IPs to the router network.

Examples of NoCs currently under development are Æthereal [3; 4], Nostrum [5] and Xpipes [6]. Æthereal is further discussed in the next section.

2.1.2 Æthereal

The Æthereal NoC [3; 4] consists of two components: routers [9] and network interfaces [10]. Routers can be connected with each other or with NIs by means of links. There are no topology constraints. Data is transported from one NI to another through the router network. Links, the physical means of transportation, are shared among different users by means of a time division multiple access scheme (TDMA).

Figure 3.1 shows the basic components of the Æthereal NoC. NIs offer transport layer communication services to IPs and hide the communication infrastructure from the IPs. By doing so, NIs separate computation from communication.
2.1. Networks-on-Chip

Æthereal provides backward compatibility with bus based protocols by using a transaction based model. For such a model, there are master IPs and slave IPs. Master IPs can start a transaction by issuing a request. This can for instance be a request for write (send data to slave) or a request for read (ask the slave to send data to master). The slave executes the transaction and optionally sends back data or an acknowledgment. Æthereal offers end-to-end (between IPs) transport layer services by means of connections which transport messages from one network interface port (NIP) to another.

A connection consists of one or multiple channels. The notion of (virtual) channels appears when TDMA slots are reserved consecutively along a path of links through the router network. Channels transport packets from one NIP to another (network layer communication service).

The current implementation of Æthereal always uses two channels per connection, one forward and one reverse channel. If the reverse channel is not used for transporting data, it is still there to transport end-to-end flow control. Connections have different properties such as guaranteed throughput (GT) or best effort (BE). For GT connections, time slots are reserved in a TDMA slot wheel to give guarantees for both latency and throughput. BE data is scheduled on non-reserved slots or unused slots, both latency and throughput depend on the availability of free slots.

The Æthereal NoC is a packet switched NoC that uses wormhole routing. For this technique, packets are further split into flits and routers do not have to wait for complete packet arrival but can immediately propagate a flit to the next router, significantly decreasing required router buffer sizes [11]. For Æthereal, the first flit of a packet contains a header with credits (end-to-end flow control), destination queue id and path information. The path that is followed by flits is predetermined at the sources (source routing). The current implementation of Æthereal has three words per flit and has two extra bits of sideband information per word (see [10]).

NIs have an input buffer (from IP to network) and an output buffer (from network to IP) for each connection. End-to-end flow control is implemented to ensure that buffers do not overflow. To do so, credits are added to packet headers and subsequently sent over a channel to the source NI. Input queuing is used at routers. As opposed to BE buffers, those for GT connections are very small (1 flit) due to the fact that data is guaranteed to propagate to the next router in the next flit clock (clock on which flits are propagated through the network). An extra wire is added (besides the link wires) to implement link level flow control, ensuring that router buffers do not overflow.

Figure 2.1: NoC with 3x3 mesh topology
Chapter 2. Concepts and related work

The following list summarizes Aethereal terminology used throughout this thesis:

**Intellectual property (IP)** IPs are entities that produce or consume data (e.g. processing unit or memory unit).

**Network interface (NI)** A NI is an interface between an IP and a NoC. It separates computation from communication. Packetization/depacketization from/to messages takes place at the NI. A NI receives messages from an IP and sends packets (split into flits) to the network and vice versa. Temporal speed differences between an IP and the network are resolved in the NI by buffering.

**Router (R)** A network of routers switches data from one NI to another. Aethereal routers use input queuing and wormhole routing.

**Link** Links are the physical means of communication for NoCs that connect NIs and routers.

**NI port (NIP)** NIPs are the ports through which IPs are connected to NIs.

**Router port** Router ports are the communication exit and entrances for the modules inside the NoC.

**Buffer** Buffers are necessary to cope with temporal speed differences. NI input buffers are used to store data that is generated by IPs whereas NI output buffers store data that is generated by the network. A crediting mechanism (flow-control) is used to ensure that sent data does not exceed output buffer space. Routers are equipped with input buffers. The input buffers for GT connections are very small because data is guaranteed to be forwarded. BE buffers store BE data when there is no room for them in the next router or the connected NI.

**Connection** Messages are communicated between NIPs by means of connections. Communication can go from master to slave through a forward channel and from slave to master through a reverse channel. For Aethereal, a connection at least consists of a forward and a reverse channel. If the reverse channel is not used for data it is still required for end-to-end flow control.

**Channel** Transport of packets between NIPs takes place via (virtual) channels. The notion of virtual channels exists when consecutive slots are reserved for the links that form the path between two NIPs.

### 2.2 NoC monitoring service

An event based NoC monitoring service has been proposed in the DEMONS project [1; 7]. This service provides runtime observability of NoC behavior to enable for instance communication-centric debugging.

The service allows the deployment of an arbitrary number of probes to be connected to NoC components. Currently, probes can be connected to routers and NIs from where all NoC components can be observed. Each probe consists of a sniffer, an event generator (EG) and a monitoring service network interface (MNI) as depicted in Figure 2.2. Sniffers are attached to routers and NIs allowing access to all internals.

In the event generator, events are generated based on the sniffed data. This event is sent to a gathering module called a monitoring service access point (MSA) which analyses the event or sends it off-chip.
2.2. NoC monitoring service

In general an event is a happening of interest; the event format for the monitoring service is defined as:

\[
event = (identifier, timestamp, producer, attributes)
\]

The identifier relates to the class of an event, the timestamp gives the time that the event occurred, the producer relates to the source entity of the event and the attributes contain event information.

Communication between probes and MSAs can be handled by the monitored NoC or by a dedicated interconnect. Currently, GT connections are reserved in the monitored interconnect to transport monitor data. All monitor data can be transported to a single MSA (centralized) or to multiple MSAs (distributed). Figure 2.3 shows a NoC where multiple probes are connected to some of the routers in a 4x4 mesh and communicate with a single MSA (centralized) by means of a connection of the monitored NoC. One of these monitor connections is depicted with arrows.

![Figure 2.3: NoC with several probes and a single MSA, one of the monitor connections is indicated with arrows](image)

Active monitoring systems insert test data in a monitored system and observes the reaction of the system to those test patterns. Passive monitoring systems, such as the proposed monitoring service, observe existing data and is in this respect non-intrusive.

Intrusiveness is an important issue for monitoring in general. Two types of intrusiveness have to be dealt with for runtime NoC performance monitoring:
communication intrusiveness and measurement intrusiveness. This is further explained in Section 2.3.3

2.3 Runtime NoC performance monitoring

2.3.1 Introduction

Network performance monitoring is commonly used for off-chip networks to trace performance problems such as high latencies and high congestion and to benchmark system performance. Runtime NoC performance monitoring provides runtime observability of NoC performance as part of the NoC monitoring service.

Measurements are performed at probes (Figure 2.2) to obtain performance measures such as connection throughput and link utilization. A performance measure is specified by a performance measure target (e.g. connection and link) and the observed performance metric (e.g. throughput and utilization). Performance measures are discussed in Chapter 3.

Performance events are generated at probes and transported over the NoC to one or multiple MSAs from which they can be transported off-chip or used locally (Figure 2.3).

Figure 2.4 shows the basic steps for runtime NoC performance monitoring.

![Diagram of runtime NoC performance monitoring](image)

The first step is measurement at a probe. Measurement data can immediately be translated to the desired performance measure or this can be done at a MSA. An event is generated which includes the measurement result. This event is transported from a probe to a MSA over an interconnect (in our case the monitored NoC). Finally the data is used at a MSA or sent off-chip. The steps are further explained in the remainder of this section.

2.3.2 Measurements and event generation

Actual measurements are required to obtain performance measures. For instance, if link throughput is the performance measure, the number of words or flits that passed the link during a period of time is the required measurement. There are often multiple places where a specific measurement can be taken. This is discussed in Chapter 4. The time at which measurements should be taken depends on where it is taken. If we assume a synchronous system (which is often the case) then measurements can be synchronized with the appropriate clock. Measurements can be accumulated and averaged over a period of time $\Delta T$. For the remainder
2.3. Runtime NoC performance monitoring

of this document, $\Delta T$ is measured in flit clocks unless stated otherwise. The flit clock time period for the $\&$theral NoC equals 6 ns.

Performance measures are always a function of measurement data. Consider link utilization which is defined as the actual throughput divided by the potential throughput of a link (see Chapter 3 for a more detailed discussion on link utilization). The number of flits over a link during a period of time is required to compute the actual throughput. The potential throughput is, in the case of link throughput, a fixed value (currently 2GB/link/sec or 0.17 Gflits/link/sec for $\&$theral ([3] and [4])). The link utilization performance measure is now computed by dividing the two throughputs. This computation can be performed at the probe, at the MSA or even off-chip.

$$\text{link utilization} = \frac{\text{actual throughput}}{\text{potential throughput}}$$

The generic event format for the monitoring service contains an identifier, a timestamp, a producer and attributes (see Section 2.2). The size of the generated event has significant influence on the generated load. In Chapter 4, some optimizations for this event format are shown for runtime NoC performance monitoring for the $\&$theral NoC.

2.3.3 Transport of events from probe to MSA

For the current monitoring service, monitor data is transported from probes to MSAs by means of GT connections which are specially reserved for the monitoring service. This introduces the risk of communication and measurement intrusive­ness. Both can change the measurements and have to be dealt with in order to make runtime NoC performance monitoring usable.

Communication intrusiveness occurs when a monitoring service introduces data on the network. The insertion of data can change the communication behavior and therefore introduces the risk of hiding problems or even creating new problems. For the proposed monitoring service, the monitoring itself is passive and is in that respect non-intrusive. But the monitor data needs to be transported from a probe to an MSA. If this communication takes place over the monitored NoC, as is the case, the risk of communication intrusiveness exists. This problem is dealt with by the NoC monitoring service.

Measurement intrusiveness occurs when measurements include their own generated data (e.g. measure link throughput and include the load generated by the probe that measures link throughput). This can only happen when the monitored interconnect is used to transport monitor data. This problem should be solved for runtime NoC performance monitoring. Two solutions are presented in Section 4.3.

2.3.4 Interpretation of data

Data generated by runtime NoC performance monitoring is used at an MSA or sent off-chip. Runtime NoC performance monitoring typically results in a large set of performance measure values $X$. The average over all samples of $X$, $\bar{x}$ and the sample variance, $s^2$ (or alternatively the standard deviation of $X$, $s$) give a good estimate for the expected value but can hide crucial performance information. Figure 2.5 shows the probability distribution of an imaginary performance measurement where the result has a normal distribution. The presented average and standard deviation give information about the ran experiment but do not tell us what the behavior was at a specific time.
Chapter 2. Concepts and related work

Measurements are typically accumulated at probes during a period of time $\Delta T$ and subsequently averaged over $\Delta T$. When chosen properly, the averages are independent of each other and can therefore be used as so-called sub-run averages [12]. Sample average and standard deviation of the sub-runs give insight in the local behavior and thereby alleviate the aforementioned problem. Furthermore, sub-runs can be used to calculate confidence intervals for the sample average, which provides insight in the reliability of the measured average.

The appropriate sample period for the sub-run averages depends on the characteristics of the observed data. Very small time periods ensure that nothing is missed but generate large amounts of data whereas very high time periods generate low amounts of data but introduce the risk of averaging out critical performance behavior.

To illustrate the importance of appropriate $\Delta T$ values consider the measurement depicted in Figure 2.6. The solid line depicts the actual value for a certain performance measure (e.g. link throughput) with a sine shaped disturbance on top of it. The shown $\Delta T^*$ value ($\delta T_1$ in Figure 2.6) averages out the sine wave disturbance making it impossible to observe this behavior. Good values of $\Delta T$ are chosen based on the knowledge of (statistical) properties of the observed measure.

2.3.5 Applications for runtime NoC performance monitoring

NoC resource monitoring for QoS management

The computational demands of modern multimedia applications require platforms with multiple processing units connected by for instance NoCs. QoS is crucial for these systems where the behavior is unpredictable at design time.

QoS involves management of both network and computational resources based on application quality necessities, resource requirements and available resources. In the best case, resource requirements and availability can be predicted accurately and a QoS manager can act based on these predictions. In [13] for instance, a QoS system is presented that predicts execution time of an application at runtime while taking into account data dependencies. But such predictions depend on severe knowledge of the application and the data that the application
2.4 Runtime performance monitoring for the internet

![Graph showing performance measurement with ΔT chosen too high to observe sine wave shaped disturbance](image)

Figure 2.6: Performance measurement with ΔT chosen too high to observe sine wave shaped disturbance

produces. Resource monitoring fills the gap when processes are not predictable. Runtime NoC performance monitoring can be used to monitor NoC resources. In Chapter 6.1 we present an example where NoC utilization is monitored and controlled by means of feedback control. The proposed control method makes control decisions based on predictions and resource monitoring and therefore fits well within the QoS concept.

Communication-centric debugging

Communication-centric debugging is one of the main drivers for the DEMONS project. NoC performance problems can be regarded as bugs, can be root causes of bugs or can hide bugs. Runtime NoC performance monitoring can be used to detect performance problems and to trace these problems back to their root causes by observing the performance of the surroundings of the observed problem.

2.4 Runtime performance monitoring for the internet

Internet is an example of a network where runtime performance monitoring is frequently used. Although experiments with internet like networks and with the protocols used for the internet can be conducted in an artificial environment, actual internet performance monitoring, just like runtime NoC performance monitoring, always takes place while the system is actually running.

Performance monitoring is for instance used by internet service providers to identify congestion problems in their subnet. Internet performance monitoring is also available for home users. Take for example the Microsoft Windows ping tool which provides latency information of the connection from a user to a specified ip address. In [14], a framework is presented for performance metrics for the internet in an attempt to structure internet performance monitoring.

Internet and NoC performance monitoring are comparable in the sense that in both cases performance monitoring of a switched network is performed. Therefore they share some performance measures such as router utilization. But this is where the similarities of internet and NoCs stops. Available resources for NoCs are much scarcer than for the internet and energy efficiency is much less important.
for the internet. The differences become even greater if we compare Æthereal with the internet. Æthereal is connection based creating virtual channels from sender to receiver whereas the internet uses pure packet switching where each packet can follow a different path from sender to receiver.
Chapter 3

Performance measures

A NoC is a complex system for which numerous interesting performance measures exist. To make a runtime NoC monitoring system usable, it should at least support the most common performance measures.

Measurements can be taken at different NoC entities (e.g. at a link or at a connection). We call these entities performance measure targets. Different performance metrics, such as throughput and latency, can be observed at these performance measure targets.

Performance metrics and performance measure targets are presented as part of a framework that is based on the one presented in [15].

3.1 Framework

3.1.1 Introduction

A performance measure is specified by two basic elements: a performance metric (e.g. throughput and latency) that describes what to measure and a performance measure target (e.g. link and connection) that describes where to measure.

The framework, presented in this section, specifies both performance measure targets and performance metrics for runtime NoC performance monitoring. The presented framework is based on the one proposed in [15] for NoC performance analysis and benchmarking.

The framework presented in [15] is targeted at performance analysis and benchmarking in general whereas the one presented here is targeted at on-chip NoC performance monitoring at runtime. Therefore, the frameworks slightly differ in supported performance measures. The framework for runtime NoC performance monitoring is targeted at NoC internal as well as NoC external behavior whereas the other framework focuses on NoC external behavior.

We propose a framework for runtime NoC performance monitoring that consists of the performance metrics and performance measure targets shown in table 3.1.

3.1.2 Performance measure targets

In Figure 3.1 all performance measure targets investigated for runtime NoC performance monitoring are depicted. For simplicity, the picture only shows one router; typical NoCs consist of many routers. The performance measure targets are based on the Æthereal NoC [3; 4] but most of them are generic for NoCs. See Sections 2.1.1 and 2.1.2 for further details about the presented targets.
Chapter 3. Performance measures

(a) Performance metrics
- Throughput
- Utilization
- Latency
- Contention
- Jitter

(b) Performance measure targets
- Network interface (NI)
- Router (R)
- Link
- Port
- Buffer
- Connection (messages)
- Channel (packets, flow-control)
- Switching network

Table 3.1: Performance metrics (a) and performance measure targets (b)

Figure 3.1: Performance measure targets framework

Performance measure targets are not necessarily physical entities. Connections and channels are for instance conceptual targets that only exist while the system is running.

Figure 3.2 shows the relationships between the targets in UML notation.

3.1.3 Performance measures per performance metric

Introduction

We first give an informal introduction to a performance metric and then formally define the metric. Then, the targets for which the performance metric is interesting are discussed.

Throughput

Throughput is, next to latency, the most commonly used performance metric. It expresses the actual data that goes through a resource during a period of time ($\Delta T$) and should not be confused with bandwidth, which is the amount of data that could go through a resource.

Definition 1 (Throughput $\frac{(\text{bits/bytes/words/flits})}{\text{second}}$). In general, throughput is defined as the quantity of goods that go through a process during a specified period of time. It is common for networks to look at the number of bits/bytes/words that pass through a certain port during a specified time.

Of interest are the throughput of individual connections, their constituting channels and at the ports and links. Enabling these performance measures also enables the analysis of throughput for the complete system.

\footnote{The jitter metric differs from all other metrics in the presented table in the sense that it is a metric for a metric (e.g. latency jitter).}
3.1. Framework

Utilization

Performance results are always relative to the monitored subject. Thirty kilometers an hour is considered slow for a car but fast for a bicycle. Utilization puts things in perspective by expressing actual observed performance versus potential performance.

Definition 2 (Utilization (%)). *Utilization is the percentage that a resource is actually used out of the total potential of the resource.*

\[
\text{Utilization} = \frac{\text{actual use}}{\text{potential use}}
\]

Utilization for connections, channels, ports and links is closely related to throughput. For these measures we look at actual throughput versus available bandwidth. Buffer utilization expresses the filling of a buffer versus its size.

If utilization is measured for all input or output ports/links of a router we have router utilization. In the same way, network utilization can be obtained by observing all link utilizations or equivalent measures.

Latency

Applications expect network transactions to be performed in a limited and constrained amount of time. Especially for real-time or multimedia applications, transfer times should be bounded.

Definition 3 (Latency (seconds)). *Latency is the time that it takes for a certain unit of data (e.g. transaction, message, packet, flit, word) to get from one point to another. Communication from one point to another can take place over multiple paths. If this is the case, latency can be defined as the maximum latency of all paths or as the average of the latency over all paths.*

Latency is introduced in many points in a NoC. Figure 3.3 shows the latency measures.
In the presented framework in Figure 3.1, latencies can be measured for connections and for the channels that constitute these connections. These latencies are further split into latencies at NIs and latencies caused by the switching network as depicted in Figure 3.3. Latency can also be measured for transactions (i.e. time between transaction being issued and transaction completed). This is not part of this framework because it includes performance measures that can not be taken inside the NoC.

Some of the performance measure targets have fixed latencies and it therefore makes no sense to measure them at runtime. Examples of these fixed latencies are port latency and link latency.

In the following the aforementioned latency measures are discussed.

**Channel latency** \( (T_{\text{channel}}) \) is the latency of an individual channel. Channels transport packets from one NIP to another. Channel latency is therefore defined as the difference between the time at which a packet is created and the time at which the packet is completely delivered at the destination NI. Packet headers contain end-to-end flow control information and therefore latency of end-to-end flow control can be observed by observing channel latency.

**Connection latency** \( (T_{\text{connection}}) \) Connections transport messages from one NIP to another. Connection latency is the difference between the time at which a message is offered to the sending NI and the time at which a message is completely delivered at its destination NI. A connection can transport messages in a unidirectional (e.g. master to slave) or bidirectional (e.g. master sends data and asks slave for acknowledgment) way. Connections can consist of multiple channels (currently always two channels per connection). We always look at the latency of a single point to point connection. The key difference between channel latency is that connection latency also includes the waiting times at the NI queues.

**Network interface latency** \( (T_{NI}) \) is the time spent by a message or a transaction in a network interface. A connection that goes from master IP to slave IP has two network interface latencies, one at the sending side and one at the receiving side.

**Router latency** \( (T_{R}) \) is the time that a unit of data (message, packet or flit) spends in the router.

**Switching network latency** \( (T_{\text{switching network}}) \) is the time that a unit of data (message, packet or flit) spends in the switching network. The switching network latency can be derived from router latencies.
3.2. Summary of performance measures

Jitter

For applications, constant data flow can be very important. Nervous behavior (or jittering) of latency or throughput on a connection can be devastating for the application performance.

**Definition 4** (Jitter (bytes/second, seconds, %, #)). *Jitter is an unexpected and unwanted deviation of a certain measure. Common jitter measures are throughput and latency jitter.*

Although jitter can be defined for numerous metrics, latency jitter is the most commonly used metric. [15] presents three interesting interpretations for latency jitter. Namely:

- **Latency jitter definition 1** Jitter can be seen as the latency variation between two consecutive messages coming from the same source and using the same connection.
- **Latency jitter definition 2** Jitter can be seen as the difference between inter-arrival times of messages received from the same connection and the inter-departure times of messages put on that connection.
- **Latency jitter definition 3** is the difference between actual message latency and average message latency of all packets received over the same connection.

The first jitter latency definition is most commonly used and shows the latency difference between two messages consecutively sent over the same connection; time shifts in departures and arrivals of messages are not expressed by this latency metric. These shifts are exposed by the second latency metric. The third jitter metric exposes abnormal behavior by comparing message latencies with the average message latency.

Contention

In a NoC, links are shared among connections. When two connections request the same link at the same time they content.

**Definition 5** (Contention (#)). *Two units of data are said to content if they want to use same resource at the same time.*

Contestation can take place at two places in a NoC. The first place is the NI where a single link is shared among all connections of the NI. The second place is a router output port where multiple input data use the same output. Note that for Etherereal, contention can only occur for BE traffic with either BE traffic or GT traffic. Two streams of GT traffic can share the same resource but never want to use it at the same time.

3.2 Summary of performance measures

The performance measures derived in the previous sections from the performance metrics and performance measure targets as presented in table 3.1 have different levels of abstraction. The Open System Interconnect (or OSI) model gives an abstraction model for communication systems. Three layers of the OSI model are of interest for the presented framework. The transport layer which deals with transactions and is independent of the network. The network layer that deals with the actual communication of data over the network. And finally, the data link layer which deals with error correction/detection and flow-control. Because we assume an error-less system, only the flow-control part of the data link layer is used for performance monitoring.
In the previous section we saw that not all metric target combinations result in interesting performance measures. Some of the performance measures are fixed (e.g. link latency), some have no meaning (e.g. router contention) and some can not occur (connection contention can for instance not occur, at least not from the NoC point of view). Leaving out these performance measures results in Table 3.2. Jitter is a measure on performance measures and can be measured for all performance measures. Out of the 64 possible combinations from Table 3.1 only 28 are left.

Links connect ports, therefore, most of the measures taken at links are similar to the measure taken at the connected port. NI latency is part of both the transport and the network layer. This is due to the fact that a NI is an interface between the transport and the network layer.

The performance measures presented in Table 3.3 are designed and implemented for the Æthereal NoC. The list is the result of an elimination process where importance for a runtime NoC performance monitoring system (does a SoC engineer expect this performance measure) and the complexity (how much effort is involved in designing and implementing this measure) were traded off. The design and implementation of the performance measures is described in Chapter 4.

<table>
<thead>
<tr>
<th>Performance measure</th>
<th>OSI layer</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Target</strong></td>
<td><strong>Transport layer</strong></td>
</tr>
<tr>
<td>Network interface</td>
<td>Latency</td>
</tr>
<tr>
<td>Router</td>
<td>Utilization</td>
</tr>
<tr>
<td>Latency</td>
<td>x</td>
</tr>
<tr>
<td>Link/port</td>
<td>Throughput</td>
</tr>
<tr>
<td>Utilization</td>
<td>x</td>
</tr>
<tr>
<td>Contention</td>
<td>x</td>
</tr>
<tr>
<td>Buffer</td>
<td>Utilization</td>
</tr>
<tr>
<td>Connection</td>
<td>Throughput</td>
</tr>
<tr>
<td>Utilization</td>
<td>x</td>
</tr>
<tr>
<td>Latency</td>
<td>x</td>
</tr>
<tr>
<td>Channel</td>
<td>Throughput</td>
</tr>
<tr>
<td>Utilization</td>
<td>x</td>
</tr>
<tr>
<td>Latency</td>
<td>x</td>
</tr>
<tr>
<td>Switching network</td>
<td>Latency</td>
</tr>
</tbody>
</table>

Table 3.2: Performance measure overview

Table 3.3: Performance measures designed and implemented for the Æthereal NoC

\[2\text{Jitter is not displayed in this table, it can be measured for all performance measures in the table.}\]
3.3 Conclusions

In this chapter, we specified performance measures by presenting a framework that consists of performance measure targets and performance metrics. The resulting performance measures are shown in Table 3.2. Some of the presented performance measures are designed and implemented for the Æthereal NoC (see Chapter 4 and Table 3.3).
Chapter 4

Design and implementation

In this chapter, the feasibility of runtime NoC performance monitoring is shown by the design and implementation of some of the performance measures (see Chapter 3) for the \textit{A}thereal NoC [3; 4]. Link utilization, link throughput, router utilization, buffer utilization and connection latency are investigated (see Table 3.3). For each of these performance measures we identify the required measurement (e.g. number of flits at a link during a period of time to calculate link throughput), measurement approaches, an event format, communication costs and possible optimizations.

Measurement intrusiveness can have a devastating effect on runtime NoC performance monitoring (see Section 2.2). In this chapter we discuss two methods to solve the measurement intrusiveness problem.

4.1 Measurements

4.1.1 Link utilization and link throughput

\textit{A}thereal links are always bi-directional. If we refer to link measures we always talk about one of the two directions. Both directions are supported by our implementation. Link utilization is defined as the actual throughput on a link divided by the potential throughput of that link. As explained in Section 2.1.2, data is transported over the network in flits. In the current implementation, flits are composed of three words with two bits of sideband information added to each word (see Figure 4.1 and [3]).

<table>
<thead>
<tr>
<th>type</th>
<th>packet header/payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>payload</td>
</tr>
<tr>
<td>etp</td>
<td>payload</td>
</tr>
</tbody>
</table>

Figure 4.1: Flit format for \textit{A}thereal

The first bit of the first two sideband bits indicates the type of flit (BE or GT). The first word of a flit contains either a header or a payload word. This is identified by the second of the first two sideband bits. The number of payload words in the flit, ranging from zero to three, is indicated by the second two sideband bits. From the third two sideband bits, only the first one is used. It is used to state whether a flit is the last one of a packet.
4.1. Measurements

Flits are always entirely scheduled to links, in disregard of the number of words that actually contain payload. Therefore, for link utilization, link throughput is measured by counting the number of flits that cross a link during a period of time and dividing the result by this time period. Link throughput can be measured at different granularities.

We discuss a method for measuring at the level of words and a method for measuring at the level of flits.

The brute force approach is to sniff link wires and use an OR port to see whether one of them is active. This approach is based on the observation that the first word of a flit either contains a header or payload. Whether the first word of a flit is a header can be observed at the second of the first two sideband bits. This requires one wire to be sniffed, if it is active, it is counted as a flit. There are two possibilities if this link is not active; a flit is scheduled without header or there is no flit scheduled. The size sideband bits are observed in order to determine the situation. This requires both sideband wires to be sniffed. This method is suitable for measuring link throughput by counting the words that pass the link. It can also be used to measure the number of flits that pass a link.

Another method is to observe scheduler internal signals to check for which output ports and hence to which link there is data to be scheduled. As stated before, scheduling is performed at the granularity of flits. It is not possible to determine the number of words that are scheduled on a link. This method is therefore most suitable for counting the number of flits that pass link.

4.1.2 Router utilization

Router utilization can be measured by accumulating all input link utilizations for all links that are connected to the input or for all links that are connected to the output of the router. Router utilization is obtained by dividing the accumulation result by the number of observed links. The same measurement approaches can be used as for link utilization (see previous subsection).

4.1.3 Buffer utilization

Buffer utilization is defined as the average buffer filling during a period of time divided by the potential buffer filling (buffer size) during that period of time. We quantify buffer filling by counting the number of words in a buffer during a period of time $\Delta T$ and dividing the result by $\Delta T$.

For Athereal, the buffers used in NIs are able to give filling information because buffer filling information is used for scheduling purposes. The signals between a NI buffer and the scheduling module can be sniffed by a probe. Router buffers are not capable of providing filling information. Therefore, a mechanism is required that keeps track of the buffer filling. For this purpose, write and read valid signals can be sniffed which indicate whether data is actually read or written to increase or decrease a counter (initialized to buffer size). Or alternatively, the buffers can be extended to give buffer filling information.

4.1.4 Connection latency

Athereal connections are able to communicate messages in two directions. For connection measures we always look at one of the two directions.

As observed in Section 3.1.3, connection latency is defined as the difference between the time that the first word of a message enters the input queue and the last word of that message leaving the output queue. Both these times must be measured to compute connection latency (see Figure 4.2).
Our method uses two probes (if we assume that a connection transports messages from a master to a slave located at different NIs). The first probe is attached to the sending NI, sniffs the input queue and creates a timestamped event when it observes that a new message is offered. The second probe is attached to the receiving NI and sends a timestamped event when it observes the last word of a message leaving the queue. Connection latency is calculated at the MSA by subtracting the timestamps. It is crucial that subtracted timestamps relate to the same message. Messages arrive at destinations in the same order as they were sent (in order arrival) for Ethereal making it easy to ensure that timestamps relate to the same message. An identifier must be added to messages if a routing scheme does not guarantee in order arrival (e.g., hot potato routing).

I assume that there is a synchronized timestamping mechanism which uses a counter of a fixed size at both probes. The timestamping mechanism should support timestamps high enough to cope with the maximum latency that can occur for messages. Latency can then be obtained by calculating the relative difference between the generated timestamps, if the probe at the producing side generates a timestamp of value \( n \) and the consuming probe generates a timestamp of value \( m \), the latency is defined as \( L_{\text{message}} = (n - m) \mod (\text{timestamp\_counter\_size}) \). The latency results are not reliable if the latency exceeds \( \text{timestamp\_counter\_size} \). If it is not possible to guarantee message latency upper bounds, a warning should be generated when \( \text{timestamp\_counter\_size} \) is exceeded. This is solved for the monitoring service, a synchronization event is generated when timestamp counters reach their maximum value ([7]).

There are several approaches to minimize generated load. First of all, the timestamp size should be as small as possible. As previously suggested, a warning can be generated when timestamp size is exceeded. Generated load can be decreased significantly by looking at the latency of \( n \) messages at a time. An event is generated when the first word of \( n \) subsequent messages on a connection enters the input queue and when the last word of \( n \) subsequent messages on a connection has left the output queue. This method requires bigger timestamps but reduces the rate at which events are generated with a factor \( n \). The effect of the decreasing rate is much bigger than the effect of the increasing timestamp size. Consider going from \( n = 1 \) with a timestamp size of 1 byte (supported latency of \( 255 \cdot 6 = 1530 \) ns) to \( n = 8 \) and we want to support the same latency per individual message. The latency that has to be supported for \( n = 8 \) messages equals \( 8 \cdot 1530 \) ns; two bytes are required to represent this. Thus, timestamp size doubles but is sent eighth times less often, decreasing the load with a factor 4. In this way, communication costs can be traded for measure accuracy.
4.2 Event generation

4.2.1 Event format optimization

The generic event format for the monitoring service consists of an identifier, a timestamp, producer and attributes (see Section 2.2). An identifier is only necessary when different types of events are generated during the same run from the same probe. Otherwise it can be left out.

Except for connection latency, all events generated by runtime NoC performance monitoring have periodic behavior. The time difference between each measurement equals $\Delta T$. If the probe starts generating events at time 0 (when the system is started) the timestamps can easily be reconstructed at the MSA. If this is not the case, the probe can send a reference timestamp with the first event from which all further timestamps can be reconstructed. For our implementation of the monitoring service, the MSA is aware from which connection it receives data. The connection directly relates to producer, the producer field can be left out for all performance measures.

4.2.2 Link utilization and link throughput

Event format

The event format supports multiple link utilization/throughput events at a single probe (e.g. monitoring all link utilizations at a single router). The presented format includes an identifier field. This is not always necessary as explained in Section 4.2.1. The event format is:

$$\text{event} = (\text{identifier}, \#\text{flits.during.} \Delta T_1, \ldots, \#\text{flits.during.} \Delta T_n)$$

where $n$ denotes the $n$-th monitored link.

Communication cost analysis

The number of bits that are required to encode the number of flits accumulated during $\Delta T$ equals $\log_2(\text{max.}\#\text{flits.during.} \Delta T)$ where $\text{max.}\#\text{flits.during.} \Delta T$ is equal to $\Delta T + 1$ (assuming that the unity of $\Delta T$ is flit clocks). E.g. during 255 flit clocks we can count 0 to 255 flits, this requires $\log_2(256)$ bits. If events are immediately sent to the MSA, this implies a load of:

$$\frac{\log_2(\text{identifier.size}) + \log_2(\Delta T + 1)}{\Delta T \cdot 6 \cdot 10^{-3}} \text{ (Mbits/s)}$$

where the factor $10^{-3}$ is used instead of $10^{-9}$ to directly obtain $\text{Mbits/s}$ rather than $\text{bits/s}$. If an identifier of 1 byte is used (support for 256 different events), and $\Delta T$ equals 100 flit clocks the generated load becomes:

$$\frac{8 + \log_2(101)}{100 \cdot 6 \cdot 10^{-3}} \approx 24,43(\text{Mbits/s}) \approx 3,05(\text{MB/s})$$

This however assumes that we are able to send 6,7 bits which is obviously impossible. In fact, it is unlikely that an arbitrary number of bits can be send through the network efficiently. For instance, when the NoC is designed for sending bytes, there is little difference between sending 7 and 8 bits of data. Real networks often transport multiple bytes of data.

Here we make the comparison between sending in bits and sending in bytes.

If we round the bits required for $\text{max.}\#\text{flits.during.} \Delta T$ to complete bytes the load becomes:
1 + \left\lfloor \log_2(101) / 8 \right\rfloor \approx 3,33 (MB/s)

Currently, Etherereal has a link capacity of 2 GB/s and a communication cost (network layer costs) of approximately 30%, leaving 1.4 GB/s for payload. The communication cost for this performance measure is 0.24% compared to this link capacity. Note that generated load decreases when \( \Delta T \) increases and that the example value of 100 flit clocks is equal to a time period of 600 ns which is small.

The load difference between rounding to bits and bytes depends on \( \Delta T \). If \( \Delta T \) is chosen close to a number that can be represented by one or multiple bytes the difference becomes less. Figure 4.3 shows the generated load when rounding to bytes and when rounding to bits and with and without identifier as a function of \( \Delta T \). The effect of rounding to bytes is clearly shown. The bit rounding plots show multiple small hops each time an extra bit is required (15 to 16, 31 to 32, etc) whereas the hops for the byte rounding plots are much bigger and occur less often (255 to 256, 65535 to 65536, etc).

![Figure 4.3: Generated load for single link utilization/throughput measure as a function of \( \Delta T \) when rounding to bits and to bytes and with and without identifier](image)

Identifier has a big influence on the generated load, especially for low \( \Delta T \) values. It is therefore profitable to remove this field if there are no other event types in the monitored NoC. The load shows logarithmic behavior. Very low \( \Delta T \) values generate a high load. The highest load is generated when \( \Delta T = 1 \); this results in a load of 0.33 GB/s which is 24% of the link capacity. Such low \( \Delta T \) values should be avoided. This holds for all performance measures with periodic measurements.

### 4.2.3 Router utilization

**Event format**

Router utilization is measured by accumulating the number of flits that pass all links connected to the outputs or to the inputs of a router.

\[
\text{event} = (\text{identifier, \#flits during } \Delta T)
\]

**Communication cost analysis**

The cost for router utilization can be described by:
4.2. Event generation

\[ \frac{\log_2(\text{identifier\_size}) + \log_2(\text{max\_\#flits\_during\_}\Delta T)}{\Delta T \cdot 6 \cdot 10^{-3}} \text{(Mbits/s)} \]

In this case, max\_\#flits\_during\_\Delta T is equal to the multiplication of arity a and \Delta T plus one (a \cdot \Delta T + 1). The only difference with link utilization/throughput is the factor a. Consider a router of arity 4 and a \Delta T value of 100 flit clocks. The generated load then becomes:

\[ \frac{8 + \log_2(401)}{100 \cdot 6 \cdot 10^{-3}} \approx 27.74 \text{(Mbits/s)} \approx 3.47 \text{(MB/s)} \]

Rounding to bytes results in a load of:

\[ \frac{1 + \lceil(\log_2(401)/8) \rceil}{100 \cdot 6 \cdot 10^{-3}} \approx 5 \text{(MB/s)} \]

This is 0.36% compared to the link capacity of 1.4 GB/s. Figure 4.4 shows the generated load when rounding to bits and bytes as a function of \Delta T for different arities.

![Generated load for router utilization measure as a function of \Delta T when rounding to bits and to bytes for different arities](image)

The graph shows a small difference in load for different arity values when rounding to bits. Rounding to bytes for arity 4 shows a similar peak as observed for rounding to bytes for link utilization. The peak is located at \Delta T = 63 where max\_\#flits\_during\_\Delta T = 253.

4.2.4 Buffer utilization

Event format

Utilization can be monitored for a single buffer or for multiple buffers at a single probe. The event format when monitoring n buffers is:

\[ \text{event} = (\text{identifier, buffer\_filling\_during\_}\Delta T_1, ..., \text{buffer\_filling\_during\_}\Delta T_n) \]
Chapter 4. Design and implementation

Communication cost analysis

The number of bits that are required to encode the buffer filling accumulated during $\Delta T$ equals $\log_2((\text{buffer size} \cdot \Delta T) + 1)$, this implies a load of:

$$\frac{\log_2(\text{identifier size}) + \log_2((\text{buffer size} \cdot \Delta T) + 1)}{\Delta T} (\text{Mbits/s})$$

E.g. if 8 bits are used for timestamping, 8 bits for identifier (support for 256 different events), buffer size equals 40 (words) and $\Delta T$ equals 100 flit clocks the generated load becomes:

$$\frac{8 + \log_2((40 \cdot 100) + 1)}{100 \cdot 6 \cdot 10^{-3}} \approx 33,33 (\text{Mbits/s}) \approx (4,17 \text{(MB/s)})$$

Rounding to full bytes results in:

$$1 + \left[ \frac{\log_2((40 \cdot 100) + 1)/8}{100 \cdot 6 \cdot 10^{-3}} \right] \approx 5 (\text{MB/s})$$

The cost for this example is 5% compared to the link capacity of 1,4 GB/s. Figure 4.5 shows the generated load when rounding to bytes and when rounding to bits with and without identifier.

As opposed to the previously investigated performance measures, there is no peak in the generated load for the shown range. The reason for this is that within the displayed range there is no crossing from one byte representation to two byte representation.

4.2.5 Connection latency

Event format

Connection latency requires a timestamp as measurement data. A message identifier field is required when more than one connection latency is monitored by using the same probe. The event format is:

$$\text{event} = (\text{identifier, timestamp})$$

Figure 4.5: Generated load for buffer utilization measure as a function of $\Delta T$ rounding to bits, rounding to bytes, with and without identifier

As opposed to the previously investigated performance measures, there is no peak in the generated load for the shown range. The reason for this is that within the displayed range there is no crossing from one byte representation to two byte representation.
Communication cost analysis

The connection latency measure requires two connections, one for the probe at the master side and one for the probe at the slave side. As opposed to other performance measures, connection latency does not generate events periodically with time period $\Delta T$. The generated load for each of the two probes when monitoring connection $C$ depends on the message size (in bytes), $load_C$ (in B/s) and timestamp size (in bytes):

$$\frac{load_C}{message\_size} \cdot timestamp\_size(B/s)$$

The generated load for both the probes differs a factor $\frac{timestamp\_size}{message\_size}$ from the load on connection $C$ ($load_C$).

For instance, monitoring a connection which transports traffic with a message size of 32 bytes (8 words), a load of 200 MB/s and timestamp size equals 2 bytes (supporting latencies up to $(2^{16} \cdot 6\text{ns}) = 65536 \cdot 6\text{ns} \approx 0,4\mu\text{s}$) generates a load of 12,5 MB/s for each of the two connections resulting in a cost of 1,79% compared to link capacity. Figure 4.6 shows generated load as function of $load_C$ with a message size of 32 bytes for a timestamp size of 1 and of 2 bytes.

![Figure 4.6: Generated load for connection latency measure as function of load_C for a message size of 32 bytes and a timestamp size of 1 and of 2 bytes](image)

The presented load is generated twice per connection latency measure. This is the most expensive performance measure in this investigation. It is clearly profitable to keep timestamp size as small as possible. Therefore, the connection latency measurement mechanism should work good under normal conditions and should support exceptionally high latencies by generating a warning message.

4.3 Measurement non-intrusiveness

4.3.1 Introduction

Intrusiveness (see Section 2.2) is caused by the fact that the monitored NoC is also used to transport monitor data. Measurement intrusiveness is the effect that monitor data streams are included in measurements and should not be confused with communication intrusiveness where the communication is altered, even though these two forms are closely related. They both are capable of influencing
measurements. We present two solutions for resolving measurement intrusiveness. The first one is actually implemented for the \AEthereal NoC. Both solutions detect whether data belongs to the monitoring service or not.

4.3.2 Detect monitoring connection by observing packet headers

Since monitor data is transported from probe to MSA over dedicated connections, measurement intrusiveness can be solved by identifying and ignoring packets that belong to these monitoring service connections. The first word of the first flit of a packet is a header. The information in this header can be used to determine to which connection a packet belongs. A header, as depicted in Figure 4.7, contains destination queueid, source queue credits and the remaining path. NI queues are directly related to connections. The queueid identifies a queue per NI. The NI has to be known in order to derive connection from this queueid. The remaining path can be used for this purpose. Connections can thus be identified by the remaining path and destination queueid information.

\begin{center}
\begin{tabular}{ccc}
\textbf{credit} & \textbf{qid} & \textbf{path} \\
\end{tabular}
\end{center}

Figure 4.7: \AEthereal packet header

This method to avoid measurement intrusiveness requires look-up tables at each probe that contain this information for all monitoring service connections that pass that probe.

4.3.3 Add flag to flit sideband information

For the \AEthereal flit format (Figure 4.1, [3]), each word has two additional bits of sideband information. The current \AEthereal flit size is 3 words and therefore there are 6 bits available for additional information. The first two bits are used to indicate the type of flit and to indicate whether a flit contains a header or not. The second two bits express the number of payload words in the flit. One of the last two bits is used to indicate whether a flit is the last one of a packet, the other one is not used and can be used to indicate whether a flit belongs to a packet on a monitoring connection or not. This sideband bit is only visible when the last word of a flit is being transported over the link. If we want to take measurements at the level of words, a mechanism is required that stores measurement data and only uses it when it turns out that it does not belong to a monitoring connection. This could for instance be implemented by a small counter and a small buffer. The counter keeps track which word in the flit is being observed. The number of words in the flit is stored in the buffer.

4.4 Computation of performance measure from measurement data

The translation from measurement data to performance measure can be performed at the probes, the MSA or off-chip.

Computation costs can be minimized by performing the computation at a place where computational resources can be maximally shared. A good place is the MSA if multiple probes are sharing the same MSA and an off-chip process.
4.5 Support for multiple performance measures at a single probe

if multiple MSAs share the same off-chip process. The measurements typically result in integer values which can easily be represented by binary numbers. The computations however, typically result in floating point numbers which are far more expensive to represent and therefore are more expensive to transport from probe to MSA.

Consider for instance link utilization. At the probe, the number of words during a period of time is measured resulting in an integer value. To obtain utilization, the integer value is divided by the time period. This result then has to be divided by the potential throughput. This requires floating point computations and the result is a floating point number.

4.5 Support for multiple performance measures at a single probe

The monitoring service concept provides support for multiple different events at the same probe in order to share resources and therefore minimize costs. Runtime NoC performance monitoring must support multiple performance measures at different targets at the same probe. For instance, the system must be capable to support the measurement of all link utilizations at a single router. It should also support measurement of different performance metrics at the same probe.

Computational resources can be shared amongst performance measures (e.g. iT counter, timestamping mechanism, adder).

Taking multiple performance measures at the same probe also introduces new optimization possibilities to decrease the generated load.

4.6 Conclusions

Measurements for performance measures can often be taken at different places. Existing signals should be exploited as much as possible. We have shown several efficient places for the EThereal NoC for the investigated performance measures.

Except for connection latency, all discussed performance measure costs show logarithmic behavior to $\Delta T$. The results from the communication cost analysis for the investigated periodic performance measures suggest that costs are reasonable compared to NoC capacity as long as $\Delta T$ is not too low. The load generated by the connection latency measure depends on the load, the message size and the timestamp size for the monitored connection. The communication cost for the connection latency measure is the highest of all investigated performance measures. It can be lowered by looking at the latency of multiple subsequent messages rather than at a single message.
Chapter 5

Experimental cost quantification

By designing and implementing performance measures for the Æthereal NoC [3; 4] we have obtained an experimental environment which, in this chapter, is used to obtain realistic cost figures for runtime NoC performance monitoring.

5.1 Environment

The Æthereal NoC flit accurate SystemC simulator [8] is used to conduct experiments. This simulator gives us a flexible environment capable of producing realistic measures.

We have extended the simulator with runtime NoC performance monitoring to support some of the performance measures discussed in Chapter 3.

5.2 Cost measures

Here we focus on the dynamic communication costs for runtime NoC performance monitoring. An investigation of the (static) area costs for the monitoring service is presented in [7].

The Æthereal NoC simulator enables us to get realistic communication cost figures for runtime NoC performance monitoring.

We look at three cost measures in order to quantify the communication cost of runtime NoC performance monitoring. Firstly, the generated load at the probe is measured. The generated load should be the same as the one calculated in Chapter 4.

Secondly, the number of flits that is inserted in the NoC by the runtime NoC performance monitoring service is measured. This gives a fair measure for the amount of generated traffic including the communication cost (network layer costs, approximately 30%). Finally we measure consumed energy. Besides generated traffic, energy also expresses the distance traveled by the traffic as well as the energy consumed by the NIs and the routers.

5.3 Approach

As discussed in Chapter 4, the load generated at the probes heavily depends on $\Delta T$ for most of the investigated performance measures. For each periodic measure,
different $\Delta T$ values are investigated. The logarithmic behavior, as described in Chapter 4 should be visible for all the investigated performance measures. For each investigated measure we look at a low $\Delta T$ value of 10 flit clocks (60 ns), a high $\Delta T$ value of 1000 flit clocks (6 $\mu$s) and at $\Delta T$ values where the event encoding jumps from requiring one byte to two bytes.

Our current solution for runtime NoC performance monitoring should scale linearly with the amount of measurements. Experiments are run with different numbers of performance measures to verify whether runtime NoC performance monitoring scales as expected. First we quantify costs for monitoring a single type of performance measure in a configuration. This is done for link utilization, router utilization and connection latency. The events in these setups do not require an identifier. Finally we study an example where different types of performance measures are taken.

5.4 Experimental setups

In order to quantify communication costs we study an MPEG codec for which the communication is mapped on a 3x2 mesh NoC and a 3x1 mesh NoC. The MPEG application consists of multiple computation units communicating with each other through memories. For this purpose, the application uses 21 connections and each connection uses the memory mapped I/O (MMIO) protocol. IPs send data to memory and send requests for data to memory resulting in a bidirectional flow of data. The examples are presented in [16].

The MPEG 3x1 setup is an optimization of the MPEG 3x2 setup. The MPEG 3x2 setup has a slot table of 128 slots whereas the MPEG 3x1 mesh has a slot table of 8 slots. This optimization is possible because IPs are placed in a smarter way having less connections using the same link. Figure 5.1 and 5.2 shows the MPEG application IPs connected to a 3x2 mesh and a 3x1 mesh.

A probe is attached to each router in the mesh for the shown configurations. For the MPEG 3x1 example, the MSA is located at NI0005. For the MPEG
3x2 example, the MSA is located at N10102. For both setups, the connection between the probe that is connected to the upper left router in the mesh and the MSA has maximal distance. This probe is chosen for the single probe experiments in order to make energy results conservative (i.e. distance traveled by data influences energy consumption). The probed configuration is used as a basis for the experiments. This implies that the static costs (e.g. energy consumed because probes are there) are not taken into account. Table 5.1 shows the cost figures for a simulated time of 200000 ns for both the NoC configurations with runtime NoC performance monitoring disabled. It might be necessary to increase the slot table size in order to fit monitor connections on the NoC. Note that for a fair comparison, experiments are conducted with similar slot table sizes and slot reservations for the existing connections. If extra slots are required when adding monitoring service connections then the costs of the configuration with extra slots but without monitor data is taken as a reference.

<table>
<thead>
<tr>
<th>Setup</th>
<th>Flits at NIs (#)</th>
<th>energy (μJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPEG 3x2 (128 slots)</td>
<td>82127</td>
<td>28, 35</td>
</tr>
<tr>
<td>MPEG 3x1 (8 slots)</td>
<td>118936</td>
<td>18, 30</td>
</tr>
<tr>
<td>MPEG 3x1 (30 slots)</td>
<td>101916</td>
<td>17, 69</td>
</tr>
</tbody>
</table>

Table 5.1: Costs for MPEG application without runtime NoC performance monitoring

The application introduces approximately 30% more flits in the MPEG 3x1 configuration than in the MPEG 3x2 configuration. This is mainly due to the different slot table sizes and hence different slot reservations for both setups. The big slot table size for the MPEG 3x2 configuration makes it possible to allocate multiple subsequent slots per connection in the table where only for the first flit a header is added reducing the header overhead.
5.5 Link utilization

The design for the link utilization performance measure is discussed in Chapter 4. For both setups similar link utilization experiments are performed. Link utilization is measured for one link, one per router and all links. For these experiments we leave out the identifier because only one type of event occurs and we leave out the timestamp assuming that it can be reconstructed as described in Chapter 4.

As discussed in Section 2.3, good choices for $\Delta T$ values depend on characteristics of the monitored system and application. Figures 5.3 and 5.4 show the result of measurement of the link utilization of the first link of the first router for the MPEG 3x2 NoC with $\Delta T = 100$ and $\Delta T = 1000$ and respectively MPEG 3x1 configuration with $\Delta T = 10$ and $\Delta T = 100$. $\Delta T$ values for the left hand plots are chosen low compared to the slot table sizes (128 and 8). Therefore, they show noisy behavior. For the right hand plots these effects are averaged out.

![Figure 5.3: Link utilization of a link in the MPEG 3x2 mesh](image)

![Figure 5.4: Link utilization of a link in the MPEG 3x1 mesh](image)

The cost figures for all link utilization experiments are given in Table 5.2. Analogue to the calculations from Chapter 4 the presented results for load and number of flits show decreasing behavior for increasing $\Delta T$ values. However, the jumps when changing from single to double byte representation are not observed for energy and number of flits cost measures. This can be explained by the slot table mechanism and flit sizes. The current Ethereal implementation has a flit size of three words. The first word can contain a header or payload, the other
words can contain payload. Now assume that only one slot is reserved for the monitoring connection. The first word is used for the header leaving eight bytes for payload. The available bandwidth for payload in the MPEG 3x2 setup is then \( \frac{128 \times 10^6}{10^6}(B/s) = 10,42 \text{ MB/s} \) which is the minimum amount of reservable bandwidth for this slot table size. It does not matter whether we send 1 byte each 255 flit clocks (required bandwidth of 1,30 MB/s) or 2 bytes each 256 flit clocks (required bandwidth of 6,51 MB/s); it always requires a full flit to be send. Therefore it is advisable, for future research, to aggregate multiple of these small events to use flit space as efficient as possible.

<table>
<thead>
<tr>
<th>Setup</th>
<th>( \Delta T ) flitclocks</th>
<th>Load at probes ( MB/s )</th>
<th>#Flits at NIs %</th>
<th>Energy %</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPEG 3x2 (128) 1 link</td>
<td>10</td>
<td>1x16,7</td>
<td>2,05</td>
<td>0,86</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>1x1,67</td>
<td>0,64</td>
<td>0,27</td>
</tr>
<tr>
<td></td>
<td>255</td>
<td>1x0,65</td>
<td>0,31</td>
<td>0,13</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>1x1,30</td>
<td>0,31</td>
<td>0,13</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>1x0,33</td>
<td>0,08</td>
<td>0,03</td>
</tr>
<tr>
<td>MPEG 3x2 (128) 6 links</td>
<td>10</td>
<td>6x16,7</td>
<td>12,33</td>
<td>3,24</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>6x1,67</td>
<td>3,88</td>
<td>1,02</td>
</tr>
<tr>
<td></td>
<td>255</td>
<td>6x0,65</td>
<td>1,85</td>
<td>0,49</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>6x1,30</td>
<td>1,84</td>
<td>0,49</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>6x0,33</td>
<td>0,47</td>
<td>0,12</td>
</tr>
<tr>
<td>MPEG 3x2 (128) all links</td>
<td>100</td>
<td>6x3,34</td>
<td>3,88</td>
<td>1,02</td>
</tr>
<tr>
<td></td>
<td>255</td>
<td>6x1,30</td>
<td>1,85</td>
<td>0,49</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>6x2,60</td>
<td>1,85</td>
<td>0,49</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>6x0,66</td>
<td>0,47</td>
<td>0,12</td>
</tr>
<tr>
<td>MPEG 3x1 (8) 1 link</td>
<td>10</td>
<td>1x16,7</td>
<td>4,86</td>
<td>3,78</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>1x1,67</td>
<td>0,19</td>
<td>0,15</td>
</tr>
<tr>
<td></td>
<td>255</td>
<td>1x0,65</td>
<td>0,19</td>
<td>0,15</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>1x1,30</td>
<td>0,18</td>
<td>0,15</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>1x0,33</td>
<td>0,05</td>
<td>0,04</td>
</tr>
<tr>
<td>MPEG 3x1 (8) 3 links</td>
<td>10</td>
<td>3x16,7</td>
<td>14,59</td>
<td>7,56</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>3x1,67</td>
<td>1,46</td>
<td>0,76</td>
</tr>
<tr>
<td></td>
<td>255</td>
<td>3x0,65</td>
<td>0,56</td>
<td>0,29</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>3x1,30</td>
<td>0,56</td>
<td>0,29</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>3x0,33</td>
<td>0,14</td>
<td>0,07</td>
</tr>
<tr>
<td>MPEG 3x1 (8) all links</td>
<td>100</td>
<td>1x5,00;2x6,66</td>
<td>1,46</td>
<td>0,76</td>
</tr>
<tr>
<td></td>
<td>255</td>
<td>1x1,95;2x2,60</td>
<td>0,57</td>
<td>0,29</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>1x3,90;2x5,20</td>
<td>0,56</td>
<td>0,29</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>1x0,99;2x1,32</td>
<td>0,14</td>
<td>0,07</td>
</tr>
</tbody>
</table>

Table 5.2: Costs for monitoring link utilization relative to the costs of the configuration without runtime NoC performance monitoring (Table 5.1)

The load generated at the probes as calculated in Chapter 4 and verified by means of simulation is shown in the table. \( axL_1, bxL_2 \) means that \( a \) probes generate a load of \( L_1 \) and \( b \) probes generate a load of \( L_2 \). Note that for both the MPEG 3x2 and the MPEG 3x1 setup monitoring all links with a \( \Delta T \) value of 10 flit clocks was not possible without completely disturbing the original connection setup. These are therefore left out.

The load generated for a \( \Delta T \) value of 10 flit clocks (60 ns) is still considerable. We have seen that this value is very low and even exposes slot table level behavior. Except for the MPEG 3x2 setup with a \( \Delta T \) value of 10 flit clocks, runtime NoC performance monitoring scales well for link utilization. For the conducted
5.6 Router utilization

The design of the router utilization measure is discussed in Chapter 4. For both the setups, router utilization is measured for a single router and all routers. All routers have a probe and all probes communicate with a single MSA. The MSAs are again placed at the right most and bottom most router in order to maximize distance between the first probe and the MSA. The MPEG 3x2 setup has more connections sharing the same link than the MPEG 3x1 setup which is expected to result in higher utilization. This is especially the case for paths from computation IP to memory. On the other hand, more flits are created for the MPEG 3x1 setup due to the smaller slot table size. From the experiment we will derive which effect has more impact on router utilization.

Figures 5.5 and 5.6 show router utilization for all routers for the MPEG 3x2 setup and the MPEG 3x1 setup. $\Delta T$ values are chosen appropriately for the slot table sizes of both setups.

The router utilization of the bottom three routers in the MPEG 3x2 mesh are low compared to the other router utilizations in this mesh. All traffic that is transported by the lower three routers is also transported by one of the upper three routers. The upper left and upper right router utilizations for the MPEG 3x2 setup are slightly higher than than the left and right router utilization in the MPEG 3x1 setup. The center router utilization is significantly higher for the MPEG 3x2 setup. Thus we conclude that the effect of sharing links has more impact in this case than the generation of more flits due to a smaller slot table size. Table 5.3 shows the communication costs for the conducted experiments.
Chapter 5. Experimental cost quantification

The table shows a small increase in communication cost for the MPEG 3x2 router when going from single to double byte representation ($\Delta T$ from 84 to 85 for single router). Router utilization scales almost linear.

5.7 Connection latency

In Chapter 4, the design of the connection latency performance measure is discussed. For experiments with the connection latency measure we look at some of the connections of the MPEG application where communication is mapped on a 3x1 mesh NoC as shown in Figure 5.2. First, connection latency is measured for the GT connection from the audio IP block to the memory IP block. Audio sends data to memory with a load of 120 (MB/s) and a burst size of 16 bytes. The resulting message rate equals 7.5 (MMessages/s). It also sends a read request (asking memory to send data to audio) with the same rate doubling the message rate on the forward direction to 15 (MMessages/s).

The upper bound for the latency on this connection equals 744 ns (obtained by using the Atherereal design flow which is capable of giving upper bounds for latencies on GT connections). We therefore choose a timestamp size of 1 byte which supports 255 flitclocks = 1530ns. From the communication cost analysis from Chapter 4 we expect a generated load at both probes of 15 (MB/s).

Probes are attached to N10004 and to N10008 to sniff the appropriate queues. The MSA is placed at N10008. A connection from probe to MSA with a bandwidth of 15 MB/s is reserved for both probes.

Figure 5.7 (a) shows the latency of the messages on the connection from audio to memory. The number of different latencies that occur is limited and the latency plot over time shows periodic behavior. These are characteristics of GT traffic using TDMA slot tables (see A). Figure 5.7 (b) shows the distribution of the various latencies.

The costs for the connection latency experiments are given in Table 5.4.

5.8 Combining multiple performance measures

For this experiment we combine multiple performance measures. The experiment is conducted on the MPEG 3x2 configuration. We measure link utilization for all router interconnecting links (6 links) and we measure router utilization for each router. An identifier field is added to the events because there are now two types of performance measures. We compare the costs of this combined experiment with the costs for the individual experiments. The costs are given in Table 5.5.
5.9 Conclusions

The experiments show that communication costs for runtime NoC performance monitoring are low compared to the costs of the application traffic as long as a reasonable $\Delta T$ is chosen. For instance, router utilization for all routers in a 3x1 mesh running an MPEG application results in 1.46% of additional flits in the NoC and 0.76% of additional energy compared to the original application when $\Delta T$ equals 100 flit clocks.

The experiments also showed that runtime NoC performance monitoring scales well for the investigated performance measures. Number of flits and consumed energy grows linear or better when adding performance measures.

### Table 5.3: Costs for monitoring router utilization for all routers in the mesh relative to the costs of the configuration without runtime NoC performance monitoring (Table 5.1)

<table>
<thead>
<tr>
<th>Setup</th>
<th>$\Delta T$</th>
<th>Load at probes</th>
<th>#Flits at NIs</th>
<th>Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>flitclocks</td>
<td>MB/s</td>
<td>%</td>
<td>%</td>
</tr>
<tr>
<td>MPEG 3x2 (128)</td>
<td>10</td>
<td>16.7</td>
<td>2.04</td>
<td>0.86</td>
</tr>
<tr>
<td>one router</td>
<td>84</td>
<td>1.98</td>
<td>0.65</td>
<td>0.28</td>
</tr>
<tr>
<td></td>
<td>85</td>
<td>3.92</td>
<td>0.66</td>
<td>0.28</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>1x3,30</td>
<td>0.64</td>
<td>0.27</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>0.33</td>
<td>0.08</td>
<td>0.03</td>
</tr>
<tr>
<td>MPEG 3x2 (128)</td>
<td>10</td>
<td>6x16.7</td>
<td>12.33</td>
<td>3.24</td>
</tr>
<tr>
<td>six routers</td>
<td>4x84;2x63</td>
<td>4x1,98;2x2,65</td>
<td>3.86</td>
<td>1.02</td>
</tr>
<tr>
<td></td>
<td>4x85;2x64</td>
<td>4x3,92;2x5,21</td>
<td>3.86</td>
<td>1.04</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>6x3,34</td>
<td>3.88</td>
<td>1.02</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>6x0.33</td>
<td>0.47</td>
<td>0.12</td>
</tr>
<tr>
<td>MPEG 3x1 (8)</td>
<td>10</td>
<td>16.7</td>
<td>4.86</td>
<td>3.78</td>
</tr>
<tr>
<td>one router</td>
<td>63</td>
<td>2.65</td>
<td>0.76</td>
<td>0.59</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>5.21</td>
<td>0.74</td>
<td>0.58</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>1x3,34</td>
<td>0.49</td>
<td>0.37</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>0.33</td>
<td>0.05</td>
<td>0.04</td>
</tr>
<tr>
<td>MPEG 3x1 (8)</td>
<td>10</td>
<td>3x16.7</td>
<td>14.59</td>
<td>7.56</td>
</tr>
<tr>
<td>three routers</td>
<td>2x63;1x50</td>
<td>2x2,65;1x3,33</td>
<td>2.47</td>
<td>1.28</td>
</tr>
<tr>
<td></td>
<td>2x64;1x51</td>
<td>2x5,21;1x6,54</td>
<td>2.42</td>
<td>1.26</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>3x3.34</td>
<td>1.46</td>
<td>0.76</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>3x0.33</td>
<td>0.14</td>
<td>0.07</td>
</tr>
</tbody>
</table>

Table 5.4: Costs for monitoring connection latency relative to the costs of the configuration without runtime NoC performance monitoring (Table 5.1)

<table>
<thead>
<tr>
<th>Setup</th>
<th>load at probes</th>
<th>#Flits at NIs</th>
<th>Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MB/s</td>
<td>%</td>
<td>%</td>
</tr>
<tr>
<td>MPEG 3x1 (8)</td>
<td>15</td>
<td>6.7%</td>
<td>1.6%</td>
</tr>
<tr>
<td>audio to mem</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1 connection)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The sum of the individual costs equals 5.71% extra flits and 1.39% extra energy. In this case, the cost for the combined experiment is lower than the sum of the individual experiments. This is due to a better flit efficiency, which cancels out the effect of the extra byte needed for the identifier.

### 5.9 Conclusions

The sum of the individual costs equals 5.71% extra flits and 1.39% extra energy. In this case, the cost for the combined experiment is lower than the sum of the individual experiments. This is due to a better flit efficiency, which cancels out the effect of the extra byte needed for the identifier.

5.9 Conclusions

The experiments show that communication costs for runtime NoC performance monitoring are low compared to the costs of the application traffic as long as a reasonable $\Delta T$ is chosen. For instance, router utilization for all routers in a 3x1 mesh running an MPEG application results in 1.46% of additional flits in the NoC and 0.76% of additional energy compared to the original application when $\Delta T$ equals 100 flit clocks.

The experiments also showed that runtime NoC performance monitoring scales well for the investigated performance measures. Number of flits and consumed energy grows linear or better when adding performance measures.
Chapter 5. Experimental cost quantification

(a) Connection latency as a function of the time at which a message is offered to the NoC

(b) Distribution of connection latency

Figure 5.7: Connection latency for single connection for connection from audio IP to memory

<table>
<thead>
<tr>
<th>Setup</th>
<th>$\Delta T_{\text{flitclocks}}$</th>
<th>Load at probes $MB/s$</th>
<th>#Flits at NIs $%$</th>
<th>Energy $%$</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPEG 3x2 (128)</td>
<td>2x63;4x84;6x255</td>
<td>2x5,30;4x3,97;6x1,31</td>
<td>4,16</td>
<td>1,10</td>
</tr>
</tbody>
</table>

Table 5.5: Costs for combination of link utilization and router utilization measures relative to the costs of the configuration without runtime NoC performance monitoring (Table 5.1)
Chapter 6

NoC congestion control

Control systems have shown their value in many electrical engineering applications. In this chapter we present NoC congestion control as part of Quality-of-Service (QoS) management and show an example of NoC congestion control for theEtherereal NoC.

6.1 Introduction

6.1.1 Goal for NoC congestion control

It is well known that an increase in network utilization results in an exponential increase of connection latencies (see for instance [11]). Our aim is to control network utilization in order to circumvent network congestion with the ultimate goal of keeping latency of BE connections bounded.

This goal is reached by bounding the utilization of single links. If we extend this to the complete network the target is to put the utilizations of all links in the network to an upper bound. We choose 75% as target for link utilization. This is a conservative value. Other choices for link utilization values have no influence on the presented control method.

6.1.2 NoC congestion control as part of QoS

As described in Section 2.3.5, QoS managers can exploit network resource information to make good decisions.

NoC congestion control is part of QoS management. Either a global or a local QoS manager decides that certain network resources are not used in the right way and subsequently undertakes the right control action. It has the choice of either reconfiguring the network to comply more with the circumstances (this is only possible if there are enough network resources) or it applies control to slow down or speedup certain processes. Applying control does not require an application to shut-down. On the other hand, NoC feedback control requires applications to be controllable and can mean a degradation of quality. Of course, the QoS management system must be aware of all these trade-offs to make the right decisions.

6.1.3 Control theory

Control theory [17] is a field of research that is concerned with the control of the inputs of a process with the purpose of steering its outputs toward a desired value (often called reference variable). There are many control techniques in the field
of control theory ranging from classical to modern approaches. In the classical approach, systems are modeled and controlled using transfer functions. Figure 6.1 shows a simple feedback control system where the output of process (P) is fed back and subtracted from a desired value (input). The resulting error is used as the input of a controller which applies the right amount of control to get the desired output value. The classical approach is well suited for design problems with single inputs and single outputs (SISO) but for more complex systems, the modern approach is preferred.

![Feedback control system](image)

**Figure 6.1: Feedback control system**

The modern approach uses state feedback rather than output feedback. For this technique, a process is modeled by a state space description which consists of input, output and state variables related by differential equations for the continuous case and difference equations for the discrete case. These equations capture all system dynamics. The variables and equations can easily be written as vectors and matrices. More inputs and outputs only imply an increase in vector and matrix size making this technique highly scalable. It is therefore well suited for so-called multiple input multiple output (MIMO) systems.

A controlled system is said to be stable if and only if it satisfies the bounded input bounded output (BIBO) condition. This states that when the system is fed with arbitrary bounded inputs over an arbitrary time the outputs must also be bounded. For a linear discrete system, this condition is satisfied when all poles of its transfer function (points where the denominator of the transfer function equals zero) are located within the unity circle. For more complex systems (including non-linear systems) stability is typically analyzed by means of a Lyapunov function.

The controller proposed in this chapter is a model predictive controller (MPC) [18] which is a special type of optimal controller. Optimal controllers are modern controllers that are designed by optimizing a cost function [19]. This cost function takes into account the dynamics of the system. Note that optimal should not to be read as the most optimal controller for a specific problem. MPCs solve cost functions on line rather than at design time.

### 6.1.4 Point of control

The ability to control a process at runtime is a fundamental requirement for control systems. There are two obvious places to control link utilization. Firstly, control can be applied to the IP sources that generate data on the NoC. Secondly, NoC resources can be scaled by adjusting NoC clock speed.

At IP level, load can be controlled by using for instance voltage scaling, by degrading e.g. video or audio quality or by entirely shutting down jobs or disabling parts of jobs that are running on an IP. Steering the speed of the network is unique for on-chip networks such as NoCs. Increasing network speed results in more network bandwidth and subsequently decreases network congestion.

For our examples we focus on control of IP load. Cases with single and multiple controllable IP sources are studied.
6.2 NoC congestion control for Æthereal

6.2.1 Introduction

The Æthereal NoC [3; 4] supports two types of communication services, GT and BE (see Section 2.1.2). A configuration can contain only GT, only BE or a combination of BE and GT connections.

The interesting configurations for applying feedback control are those with only BE connections and those with a combination of GT and BE connections. It is not obvious to introduce feedback control in a configuration with only GT connections. All connections use reserved resources and there is no traffic that can exploit the unused network resources of the GT connections. This makes it relatively easy for a QoS manager to predict network utilization. Furthermore, GT connections have guarantees concerning latencies on which high network utilization has no influence.

For simplicity, we study the utilization at a single link. Consider a NoC configuration with a number of GT and a number of BE connections. The traffic on the GT connections has variable bitrate (VBR) (e.g. MP3 encoded music with VBR or MPEG4 encoded video with VBR). A QoS manager can not predict the rate at which the data is produced and therefore has to rely on runtime measurements. The BE traffic uses its desired bandwidth if it is available. This traffic can for instance be generated by a video stream where constant high quality is not that important (e.g. videophone). Now consider a single link in the NoC. Typically such a link will transport data for multiple connections. Figure 6.2 (a) shows utilization at a link where BE traffic has constant bitrate (CBR) and GT traffic has VBR. The link utilization gets above its desired value frequently. Now we apply control to steer the BE traffic by for example disabling parts of jobs or using voltage scaling at the IPs with the goal of putting link utilization at a fixed level. The link utilization displayed in Figure 6.2 (b) has a constant, low level, which is optimal from a BE latency point of view.

Figures 6.12 (a) and (b) show link utilization and latency for a BE connection that uses the link when no control is applied. Figures 6.13 (a) and (b) show the same performance measures when control is applied. We see that the BE connection latency for the uncontrolled NoC increases significantly during the pulse shaped increase of link utilization. For the controlled NoC, both the link utilization and the BE connection latency show only a small peak.
6.2.2 Characteristics of the control problem

There are two latencies that put bounds on the control mechanism. $T_{\text{forward,prop}}$ is the propagation delay from the controlled entity to the observed link. $T_{\text{reverse,prop}}$ is the time that it takes for the observed signal to reach the controlled entity. The combination of both latencies is called round-trip latency:

$$T_{\text{round_trip}} = T_{\text{forward,prop}} + T_{\text{reverse,prop}}$$

The negative effect that this latency has on the controlled system is twofold. Traditional control design methods determine process dynamics and appropriate control schemes at design time. The variable nature of $T_{\text{round_trip}}$ can change the system dynamics in such a way that the closed loop control system becomes unstable making the design of a robust system by means of traditional methods tedious. Secondly, $T_{\text{round_trip}}$ limits the reaction speed of the controlled system making the control system speed slow compared to the speed of the total system limiting the ability to cope with fast changing outputs.

The presented control problem has multiple inputs (IP sources) and multiple outputs (links). Classical control methods are not well suited for multiple-input multiple-output (MIMO) systems. Furthermore, classical control methods cannot cope with varying latencies.

In [20], an internet congestion control method is proposed that uses classical control. The method assumes per connection output queuing at routers and uses the queue levels as output and connection load as input. This generalization results in a single-input single-output (SISO) control problem for which classical control can be used.

This approach does not fit the NoC congestion control problem. The fact that utilization at a single connection or buffer is high does not imply a high utilization of the used output link. In fact, overall network congestion can be fairly low while some of the connections have relatively high loads. A fair measure for congestion at a link is the utilization of that link.

Predictive control methods are modern control methods that are known for their capability of dealing with varying latencies. In the remainder of this chapter, we look at model-based predictive control as a control method for NoC congestion control.

6.2.3 Model predictive control

Model predictive control is a technique that combines model based predictions with actual system measurements to make good control decisions.

The basic concepts of MPC are:
6.2. NoC congestion control for Ethereal

Model based predictions A MPC tries to predict future process behavior by looking at how a process model behaves under the measured output and the current and future inputs.

Current input selection A MPC optimizes a cost function $J$ for each control step to determine the best input values.

Prediction horizon The prediction horizon determines the time that a MPC looks ahead to predict the future.

Tuning parameter $\lambda$ This parameter is used to tune a MPC from very robust to very fast.

Constraints Constraints can be set for the manipulated outputs indicating maxima, minima and rise and fall speeds.

Figure 6.3 shows a MPC as controller for NoC congestion control. The inputs for the MPC controller are the utilizations of the shared links (outputs of the controlled process). The outputs of the controller are the loads for the BE connections (inputs of the controlled process).

The optimization of cost function $J$ is typically performed by using quadratic programming (QP), this is also the case for the MPC used by the Matlab MPC toolbox. The fact that this algorithm is performed at runtime makes this the major cost of MPC. Refer to [18] for more details about MPC and to [21] for more details on the MPC used by Matlab (including optimized function description).

![Figure 6.3: MPC for NoC congestion control](image)

6.2.4 Link utilization model for multiple sources

MPC uses a model to predict future behavior. Figure 6.4 shows the NoC model for multiple IPs putting ($IP_1...IP_n$) load on a shared link. In Chapter 4 we observed that link utilization is measured at flit level. IPs have no notion of flits instead it is natural to refer to IP load in (M)bytes/s. The relation between load in MBytes/s and number of occupied slots is not straightforward. The load that is provided by a single slot (reserved in slot-table for a certain connection) depends on slot-table size and number of consecutive slots reserved in the slot-table by a connection. Ethereal offers a raw link bandwidth of 2 GBytes/s. This is shared amongst all the slots. Each slot provides a raw bandwidth of $2 \cdot 10^9 / slot\_table\_size$ bytes/s. Part of this bandwidth is used for communication costs (packet header containing transport information). The remaining part is used for payload. The number of occupied slots due to a certain load depends on how efficient the slots are used. Adding the complete load/flit relation to the model would add a significant amount of complexity. Therefore, the complete model is at MBytes/s level. We
assume that slots are used with 100% efficiency and that a header is added to each flit (this holds for BE traffic) implying a communication cost of 33%. 100% link utilization at flit level now corresponds with a load of 1333.33 MBytes/s. For instance, if the desired link utilization equals 50%, the combined load of the IP connections sharing the link are controlled to 666.67 MBytes/s. Note that the granularity at which link utilization can be controlled depends on the slot-table size. For instance, a slot-table size of 16 has 83.13 MBytes/s steps whereas a slot-table size of 64 has 20.83 MBytes/s steps. This discrete behavior of link utilization could also be modelled, again adding complexity.

The path from IP to shared link is modeled by a communication cost factor $k$ and a time delay estimate for $T_{\text{forward},prop} (z^{-f_1}...z^{-f_n})$. The load at a link is modeled by an adder which accumulates the load of multiple IP sources. In the presented model it is assumed that the control process is centralized. The time delay from monitor to control process is modeled with an estimate for $T_{\text{reverse},prop} (z^{-r})$.

![Figure 6.4: Model used for MPC](image)

### 6.3 Experiments for Āthereal NoC congestion control

#### 6.3.1 Introduction

As an example we consider a 3x1 mesh NoC configured with a single GT and a single BE connection. The GT connection is used for traffic with VBR with a maximum of 1 GB/s from IP A to IP C. The BE connection of IP B tries to transport 500 MB/s of payload from IP B to IP C. The situation is depicted in Figure 6.5. The used NoC has three routers and three NIs (one for each router). The connections share two links in the NoC. In this case it is sufficient to observe one of these links because they share the same connections. The link capacity for Āthereal equals 2GB/s. The control goal is to bound link utilization to 75% and hence link load to 1500 MB/s.

An MPC controller and a probe to measure link utilization are added to the system to meet this goal. Figure 6.6 shows the NoC in combination with the controller and the probe. The controller steers the BE traffic generated by IP B.

#### 6.3.2 Matlab simulation

Matlab's MPC toolbox is used to design the NoC congestion controller. The model from Section 6.2.4 is entered in combination with a set of parameters in order to design the MPC controller. The delay from BE source to link is estimated at 9 flit clocks, the delay from link to controller is estimated at 6 flit clocks. The latency difference is due to a higher utilization of links on the forward path due
6.3. Experiments for Aethereal NoC congestion control

IP A has reserved a GT connection of 1 GB/s and produces data in a variable manner. Link capacity of 2 GB/s

Figure 6.5: NoC configuration for (3x1 mesh with two connections)

IP B uses a BE connection with a bandwidth of 500 MB/s

Figure 6.6: 3x1 mesh with control for a single link

to the load generated by B. The round-trip latency is therefore estimated at 15 flit clocks.

An important parameter for the controller is its control speed which determines with what speed the controller observes the system and with what speed the controller controls the system. \( \Delta T \) (see Section 2.3) is chosen equal to the control speed. A control speed of 100 flit clocks (600 ns) is chosen for the experiments. This gives a reasonable reaction speed (as will be shown in the remainder of this chapter) at reasonable communication costs (6.67 MB/s for link utilization at \( \Delta T = 100 \), see Chapter 5 for realistic cost figures).

In order to test the reaction speed of the controlled system, we apply a pulse shaped GT input as shown in Figure 6.7 (a). A pulse of 700 MB/s is put on top of a load of 300 MB/s after a small period of time resulting in a GT load of 1000 MB/s. In combination with the desired BE load of 500 MB/s this results in a shared link load of approximately 2000 MB/s. The open loop response for this input is depicted in Figure 6.8 (a). During the GT pulse, the link utilization is above 75%. Of interest for this experiment is the time between the measurement of the high link utilization and the time at which the link utilization is controlled to 75% (1500 MB/s).

A slow and a fast (compared to the control speed of 100 flit clocks) sine shaped load is put on top of a constant load to see how the system behaves under constantly changing load. Both the sine waves have an amplitude of 200 MB/s and are put on top of a GT load of 850 MB/s. The slow sine wave has a time period of 5000 flit clocks (Figure 6.7 (b)) whereas the fast sine wave has a time period of 16 flit clocks (Figure 6.7 (c)). The controller should be able to cope with the slow sine wave. The fast sine wave will however not be suppressed. The fast sine wave is applied to show the limitation of the designed controller. The \( \Delta T \) value of 100 flit clocks will average out the fast sine wave. The open loop responses for fast and slow sine wave are shown in Figure 6.8 (b) and (c). The mean value of the load is above the targeted 1500 MB/s. For this experiment we
are interested to see how much the amplitude of the sine waves is suppressed and if the mean load is controlled to 1500 MB/s.

In figure 6.9, the responses of the controlled system are shown. Matlab plots the responses as observed by the MPC controller, thus at the control speed of 100 flit clocks. Figure 6.9 (a) shows the response of the MPC controlled system for the pulse shaped GT load. The response shows a peak when the pulse is encountered, the load is controlled to 1500 MB/s in 200 flit clocks (1.2 μs). The response to the slow sine wave as depicted in Figure 6.9 (b) shows that the sine wave is not fully suppressed. The amplitude of the resulting sine wave is equal to 20 MBytes/s. This is low compared to the amplitude of 270 MBytes/s for the response to the fast sine wave as shown in Figure 6.9 (c) (note that this response is plotted at a different scale as the fast sine input). For both the sine wave responses the mean load is controlled to 1500 MB/s.

Smaller ΔT values increase reaction speed and suppress the sine waves. The trade-off is between controller performance and communication cost. Figure 6.10 shows the responses to the pulse and the fast sine wave input when a controller with a control speed of 1 flit clock is used. The reaction speed is now 30 flit clocks (0.18 μs). The fast sine wave is suppressed to an amplitude of 40 MBytes/s.

The final Matlab experiment is conducted to show the limitation caused by the round-trip latency. For this experiment we apply a sine wave shaped input with an amplitude of 200 Mbytes/s and a frequency of 7 flit clocks (= round-trip latency) on top of a load of 850 MBytes/s as shown in Figure 6.11 (a). The control system is not able to suppress this behavior.

6.3.3 Æthereal simulation results

In the previous section, experiments with an MPC controlled system were conducted using Matlab. In order to see how the NoC congestion controller behaves in a more realistic environment we use the Æthereal simulator. For this purpose, the designed controller is added to the simulator.

We again apply a pulse shaped GT load as shown in Figure 6.7. A fair The open loop response for this double pulse input is shown in Figure 6.12.

The latency for the open loop response (Figure 6.12 (b)) jumps from 80 ns to 200 ns during the utilization pulse. Figure 6.13 shows the pulse response when control is applied. The reaction speed equals 1.2 μs and is equal to the value obtained by the simulation from the previous section. Typically, Matlab results and Æthereal simulator results are not exactly the same, the latencies in the Æthereal simulator are dynamic as opposed to the Matlab latencies.

The latency of the BE connection (Figure 6.13 (b) now shows a peak and is controlled back to 110 ns within 1.2 μs.

6.4 Conclusions

In this chapter we shown feasibility of NoC congestion control using runtime NoC performance monitoring by means of MPC for a small congestion control problem with one BE and one GT connection. Experiments with the Æthereal NoC have shown that the control method is stable even under varying latencies. Furthermore, the reaction speed is in the order of several microseconds.

Further research with more complex examples (e.g. multiple BE/GT connections, multiple shared links, larger networks) is necessary to see whether the presented method is as scalable as expected.
6.4. Conclusions

Plant Inputs

- (a) Pulse shaped input
- (b) Slow sine input
- (c) Fast sine input

Figure 6.7: Different inputs for Matlab simulation
Chapter 6. NoC congestion control

Figure 6.8: Open loop responses for different inputs
6.4. Conclusions

Clock cycles

(a) Response to pulse input

(b) Response to slow sine input

(c) Response to fast sine input

Figure 6.9: Responses for controller with control interval of 100 flit clocks
Chapter 6. NoC congestion control

Figure 6.10: Responses for controller with control interval of 1 flit clock
6.4. Conclusions

Clock Cycles

(a) Fast sine input (faster than round-trip latency)

(b) Response to fast sine input

Figure 6.11: Sine wave faster than round-trip latency
Figure 6.12: Link utilization (a) and latency (b) of a connection using the link without NoC congestion control
Figure 6.13: Link utilization (a) and latency (b) of a connection using the link with NoC congestion control
Chapter 7

Conclusions and recommendations

7.1 Conclusions

The answer to the research question stated in Section 1.1 is: Runtime NoC performance monitoring is both feasible and usable. The remainder of the conclusions is used to explain why this is the case.

In the following enumeration, the numbers relate to the objectives stated in Section 1.3.

1. We proved feasibility by designing and implementing runtime NoC performance monitoring for \AEthereal at reasonable communication cost and have given insight in the trade-offs between communication and computation for measurements (see Chapters 4 and 5).
   (a) The proposed framework for runtime NoC performance monitoring effectively defines the performance metrics and the performance measure targets that are of interest (see Chapter 3).
   (b) We proposed two methods to overcome the problem of measurement intrusiveness (see Section 4.3).
   (c) Techniques are introduced to minimize communication costs (see Chapter 4).

2. Experiments with the \AEthereal simulator have shown that communication costs of runtime NoC performance monitoring are low compared to the communication costs of an MPEG application. For instance, monitoring utilization for all routers in a 3x2 mesh with a sample period of 100 flit clocks (600 ns) results in 3.86% extra flits and 1.02% more energy (see Chapter 5).

3. We have given an application for runtime NoC performance monitoring: Network resource monitoring for QoS management. In Chapter 6.1, an example of resource monitoring for QoS management is given in the form of a feedback control mechanism that prevents network congestion.

MPC is a good method for controlling congestion in networks because it is stable under varying latencies and supports the multiple input multiple output nature of this control problem. The reaction speed (time between observation of the congestion problem and complete remedy of the congestion problem) of this control problem is bounded by the roundtrip latency.
Experiments with the Æthereal simulator show that the reaction time of the controlled system is in the order of several microseconds which is considered reasonable for real-time applications.

7.2 Recommendations and future research

• Design and implementation:
  - Implement runtime NoC performance monitoring in hardware (VHDL).

• Optimizations to reduce communication costs:
  - Research adaptive $\Delta T$ based on analysis of monitored data at probes.
    (trade-off between computation and communication)
  - Adapt event size to monitored NoC. For Æthereal this can for instance be done by adapting the event size to the available room for payload data in the reserved flits (2 words if only one flit is reserved).
  - Round measurement data to bits, then aggregate multiple measurement data to come as close as possible to a full multiple of a byte.
  - Investigate smart techniques for routing and scheduling of monitor data.

• Applications of runtime NoC performance monitoring:
  - Apply NoC congestion control for other NoC setups (e.g. multiple BE/GT connections, multiple shared links, bigger networks)
  - Research other (predictive) control techniques for NoC congestion control to see how they compare to MPC.
  - Add runtime NoC performance monitoring to a QoS management system.
  - Research more applications for runtime NoC performance monitoring such as communication centric debugging.
References


REFERENCES


Appendix A

Verification of an SDF model of the Æthereal NoC

In [22] a synchronous data flow (SDF) model is presented that models the timing dynamics of packet switched NoCs. In this section, a single connection mapped on the Æthereal NoC is modeled by using the presented model. The timing derived from the model is then compared to timing obtained by runtime NoC performance monitoring for the Æthereal simulator. Figure A.1 shows the setup for this experiment.

Load of 46.8 MBytes/sec
Message size of 6 words

Figure A.1: Setup for SDF verification experiment

The setup shows a master communicating with a slave through a NoC consisting of two NIs and one router. The configuration contains one connection \( C_1 \) which consists of a forward channel and a reverse channel. In this case, only the forward channel is used to transport payload, the reverse channel is used for end-to-end flow control. The sizes of the input buffer at master side and the output buffer at slave side are both 40 words. The master produces payload with a load of 46.8 MB/s and a burst size of four words. The payload is transported to the slave by using a memory mapped I/O protocol. This requires a command and address field making the complete message size six words (four words of payload, one word for command, one word for address). The slot table size equals five (one slot contains one flit, in our case one flit contains two payload words). One slot of the slot table is reserved for the forward channel of \( C_1 \). The same holds for the reverse channel.

The SDF model shown in Figure A.2 is based on this information. Master P produces data with 6 words at a time (message size). It can only do so if enough space is available in the input buffer, this is modeled with a token of size...
The time of a single slot table revolution is 30 ns. In our case, we can transport 2 words per slot table revolution. The offered bandwidth is therefore equal to 

\[ 2 \cdot 4/(30 \cdot 10^{-9}) = 267 \text{ (MB/s)} \]

which is more than sufficient for the offered load. Data can only be scheduled if there is room available at the output buffer at the slave side, this is again modeled by a token of size 40 (end-to-end flow control). Router and NI kernel both require 6 ns to fulfill their tasks.

![SDF model for verification experiment](image)

In this case, the producing NI never has to wait for space to become available in the consuming side buffer. Therefore, message latency only depends on what time the message is offered to the producing side buffer. In the best case, the message is offered to the input buffer just before the slot of \( C_1 \) has its turn causing the first two words of the message to be sent immediately. It then takes two slot table revolutions (2·30 ns) to schedule the complete message of six words. It takes 3·6 ns to transport the message through NI1, R and NI2 subsequently. In the worst case, the message is offered just after the slot of \( C_1 \). Now three slot table revolutions (3·30 ns) are required to schedule the complete message. It still takes 3·6 ns to transport the message through the network.

\[
L_{\text{message}} = 2 \cdot 30 + n \cdot 6 + 3 \cdot 6, 0 <= n <= 5
\]

\[
L_{\text{message, best case}} = 2 \cdot 30 + 6 + 6 + 6 = 78 \text{ ns}
\]

\[
L_{\text{message, worst case}} = 3 \cdot 30 + 6 + 6 + 6 = 108 \text{ ns}
\]

Runtime NoC performance monitoring is used to obtain connection latency measures per message. The result is shown in Figure A.3. The worst case message latency of 108 ns occurs only once, when the first message is offered to the NoC. Message latency is further homogeneously distributed between 78 and 102 ns. All latencies fit the general formula and do not exceed worst or best-case calculations. We therefore conclude that the SDF model is correct for this case.

For the second example, end-to-end flow control is made critical by using an output buffer of size 1 at the slave side. The SDF model for this case is shown in Figure A.4. Now, the NI at master side can only send one word at a time and has to wait for end-to-end flow control to send the next word.
Chapter A. Verification of an SDF model of the Æthereal NoC

We assume that a buffer position available when a message is offered to the input queue at the sending side. In the best case, the first word of the message is immediately transported through the forward channel. Because there is only one token available, the rest of the words have to wait for flow control.

\[
L_{\text{best\_case}} = 6 + 6 + 6 + 5 \cdot (6 + 6 + 6 + 6 + 6 + 6) = 198 \, ns
\]

We again assume that a buffer position available when a message is offered. For the worst case, slots are always just missed for both payload and end-to-end flow control adding 30 ns for both the forward and the reverse channel.

\[
L_{\text{worst\_case}} = 30 + 6 + 6 + 6 + 5 \cdot (6 + 30 + 6 + 6 + 30 + 6 + 6 + 6) = 528 \, ns
\]

Figure A.5 shows the simulation result for the setup with critical end-to-end flow control. We again see a homogeneous distribution. Although all latencies are between best and worst case, both these times do not occur during simulation. The explanation for this is that both the calculated best and worst case situations can not occur in the simulation environment. In the calculations we assumed that words are sent and flow control is generated during arbitrary slots. This is the case for BE like connections where the latency can vary significantly. But for the studied GT connection, the latency is fixed. Therefore, the slot at which the first word of a message is offered to the sending NI determines the timing of the complete message. This is implicitly modeled in the SDF model.
Figure A.4: SDF model for verification experiment with critical end-to-end flow control, 1 flit clock (fc) equals 6 ns

Figure A.5: Distribution of connection message latency obtained by simulation with runtime NoC performance monitoring