Simulating SCI and SCI/RT in Simula

Bjørn Bakke
Department of Informatics
University of Oslo

10th August 1995

PREFACE

This document contains my thesis for the Cand Scient degree at University of Oslo (UiO) and my advisor has been Stein Gjessing at the Department of Informatics, UiO.

The thesis describes the work related to designing a simulator for a subset of the Scalable Coherent Interface protocol (SCI), published in a standard by the Institute of Electrical and Electronic Engineers (IEEE). Some of the modifications proposed in relation to SCI/Real-time (SCI/RT, IEEE P1596.6) have also been incorporated. The simulator has been employed to investigate the performance of various aspects of the protocol-subset and the SCI/RT modifications.

During the early stages of the thesis, my advisor suggested that I should write the thesis in English and thereby making it more accessible — both in Europe and in USA there were people associated with SCI. I hesitated because my native language was Norwegian and not English, and my recent experience with the latter was limited to having read English professional books at lower grade. Nevertheless, I decided to write my thesis in English and found it a useful experience. It was also a laborious experience because it sometimes proved difficult to compose the correct sentences expressing the correct meaning without making them too hard to read.

I would like to thank Stein Gjessing who has been an enthusiastic advisor and who has given me help and advise of great value, people associated with SCI with whom I have discussed SCI and SCI/RT, fellow students and Cybernetisk Selskab who have created a social atmosphere here at UiO and my family who has supported and encouraged me throughout my study.

Bjørn Bakke
Department of Informatics
University of Oslo

[This page has been intentionally left blank]

Contents

[This page has been intentionally left blank]

Chapter 1

Introduction

This chapter gives an introduction to the thesis and indicates how the issues within the research fields of computer architecture, real-time systems and software development initiated the work with this thesis (section 1.1). Later in this chapter, a summary of the thesis' main goals and results are given, together with an argument for why this work was considered interesting (section 1.2 and 1.3). At the end of this chapter the main structure of the remainder of the thesis is described (section 1.4).

A new way to design parallel computers with multiple processors and memory chips called Scalable Coherent Interface (SCI) [IEEE, 1992a], has been proposed by the Institute of Electrical and Electronic Engineers (IEEE). This standard will be referred to as the *SCI-standard* or *SCI-protocol* for the remainder of this thesis.

The SCI-standard describes a protocol, governing the communication between multiple processors and memory chips, and this thesis considers a subset of this protocol. One of the goals in relation to this thesis has been to design a simulator for this protocol-subset and with the help of the simulator, to analyze the performance of this subset. The simulator was written in Simula, a high-level programming language supporting object-oriented programming, and another goal was to use the object-oriented programming technique. Because I wasn't familiar with SCI, a thorough study of the SCI-protocol was required before the simulator could be designed, and it was hoped that an object-oriented programming strategy would ensure flexibility and modifiability.

At the time of writing, there are several IEEE standardization working-groups related to SCI, investigating properties of the SCI-protocol or defining extensions to it. One of these working-groups is the SCI/Real-time working-group (SCI/RT), which tries to modify the SCI-protocol, for real-time purposes. Some of the enhancements proposed within the SCI/RT working group has been considered in this thesis, and the enhancements has been incorporated into the simulator and their behavior and performance have been analyzed.

1.1 Background

Developing computers is a quest for increased computer-power in terms of speed and capacity. There seem to be a constant demand for bigger and faster computers and as soon as the new computers are available they are quickly saturated by executing more and larger programs. To meet this constant demand, the single-processor has been developed and various techniques have been introduced, like **pipelining** and **caching**. Pipelining refers to how machine-instructions in the processor are executed in a pipelined fashion and

according to [Kogge, 1981] and [Hennessy and Patterson, 1990] the first general-purpose pipelined machine was the Stretch-IBM7030 (1959), though already the UNIVAC 1 (early fifties) overlapped the program execution with some I/O operations. Caching refers to the technique of storing recently used memory-locations in a small, local memory physically close to the processor, possibly located on the processor chip itself. In this way the relatively higher access latency of the shared memory is avoided. As with pipelining, caching is not a new technology and according to [Hennessy and Patterson, 1990] the first paper on caching was published in 1965 by Wilkes, and in [Wilkes, 1965] the use of fast core memory as slave to a slower core memory is discussed.

The technology of single processor machines are far developed and highly tuned, but there seem to be another approach needed in order to meet the future demand for higher speed and capacity. The speed of light limits the maximum signal speed and there seem to be a lower bound of the size of chips, used when the processors are implemented. Instead of putting all the effort into developing single processor computers, some people choose to focus upon parallel computers. In that way processes can be distributed among the processors.

The latency represented by memory-accesses is also a major factor governing the overall performance of the computer. If we wish to use multiple processors, we will quickly realize that using a single large physical shared memory will reduce the performance because the memory will become a bottle-neck. This problem can be solved with multiple memory blocks, each block representing a part of the total address space of the shared memory.

When a computer with multiple processors is designed and later when programs are developed for the this computer, there are different problems encountered on the software level and the hardware level. On the software level there are at least two major approaches as to how the programmer could view such a computer, the first being **shared memory** and the second being **message passing**. The shared memory approach is perhaps the conceptually simpler of the two, because most programmers are familiar with this approach from single-processor computers. In a parallel computer with shared memory there is one global memory accessible to all the processors. Message passing involves the passing of messages between processes, e.g. when the a process wants to read a location in memory, it will have to send a message to the process that controls the memory.

Depending on which approach is used on the software level, the hardware level id affected to a variable degree. Message passing does not require heavy modification of the hardware while the shared memory approach require careful hardware design. From a performance point-of-view it is desirable to have processors with local caches, but in the shared memory approach this will make things more complicate. The problem arise when several processors read the same memory-location and store the location in their respective caches. As long as the processors keep reading that memory-location there will not be any problem, in fact this resembles the situation in a single processor machine. However, once a processor wishes to modify the memory location we realize that the other caches have to be notified in some way. The problem facing us is how to make sure that the processors have a correct view of the shared memory, often referred to as **the cache coherence problem**. If we use backplane bus when designing a computer with multiple processors, we can achieve cache-coherency by using a strategy called **snooping** [Goodman, 1983]. Unfortunately the bus has its disadvantages, among them are its lack of scalability, because the bus is still a shared resource which have to be allocated to the processors in a one-at-a-time fashion. In chapter 2, the problems related to bus-based parallel computers will be discussed more thoroughly, together with some alternatives.

In the second half of the seventies, the use of **directories** to identify the caches sharing the same memory line was proposed by [Tang, 1976] and [Censier and Feautrier, 1978]. Instead of broadcasting update messages to all caches, they propose to send update-messages to each individual cache identified by the directory. One distinguishes between a **centralized directory**, where a central directory identifies the caches storing the same line, and a **chained directory**, where a chained list identifies the sharing caches [Chaiken et.al., 1990]. The latter alternative is also referred to as a **distributed directory** because the control information is distributed among the caches. (Chaiken et.al. also distinguishes between full-mapped and limited central directories, but the essential thing here is the difference between central and distributed directories)

One approach to multi-processor shared memory computers is proposed by the Institute of Electrical and Electronic Engineers (IEEE) in their standard called "SCI-Scalable Coherent Interface" [IEEE, 1992a]. This standard describes a physical interconnect and a protocol, showing how a computer with multiple processors and distributed shared memory can be designed. The SCI-standard describes a protocol which ensure cache-coherence and this part is clearly based on the ideas proposed by [Tang, 1976], [Censier and Feautrier, 1978] and [Chaiken et.al., 1990], because the SCI cache-coherence protocol uses a distributed directory to identify the sharing caches. One of the goals when SCI was developed, was to ensure scalability so that the performance increases when the number of processors and memory chips increase. In chapter 2, a more thorough introduction of the SCI-protocol is given.

When the SCI-project began in 1988, the people who were involved in the project abandoned the bus-approach when they realized the bus' lack of scalability and limited signal speed (bidirectional links cannot have the same high signal speed as unidirectional links). Stein Gjessing, Stein Krogdahl and Ellen Munthe-Kaas at the University of Oslo (UiO) became involved in the project when the SCI working-group wanted a formal verification of the SCI-protocol. Already one of the major research projects at the Department of Informatics at UiO had been to develop methodology and techniques which could be employed when programs were specified and verified. While SCI-protocol was (and still is) formally specified in C-code, formal verification of the SCI-protocol could be achieved by using the theory developed at UiO. The results of this work is presented in [Gjessing et.al., 1990a], [Gjessing et.al., 1990b] and [Gjessing and Munthe-Kaas, 1991].

At the time of writing the SCI-technology is relatively new, and no real-world computer based on SCI is known to the author of this thesis. The SCI-standard does indeed describe the expected behavior and protocols, but it does not describe *how* the protocol should be implemented.

Even if there *were* SCI-based computers available, it is still difficult to change hardware parameters and characteristics, and people have turned to other techniques to analyze the behavior and performance of SCI. Different approaches have been used, some people have developed programs using various programming languages (a software approach), others have developed mathematical models.

Software simulations of SCI involves designing a program in a programming language like C [Kernighan and Ritchie, 1988], C++ [Stroustrup, 1991] or Simula [Dahl et.al., 1982], or in a specially designed simulation tool like Verilog/VHDL. Different interconnect topologies, load-characteristics and traffic patterns can be simulated and investigated by specifying various parameter-values. Simulators for SCI known to the author of this thesis are:

- Bogaerts and Wu at Cern, Geneva, have designed a simulator for SCI consisting of a re-implementation of the transaction protocol using MODSIM-II and the IEEE C-code of the cache coherence protocol as given in the SCI-standard [IEEE, 1992a]. The simulator can also simulate SCI-interconnects consisting of multiple rings and multiple switches. A document-draft describing the simulation environment is currently available, refer to [Bogaerts and Wu, 1995] for further details.

- Bothner and Hulaas at the University of Oslo have designed a simulator for SCI and the logical layer implemented in C-code. The simulator has been used in performance evaluation of k-ary n-cubes running the SCI-protocol. Refer to [Bothner and Hulaas, 1993] for further details.

- At the University of California San Diego (UCSD) a simulator for SCI has been designed using MODSIM-II and the simulator has been used to investigate the performance of an extension to the SCI-flow control mechanism proposed at UCSD (presented in [Picker et.al., 1994]). Refer to [Picker and Fellman, 1994] for further details on the simulator developed at UCSD.

- Stein Gjessing [1] at University of Oslo has designed a simulator for SCI-rings using Simula, displaying the packet transmission as well as gathering statistical information.

- Hexsel and Topham at Edinburgh University have designed a simulator for SCI consisting of an approximate model of the SCI-link interface and a detailed model of the cache coherence protocol. The simulator is used in a performance evaluation of a shared memory multiprocessor using SCI. Refer to [Hexsel and Topham, 1994] for further details.

- The SCI-protocol itself is expressed in C-code, and this code is executable. Refer to [IEEE, 1992a] for further details.

- Scott, Goodman and Vernon at University of Wisconsin, Madison, have designed a simulator and developed a mathematical model for SCI, considering the packet transportation layer. The simulator and the mathematical model have been used when they analyzed the performance of SCI-rings. Refer to [Scott et.al., 1992] for further details.

The mathematical approach involves developing a model in which mathematical formulas describe the system. In [Scott et.al., 1992] this kind of work is described.

In fall 1993, Stein Gjessing who already had been involved in SCI for several years, realized that it was necessary and interesting to simulate some of the aspects of SCI and proposed this as Cand. Scient. thesis work. The starting point was to design a program simulating SCI using Simula, a programming language developed by people at Norwegian Computing Center [Dahl et.al., 1982] in the sixties, and use this simulator when the the performance of SCI was investigated. This thesis therefore describes work which belongs to the group of software simulation of SCI, because the simulator was written in Simula.

A project related to SCI is the SCI/real-time (SCI/RT) and the work takes place in an IEEE standardization working group. This working group is officially referred to as

[1] Stein Gjessing, University of Oslo, P.O.B. 1080 Blindern, N-0316 Oslo 3, Norway. Email: steing@ifi.uio.no

P1596.6 and its current chairman is Ralph Lachenmaier [2]. The goal of the project is to modify the basic SCI-protocol for real-time purposes and at the same time ensure that it remained compatible to the basic SCI-protocol. Whereas the SCI-protocol emphasize the correctness and efficiency of the computations performed, the real-time aspect introduce additional requirements - the computations have to be ready on time. A typical real-time system contains a set of tasks that have to be performed at periodic intervals (sampling operations), and some task are more important than others. The load can usually be determined *a priori*, so the computer system must be designed with this in mind and make sure that all tasks can be executed and terminate within their deadlines. The effort of the SCI/RT working group has not yet led to an approved IEEE standard, a *standard-draft* is available [IEEE, 1992b], and the latest proposals are, therefore, not included.

The modifications proposed within the SCI/RT working-group can roughly be said to belong to two different sets, the first being proposals that tries to modify the basic SCI-protocol so that the resulting system obeys the requirements in Rate Monotonic Scheduling (RMS) [Liu and Layland, 1973], and the second approach being proposals that tries to use a token-based scheme. Real-time systems in general, RMS and SCI/RT will be described more thoroughly in chapter 2.

1.2 A summary of the thesis' goals, work and results

1.2.1 The goals

The main goal of this thesis has been to design a modifiable and flexible program written in Simula that simulates a subset of the SCI-protocol, and to use this simulator in a performance analysis of SCI. Furthermore, the performance of various ring-sizes, load and traffic patterns should be analyzed, and these characteristics should be given as parameters to the simulator. It should also be possible to investigate the performance of various interconnect topologies, consisting of multiple rings and switches. A secondary goal has been to modify the simulator so that the performance of some of the SCI/RT-proposals could be analyzed.

More details can be found in chapter 3.

1.2.2 The work

This thesis involves work that can be divided into two logical stages. On the first stage the simulator were designed and built, and on the second stage the simulator was used in simulations related to the performance analysis. In reality, stage two began while stage one was still in progress, because simulations were ran if the simulator could provide interesting results, even though is was not complete. The modifications required to incorporate SCI/RT-modifications, were carried out while baseline SCI simulations were running.

Designing and building the simulator introduced work which also can be divided into two stages. Not being familiar with the SCI-protocol, a thorough study was required, and considerable time and effort was spent before the simulator could be designed. Again, there is no exact point during the work which marks the transition from understanding SCI-protocol to simulating it, instead there was a considerable amount of parallel work.

[2] Ralph Lachenmaier — P1596.6 Chairman, Code 505B, NAWC/ADW, Warminster, PA 18975, USA. Email: lachenmaier@nadc.navy.mil

The SCI/RT project was (and still is) very active, and can therefore be rather frustrating to a person who already is struggling to understand the basic SCI-protocol. In order to terminate the work within time, some of the enhancements proposed by the SCI/RT working group were incorporated in the simulator, while others were left out. Nevertheless, incorporating SCI/RT was one of the biggest challenges in this thesis, partly because it gave the opportunity to follow the changes within the real-time community, and partly because it would give an indication on how modifiable the simulator was.

More details can be found in chapter 4 and 5.

1.2.3 The results

The results are related to the design process of the simulator and the performance of SCI and SCI/RT.

Designing the simulator required a thorough understanding of the SCI-protocol and its modifications proposed in relation to SCI/RT. Designing the simulator proved to be laborious because the SCI-protocol is complex and describe expected behavior, not how to implement it. It has been discovered that it benefits both the understanding of SCI and the process of representing it as a simulator, to work on issues related to the two simultaneously (understanding and representing). It has also been discovered that it is possible to design a modifiable and flexible simulator for the SCI-protocol, and which later can be modified, without extensive redesign, so that modification to the SCI-protocol itself, e.g. SCI/RT, can be simulated. The programming strategy was object-oriented and this proved to be a fairly successful strategy, because the SCI/RT modifications could be incorporated without changing the structure of the simulator. Simula supported the object-oriented programming strategy, but Simula's run-time system affected the design process and an ad-hoc strategy had to be employed to achieve reasonable efficiency. This ad-hoc strategy which sought to reduce the amount of dynamic allocation, sometimes conflicted with the object-oriented programming strategy.

To analyze the performance of SCI and SCI/RT, various interconnect structures were simulated under various conditions. It has been discovered that the load, the size of send-packets, the load and traffic pattern, the ring size and flow control mechanism affect the performance of an SCI-ring, and these results correspond to those presented in [Scott et.al., 1992]. It has also been discovered that the SCI-ring is not scalable in terms of throughput and latency. Simulating an interconnect with multiple SCI-rings and multiple switches, indicate that a multi-ring interconnect could be a better alternative than a single SCI-ring. The SCI flow control mechanism has also been found to ensure fairness among nodes in an SCI-ring.

Related to SCI/RT, preemptive-priority output-queue and bypass-queue have been simulated. The results from these simulations indicate that the above modification ensure that throughput and latency are related to priority (a high priority means a higher throughput and lower latency than for a lower priority).

Simulations were run for a considerable amount of time and confidence intervals were calculated for the more important estimates. If the confidence intervals were found to be too broad (more than $\pm 5\%$) the results were rejected, and new and longer simulations were ran. It is therefore reasonable to believe that the results obtained through the SCI-simulator are reliable.

More details can be found in chapter 6, 7, 8 and 9.

1.3 Reasons for doing this work

Designing a program simulating a real-world system correctly is quite difficult, because correctness of the simulator is hard to prove. Formal verification is not always possible, and simplifications towards the real-world system are often assumed or else the program could grow as complicated as the real-world system.

Consequently there is always *some* uncertainty related to simulation-results and it is, therefore, still meaningful to design new simulators when comparable simulators already exist. Results already discovered can be compared to new results obtained using another simulator. If the new simulator was designed according to a different strategy and using another programming language, comparable simulations results will strengthen the existing ones.

It is also reasonable to assume that a software simulator would be easier to modify than the real-life counterpart. E.g. in the context of SCI, the size of send-packets has impact on the overall performance, and using a simulator to analyze this aspect we would only have specify a different parameter-value, whereas in the real-life system we would, quite possibly, have to design a new chip. Simulation is therefore an economical technique to examine systems too costly to implement for analysis-purposes alone.

This thesis will describe the design process of an SCI-simulator, performance results will be compared to existing results and additional simulation results are presented.

1.4 The structure of the thesis

The main structure of the thesis is as follows:

- Chapter 2 gives an introduction to SCI, SCI/RT and the various concepts needed in the remainder of this thesis. Chapter 2 will focus upon those parts of the SCI-standard that have been considered important in the work and that have been simulated. No knowledge of SCI is assumed on behalf of the reader but it is assumed that he or she is familiar with the basic concepts of computer architecture.

- Chapter 3 describes the main issues of this thesis. Some of the issues have already been revealed in chapter 1, but chapter 3 will go into more detail.

- Chapter 4 describes the design process of the simulator and indicates the programming strategy used. It begins with an argument on why a programming strategy is needed, then turns to the programming strategy itself describing it and why object-oriented programming is believed to support it.

- Chapter 5 describes the simulation work, and shows how the simulation-results were obtained. It starts with a description on which topologies that have been simulated and how the simulation work itself were carried out.

- Chapter 6 presents and discusses results achieved from the simulation of single SCI-rings, performed under various assumptions regarding load, traffic pattern, packet sizes and ring sizes. These assumptions is thoroughly described in chapter 5, in relation to the section of single ring topologies.

- Chapter 7 presents and discusses results achieved from the simulation of a multi-ring interconnect, consisting of four SCI-rings and four switches. This topology and other

conditions assumed during simulation are described in chapter 5 in relation to the section of multi-ring interconnects.

- Chapter 8 presents and discusses results achieved from the simulation some of the enhancements proposed in relation to SCI/RT. The various assumptions and parameter settings used in these simulations is described in chapter 5 in relation to the section of SCI/RT simulation.

- Chapter 9 contains the conclusion of the thesis, and will conclude on the original issues of the thesis, described in chapter 3, present additional results and indicate further work.

- Appendix A contains three proposals on how to represent the underlaying model of the transmitter stage entity.

At the end of each chapter a brief summary is given.

1.5 Summary

This chapter has given an introduction to the thesis, indicating its historical background, summing up its goals and results, and indicating its structure.

The general background of the thesis is the quest for computer power, a quest which has led to the development of technology like pipelining and caching. Because of physical limitations of signal-speed and size, some people choose to focus their attention on parallel computers rather than single processor machines. The protocol of Scalable Coherent Interface (SCI) [IEEE, 1992a], published by the Institute of Electrical and Electronic Engineers (IEEE), describe one way to design a parallel computer with multiple processors and distributed *shared* memory. When SCI was developed (1988-1992), University of Oslo (UiO) got involved in the project when some parts of the SCI-protocol required formal verification. Verification had already been a major research topic at UiO for several years. Several people at UiO got involved in SCI, among them were Stein Gjessing, Stein Krogdahl and Ellen Munthe-Kaas. SCI became an IEEE standard in 1992, but several related projects still exist, like SCI/Real-time (SCI/RT) which try to modify the SCI-protocol for real-time purposes. Currently the SCI/RT working-group has meetings 3-4 times a year.

The work with the thesis began in fall 1993, after Stein Gjessing had proposed the task of simulating SCI in Simula as the starting point. As work progressed, it was decided to include some aspects related to SCI/RT, and the SCI-simulator had to incorporate some of the modifications proposed in relation to SCI/RT.

As indicated in this chapter, several SCI-simulators already existed when work began, but designing a new SCI-simulator was still meaningful. Some uncertainty is always associated with simulation-results (programming-errors are one of several reasons) but if there are two different simulators producing the same simulation-results, our confidence in them increase.

Chapter 2

Introduction to SCI and SCI/RT

This chapter gives an introduction to Scalable Coherent Interface (SCI) and SCI/real-time (SCI/RT). In chapter 1, a very brief introduction to SCI and its historical background was given in relation to the introduction to the thesis, but in order to apprehend the remainder of this thesis, a more thorough introduction is required, hence this chapter. It is not assumed that the reader has any knowledge of SCI, but it is assumed that he or she is familiar with the basic concepts of computer architecture. There are three main sections in this chapter.

Section 2.1 will briefly describe the historical background of SCI.

Section 2.2 gives an introduction to the SCI-protocol itself. Rather than explaining only those parts of SCI-protocol which is strictly necessary in the remainder of this thesis, section 2.2 will give an introduction which could also serve as an overview of the SCI-protocol. Both approaches have their drawbacks, and the first approach may be unsatisfactory from the SCI point-of-view, because important elements necessary for the overall understanding would be left out. The second approach, which is chosen here, force the reader to go through more information than is strictly necessary, but the parts which can be omitted are marked in the text.

The SCI-technology is described in the IEEE standard P1596-1992 [IEEE, 1992a] and in brief terms this standard describe a physical interconnect and a protocol using the interconnect as a communication medium. Computers based on SCI-technology are parallel computers with multiple processors and shared distributed memory, and one of the goals when SCI was developed was to ensure scalability. This means that the performance of an SCI-based computer should increase when the number of processors increase. The SCI-protocol also ensure cache-coherence, which mean the caches are consistent at all times, and that the processors have a correct view of the shared memory.

Section 2.3 gives an introduction to real-time systems in general and to SCI/RT in particular. SCI/RT is currently one of several SCI-related activities and its goal is to enhance the basic SCI-protocol in a way that makes it suitable in real-time systems. Real-time systems have additional requirements compared to time-shared systems like SCI, and usually contain periodic tasks whose timing and deadline is critical for the systems. At the time of writing the effort of the SCI/RT working group has not yet led to an approved IEEE standard, but several enhancements of SCI have been proposed. Section 2.3 will describe some of these enhancements, emphasizing the proposals whose performance are investigated in the analysis later in this thesis.

2.1 The historical background and development of SCI

One of the main goals when new computers are developed is to make them faster and more powerful than existing computers. People involved in research fields like numerical analysis, database-programming and system-design seem to be in constant need of computer power. When new resources are granted, they can be easily spent, for example by running more programs.

In order to meet the demand for more computing power, processors have been developed continuously, and in the course of time, technology like **caching** and **pipelining** have been used extensively. Single processor computers are very far developed, and increasing performance is getting more and more laborious. The speed of light limits the signal speed and there also seem to be a lower bound of how small the chips can be made.

One way to meet the demand for faster computers is to use multiple processors. The processors have to cooperate and traditionally there are two main strategies to achieve this, either **message passing** or **shared memory**.

Communication between processors has traditionally taken place via a bus. A bus is a large set of wires running through the computer, to which all entities are connected, and there is a protocol governing the bus-allocation.

The bus-structure and -protocol may be very simple, but has some disadvantages. A bus is bidirectional and this limits the signal speed, and the contact point between an entity and the bus cause the signals to reflect which also limits the signal-speed. The bus-designer also has to make sure that the signal is able to propagate between all entities without interruption, and therefore he or she has to cater for the worst case situation. The worst case situation takes place when two entities located at either end of the bus wish to communicate, and the signal has to travel all the way, from one end to the other, before any other entity can access the bus. When the bus increase in size, the propagation delay will increase also. Perhaps the biggest problem with the bus is that the bus-protocols are based on a "one-at-a-time" strategy, and the bus becomes a bottleneck when traffic increase. All in all, the bus does not scale well when the number of entities connected to it (processors and memory-chips) increase.

People who previously had been involved in projects like Fastbus (IEEE960) and Futurebus (IEEE896), trying to develop high-speed buses, realized these limitations and in 1987 the so-called Superbus Study Group was started. This group decided to try another approach and in 1988 a new IEEE working-group was started. Their goal was to design a high speed interconnect that scaled well, support both message passing and shared memory, and in the latter case ensure cache coherence.

One way to reduce the latency of memory-access is to use caching. A cache is a memory-chip located close (physically) to the processor or even on the processor-chip itself. The processor store recently used memory-lines in the cache temporarily to avoid the long latency of accessing the shared memory. A processor may also use caching on several layers. In a computer with multiple processors and where the processors use caching, one is faced with the task of keeping all caches updated at all times or at least consistent, so that the processors always have a correct view of the shared memory This is often referred to as the **cache coherence problem**. If a bus is used, the cache coherence problem can be solved by a technique with a strategy called **snooping** [Goodman, 1983].

The interconnect emerging from the SCI working-group was not so much a broadcast medium as the traditional bus, and a different approach to the cache coherence problem was required. Their solution was to use **directories** to identify caches storing the

same memory-line, and a message-passing protocol was defined to pass update messages to those caches. The cache coherence problem in general and the use of directories to identify caches storing the same memory-location has been discussed in [Tang, 1976] and [Censier and Feautrier, 1978]. In [Chaiken et.al., 1990] it is distinguished between between a **centralized directory**, where a central directory identifies the caches storing the same line, and a **chained directory**, where a chained list identifies the sharing caches. In SCI the directories are doubly linked lists, consisting of caches sharing the same memory-line, and the SCI cache coherence protocol is therefore said to be **distributed directory-based**. (Refer to section 2.2 for further details on the cache coherence protocol in SCI).

The SCI working-group considered it necessary to verify the cache coherence formally, and this in turn involved the University of Oslo (UiO). For several years, one of the major research topics for years at UiO had been to develop formal verification techniques.

SCI became an IEEE-standard in 1992, and today there are still much activity going on. Companies like Dolphin, Unisys and Apple have implemented or are trying to implement the SCI-standard in hardware.

2.2 The SCI protocol

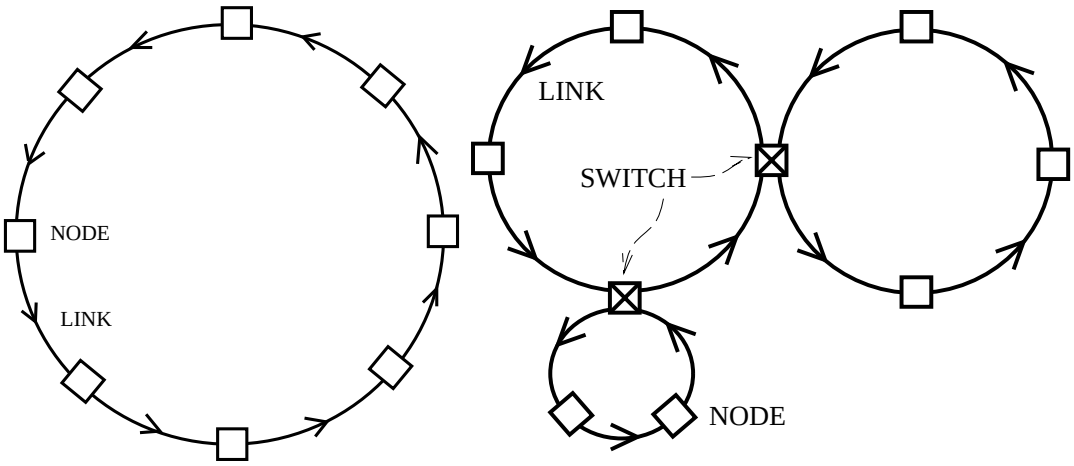
This section gives an introduction to the SCI-protocol, and it is sought to give an overview of the protocol. Those parts of the SCI-protocol which are investigated later, in relation to the performance analysis, are emphasized, while the other parts are described briefly in order give an overview of SCI and can be omitted by the reader.

The SCI-protocol is specified in the SCI-standard [IEEE, 1992a] published by the Institute of Electrical and Electronic Engineers (IEEE). The SCI-standard is a document consisting of two main parts, the first part is an introduction and tutorial for SCI, and the second part is the SCI-protocol formally specified in the C programming-language. This C-code part of the SCI-standard is the formal definition of the SCI-protocol and as such takes precedence over the tutorial. The C-code should be consulted when the tutorial seem ambiguous, but experience indicate that the C-code should be avoided until basic knowledge of the SCI-protocol is acquired.

At the time of writing no books describing the SCI-protocol is known to the author of this thesis, except for the above-mentioned SCI-standard. Several articles describing the SCI-protocol and various related aspects have been published, and those articles provide an alternative approach to SCI, like [Picker et.al., 1994], [Scott et.al., 1992] and [Bothner and Hulaas, 1991].

The SCI-standard describes a physical interconnect and an inter-processor protocol using the interconnect as a communication medium. When two entities, e.g. a processor and a memory controller, wish to communicate and exchange information they communicate according to the protocol sending their messages on the interconnect. Different entities, even different processors, can communicate because they use the same protocol.

A computer based on the SCI-protocol typically consists of multiple processors and distributed memory. The physical interconnect and the inter-processor protocol allow multiple parties to communicate at the same time, and in this way multiple processor can execute different programs and at same time access the same memory, i.e. the memory is shared by several processors. According to the classification system proposed by Flynn in [Flynn, 1972], a computer based on the SCI-protocol would belong to the class of machines described as **multiple instruction stream, multiple data stream**, abbreviated



(a) Single ring (b) Multiple rings

Figure 2.1: Examples of SCI-interconnects (logical structure)

MIMD. The SCI-community, represented by David B.Gustavson and Qiang Li, has proposed **Local Area Multi Processor** abbreviated **LAMP**, as another way to classify SCI in [Gustavson and Li, 1995].

The logical structure of the physical interconnect in an SCI-based computer are made up of entities referred to as **links**, **nodes** and **switches**. A link connects one node to another, thereby creating a point-to-point link which enables the first node to transmit to the other node, but not vice versa. The link is unidirectional and three different types of links have been defined in the SCI-standard to provide flexibility - a parallel electrical link, a serial electrical link and a serial optical link. Associated with each node are two links, one for input and one for output, and by combining links and nodes a ring structure will be created, for example a ring with eight nodes as in figure 2.1a. A switch can be regarded as a special type of node, and it can have several link-pairs associated with it, which enables multiple rings to communicate. An interconnect with three rings is shown in figure 2.1b as an example.

The concept of node is not fully defined in the SCI-standard, and design issues are left to the person who implements it. A node has a unique destination address, and is a collection of several smaller entities. It contains an **interface** to the ring, a **transfer-cloud** and the **application entities**. An application entity can be a processor, a memory chip, a cache or a combination of these three entities communicating locally via a bus or by other means. A node example is shown in figure 2.2.

As an example of inter-processor communication consider the situation where processor A wishes to read a location in memory. As already mentioned the physical memory in an SCI-based machine is distributed, and multiple memory chips compose the total shared physical memory. Processor A do not have to know the physical whereabouts of the memory chip containing the requested location, it simply passes the read-request to the transfer-cloud. The following actions will take place (the memory chip with the requested location is placed in node B):

- Processor A will issue a read-request and pass it to the local transfer-cloud. The

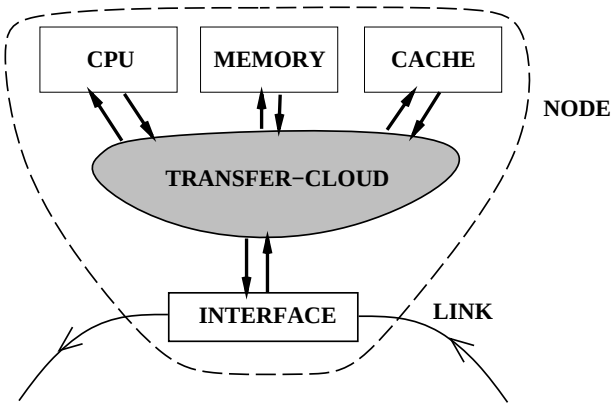


Figure 2.2: An SCI node.

task of the transfer-cloud is to handle a message issued by the application entity and possibly translating the message into a **packet** or a sequence of packets if the request involve communication with other nodes.

Assuming that the requested location is not stored in the local cache or the value there is invalid, the transfer-cloud has to translate the read-request into a packet and transmit it to the transfer-cloud in node B requesting the new value. The transfer-cloud will carry out its task by passing the packet to the interface which will put them onto the ring.

When the packet was created in the transfer-cloud of node A, it was given a unique destination address, and it will traverse the ring simply by passing those interfaces having a different address, and will finally be absorbed by the interface in node B. If the destination is located in another ring, the packet has to be moved from this ring to another by a switch and possibly perform several of these "ring-hops", before it reaches its destination.

- When the packet has reached the interface in node B, it will be passed on to the transfer-cloud associated with the interface. The transfer-cloud translate the packet into a message understood by the application entity and passed on. In this case the application entity is the memory chip containing the requested location, and when the read-request is received it will return the new value to processor A. In order to return the value in the requested location, a read-response is issued by the memory chip in node B and passed on to the transfer-cloud and the above process is reversed.

This brief description of communication in an SCI-based machine show two important properties of the SCI-protocol:

1. The SCI-protocol is layered, and consists of a physical layer, a packet transportation layer and a cache coherence layer. In this written order the layers correspond to an increasing level of abstraction, also shown in figure 2.3. Conceptually the communication takes place only between entities on the same layer and the entities communicate by using the services provided by the layer below.

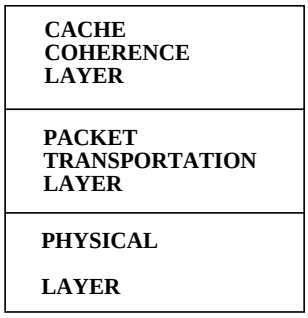


Figure 2.3: Layers in the SCI-protocol

2. The complexity of the interconnect is invisible to the application entities, and a processor can issue its usual memory-requests regardless the physical location of the memory. For example the machine can be upgraded by exchanging an old processor with a new processor of a different kind, and it is only the local transfer-cloud which have notice the difference. The local transfer-cloud has to be modified in order to cater for the different messages of the new processor.

Entities on the cache coherence layer provide services to application entities, like processors and memory chips, and make use of services provided by entities on the packet transportation layer. The cache coherence layer can be implemented by a transfer-cloud. The task of the entities on the cache coherence layer is to translate requests from the application entities into a sequence of packets, and transmit those packets to the destination transfer-cloud. The cache coherence layer is discussed further in section 2.2.2, but can be omitted by the reader because details on the cache coherence layer is not needed in the remainder of this thesis.

Entities on the packet transportation layer provide services to entities on the cache coherence layer, and make use of services by entities on the physical layer. The packet transportation layer is implemented by an interface. The task of an entity on the packet transportation layer is to transmit packets issued by the entities on the cache coherence layer to the destination entity on the packet transportation layer. The packet transportation layer is discussed in more detail in section 2.2.3.

Entities on the physical layer provide services to entities on the packet transportation layer, and is implemented by a link. The task of the link is to transmit packets from one interface to another, by transmitting the packets byte-by-byte or bit-by-bit. The physical layer is discussed briefly in section 2.2.4, but can be omitted also, while details of the physical layer is not needed in the remainder of this thesis.

The concepts of saturation and ring-circumference are defined in section 2.2.5.

2.2.1 A processor's view

Above the top-layer of the SCI-protocol (figure 2.3), the physical properties of the interconnect is invisible to the application entities. A processors will simply issue its usual memory-requests when it wishes to read a location in memory, and it will receive a response with the data in this location. A processor does not have to consider the physically location of the memory-address when the read-request is issued.

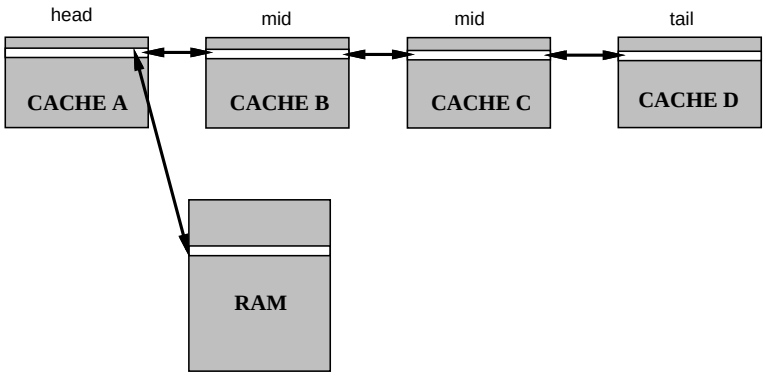


Figure 2.4: An SCI cache-list.

A memory chip will respond to memory request in the ordinary way, and regardless where the request originated.

2.2.2 The cache coherence layer (can be omitted)

The task of the cache coherence layer is to provide service to the application entities like processors, cache-chips, memory-chips or a combination of these, offering a shared memory with cache coherency. In order to implement the services of the cache coherence layer, the service provided by the packet transportation layer is used, and functionality added by combining these services.

A cache stores control information in addition to the data itself, and for each cache line a number of tags are reserved for the control information. For every memory-line (in the physical shared memory) being cached, a list is maintained containing all the caches that store the memory-line. This is a doubly linked list, and the tags are used to store the address of the predecessor and successor cache element. Because a list is maintained identifying the caches storing the same memory-line, and the control information are distributed among the caches, the cache coherence layer of SCI is said to be **distributed directory based**. The use of directories to identify the caches storing the same memory line was first proposed by [Tang, 1976] and [Censier and Feautrier, 1978], and [Chaiken et.al., 1990] distinguishes between centralized and chained directory (distributed list). An example of a cache-list is shown in figure 2.4, where four different caches store the same memory-line in the shared memory.

The protocol is **invalidation-based**, which mean that cache lines are marked invalid when they no longer contain correct values, rather than updating the incorrect values. The cache coherence layer handle three basic operations which manipulates a cache-list, insertion, deletion and reduction. These operations are atomic, which mean that once an operation is started it is guaranteed to run uninterrupted until termination. In brief terms the three operations perform the following action:

Insertion: A new cache element is inserted into a cache-list. This operation is used when a cache wants a copy of a specific memory-line.

Deletion: A cache element is removed from a cache-list. Deletion is used when the cache wants to perform a cache roll-out, i.e. the cache needs space for more recent data.

Reduction: A cache element in a cache-list is put in front of the list, and all the remaining cache-elements are marked invalid. This will reduce the cache list to a list of only one element. Reduction is used when a cache wants to modify a cache-line.

With multiple processors running simultaneously, several processors may wish to read or write the same memory-line, and this memory-line may or may not be stored in the local cache of the processors, which again may or may not imply a cache roll-out. This mean that multiple insertion, deletion and reduction operations can be initiated simultaneously acting on the same cache-list. The SCI cache coherence layer guarantees that the caches are consistent and that the processors have a correct view of the shared memory despite the complex situations which can arise.

Further details related to the cache coherence protocol can be found in chapter "Cache Coherence Overview" of the SCI-standard [IEEE, 1992a], and formal specification and verification of the protocol is discussed in [Gjessing et.al., 1990a], [Gjessing et.al., 1990b] and [Gjessing and Munthe-Kaas, 1991].

2.2.3 The packet transportation layer

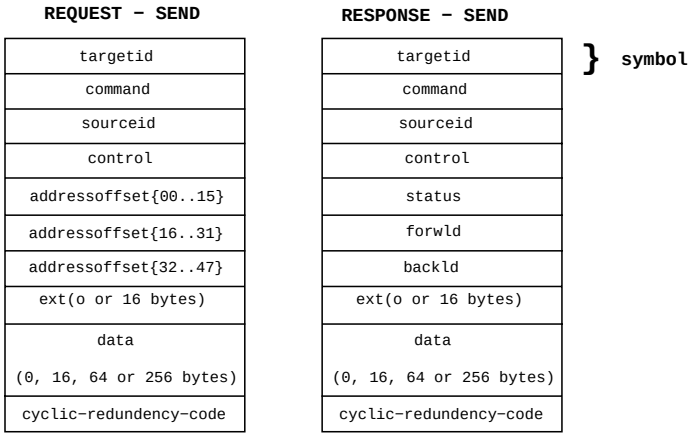
The main task of the packet transportation layer is to provide services to the cache coherence layer, services which include transmission of packets across the interconnect. The packet transportation layer make use of the services provided by the physical layer to implement its service. The physical layer provide unidirectional point-to-point transmission of bytes.

The entities that implement the packet transportation layer is often referred to as **interfaces** because they act as an interface to the physical interconnect, hiding the physical properties of the interconnect. On the packet transportation layer the communicating parties are the interfaces. Depending on our point of view, this interface could be referred to as a ring-interface (viewed from the cache coherence layer) or as a node-interface (seen from the interconnect). In the remainder of the thesis the term **node-interface** will be used when referring to an entity implementing the packet transportation layer.

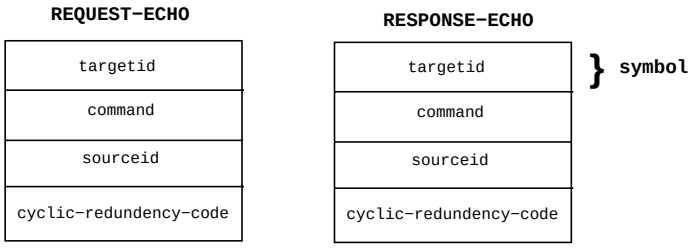
Each node-interface have one input-link and one output-link. While every node-interface have two links and the links are point-to-point, the logical structure of the interconnect is that of a ring. Using switches, entities which have more than one node-interface, multiple rings can be connected and various topologies can be formed. Since the links are unidirectional all information in a ring move in the same direction.

The interfaces communicate by exchanging packets, which are finite sequences of symbols. A **symbol** is 2 bytes and the smallest information fragment transmitted between node-interfaces. A link transmit one symbol at a time.

There are two main types of packets, **send-packets** and **echo-packets**. A send-packet carry information generated by an entity on the cache coherence layer to the destination node-interface, whereupon an echo-packet is returned as an acknowledgment. The send-packets are further divided into two sub-types, **request send** packet and **response send** packet, each corresponding to the requests and responses generated at a higher layer. echo-packets can also be divided into two sub-types, **request echo** and **response echo**, and act as an acknowledgment of request and response send-packets respectively. Figure 2.5 show the logical structure of the various packet types, and the type of information stored in each symbol. The contents of the majority of these symbols are not considered in this thesis because they are used on a higher layer. Only the **targetid** (packet's destination address),



(a) Send-packets



(b) Echo-packets

Figure 2.5: Packet types

sourceid (packet's source address), **command**, **control** and **cyclic-redundancy-code** (error checking code) are considered important in the remainder of this thesis.

When a node-interface wishes to transmit a send-packet, it will transmit the packet on the output-link, symbol by symbol, to the downstream neighbour. The send-packet carry the destination address, and if this address do not match the address of the neighbouring node-interface, it will be passed on to the next node-interface. This will go on until the packet reaches the destination node-interface and will there be stripped from the ring. The receiving node-interface immediately returns an echo-packet as an acknowledgment, and this packet continue back to the send-packet's originator. The send-packet may also pass through several switches, going from one ring to another. More details on communication between node-interfaces are found in the sections «Packet transmission protocol» and «Input-queue allocation protocol» below.

It is important to note that a packet is never interleaved by other packets. Unless strictly necessary, a passing packet is never stored completely in the intermediate node-interface before it is passed on to the next node-interface. **Worm-hole routing** is used in a node-interfaces when possible, and symbols are moved from the input-link to the output-link as quickly as possible.

Between packets, the node-interface is required to insert at least one **idle-symbol**, which is a symbol carrying ring-local information concerning ring-priority and flow control. The idle-symbol has several bit-fields which have special significance, e.g. the **lg bit-field** which will be discussed in section «Ring bandwidth allocation protocol». The required idle-symbol also eliminate differences in transmission-rate and receive-rate of neighbouring node-interfaces. In the absence of packets, the node-interface generate a continuous stream of idle-symbols, which serve as a synchronizing mechanism. This mean that there is a continuous stream of symbols on the link, either packet symbols or idle-symbols. Assuming that the links implement the physical layer of the highest capacity defined in the SCI-standard (various physical layers with various capacity have been defined, refer to section 2.2.4), one symbol is transmitted every 2nd nanosecond and therefore the bandwidth is 1Gbyte/sec per link.

In each ring, one node-interface has added responsibility and perform special ring maintenance tasks. This node-interface is called the **scrubber**, and there is exactly one scrubber in each ring. Its main task is to remove packets with corrupted destination-addresses from the ring, packets which would otherwise circulate the ring indefinitely.

The logical structure of the node-interface has been defined in the SCI-standard and is shown in figure 2.6. The **multiplexer** and the **stripper** act as interfaces to the output-link and the input-link respectively, hiding the physical nature of signaling. The **request output-queue** and **response output-queue** store temporarily the request and response packets from the cache coherence layer until they can be transmitted. The request input-queue and response output-queue store temporarily request and response packets received and addressed to the node-interface. The packets in the input-queues will be removed by the cache coherence layer. The **bypass-queue** store passing packets temporarily when the node-interface is busy transmitting a packet from one of the two output-queues, and will be emptied as soon as the transmission is done. The **save-idle buffer** is used in relation to the flow control mechanism, and will be discussed in more details in section "Ring bandwidth allocation protocol". The **receiver-stage** handles the incoming symbol stream on the input-link and controls the stripper (possibly implemented *by* the stripper). It will strip off packets addressed to the node-interface, store send-packets in one of the input-queues, replacing the packet with idle-symbols and pass those symbols to the transmitter-stage.

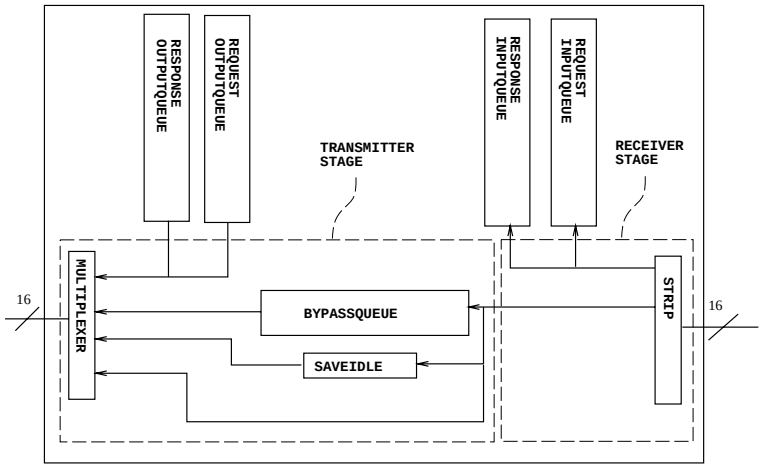


Figure 2.6: The logical structure of the node-interface

Packets which are not addressed to the node-interface are passed on to the transmitter-stage as they are. The **transmitter-stage** controls the multiplexer (possibly implemented *by* the multiplexer) and will transmit a symbol in each clock-cycle. The transmitter-stage has to decide whether to transmit from one of the output-queues, the bypass-queue, the save-idle buffer or directly from the receiver-stage.

The service of the packet transportation layer provided to the cache coherence layer is reliable and error free packet transmissions. To achieve this, rules governing various aspect of node-interface communication have been defined in the SCI-standard. The communication between node-interfaces follows a fixed pattern of transmitting send-packets and receiving echo-packets, and the rules related to this is discussed in section «Packet transmission protocol». Forward progress is emphasized throughout the SCI-standard on the different layers, because in a multi-processor interconnect a lack of forward progress will lead to dead-lock situations. The packet transportation layer is no exception in this respect, because both the ring interconnect and the input-queues in the node-interfaces are shared resources, and rules are needed to control how and when these resources are allocated. Section «Ring bandwidth allocation protocol» and «Input-queue allocation protocol» will discuss these issues.

Packet transmission protocol

This protocol is used when a send-packet is transmitted from one node-interface to another. When *N1* wishes to transmit a send-packet to *N2*, it puts the send-packet onto the ring, according to the *fair bandwidth allocation protocol* (see below). The packet traverses the ring and when it reaches *N2*, checked for error. If the send-packet is error-free and there is sufficient space in the input-queue of *N2*, the packet is stored in the input-queue, and *N2* immediately emits an echo-packet. This echo-packet represents the acknowledgment of the send-packet, carrying information on whether or not the send-packet was successfully received. This is called a **sub-action** and figure 2.7 shows an example. The SCI-protocol distinguishes between **request sub-action** and **response sub-action**, corresponding to requests and responses issued at a higher layer. A **transaction** consists of a request sub-

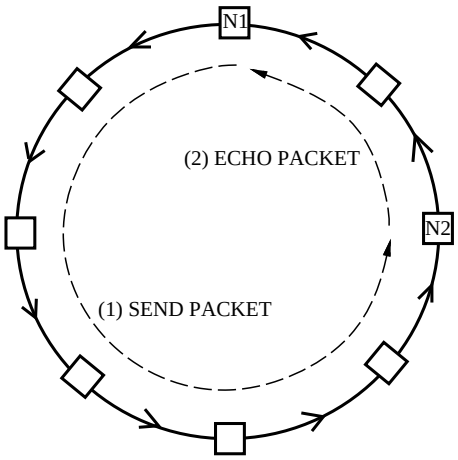


Figure 2.7: An SCI sub-action.

action and a response sub-action.

If a packet has been distorted during transmission, i.e. bit-values inside the packet have been changed because of an error, this situation is detected in the destination node and the packet is simply discarded, and no acknowledgment is returned. The source-node detects a lost packet with some kind of a timer, which initiates a signal to the cache coherence layer after a predetermined time. Notice that this also includes packets in which the destination address has been distorted, as these packets are detected by the scrubber node and discarded.

The packet transmission protocol, does not offer end-to-end acknowledgment in the case where *N1* and *N2* are on different rings. In this case the send-echo protocol is used between *N1* and the intermediate switch, and between the switch and *N2*. If the send-packet have to pass through several rings and several switches are involved, the protocol is used between the switches also. When a switch removes a packet from the ring and acknowledges it, the switch is responsible for further transmission.

Ring bandwidth allocation protocol

The main goal of rules governing ring bandwidth allocation is to make sure that each node-interface in an SCI-ring gets its fair share of ring bandwidth and in that way ensure forward progress of packet transmission. This protocol regulates the packet transmission of the node-interfaces in the same ring, and the protocol is ring-local and does not affect node-interfaces in different rings. To indicate why it is necessary to regulate the packet transmission of node-interfaces, consider the following two options on how to control the a node-interface without flow control:

Opt. 1: Always let the bypassing symbols have the right of way, and only transmit packets from the output-queue when the bypass-queue is empty.

Opt. 2: Always let the packets in the output-queue have the right of way, and temporarily store bypassing symbols in the bypass-queue.

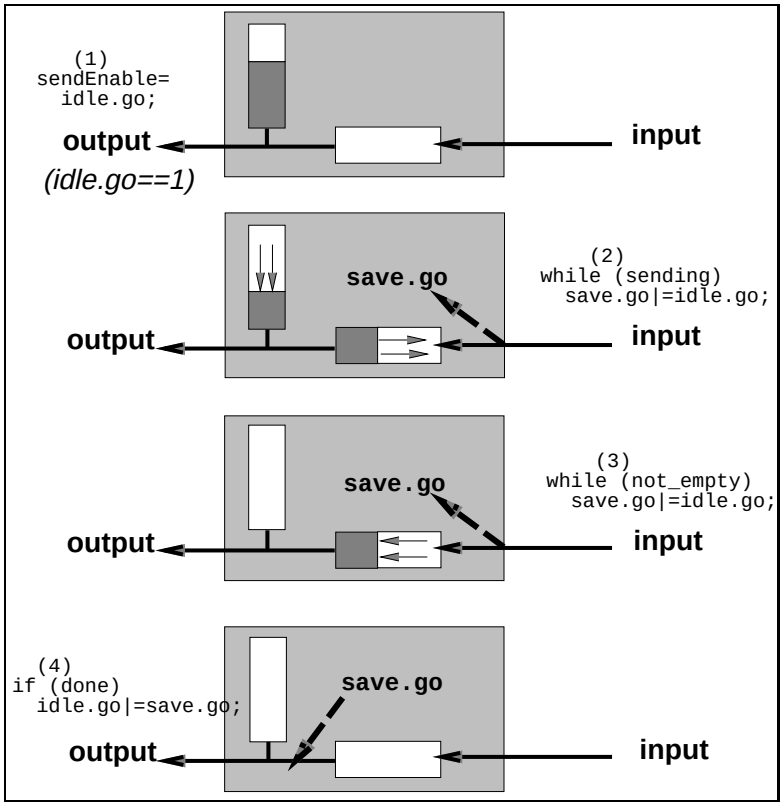


Figure 2.8: A fair SCI node-interface (reproduction of figure 3.30 in [IEEE, 1992a]).

Opt. 1 would fail to ensure a node-interface its fair share of ring bandwidth, because an upstream node-interface constantly transmitting packets would block the downstream node. The bypass-queue in the downstream node would be full at all times.

Opt. 2 is not feasible because this would require an unlimited bypass-queue. If a node-interface is constantly transmitting packets, all bypassing symbols (symbols from the node's upstream neighbour) would be stored in the bypass-queue and never leave. As a result the bypass-queue would grow indefinitely.

A node-interface according to opt. 1 is very servile whereas a node-interface according to opt. 2 is very greedy, and neither of them provide fairness and cannot ensure forward progress. The solution lay between these two extremes, and in the SCI-protocol the solution is token-based. According to this scheme a node-interface can only transmit a packet if the bypass-queue is empty and a special token passes the node-interface. When a token passes, the node-interface starts to transmit one packet from its output-queue, and simultaneously stores incoming symbols in the bypass-queue. When the node-interface is done it enters the *recovery* stage, in which the node-interface tries to empty the bypass-queue by stalling the token until the bypass-queue is emptied.

Figure 2.8 show the behavior of a node-interface which act in accordance with the rules of fair bandwidth allocation stated in the SCI-standard. A simplified model of the node-interface shown in figure 2.6 has been assumed in figure 2.8, only the bypass-queue and one output-queue is shown. The save-idle buffer is represented with a variable called **save.go**. The node-interface goes through a cycle of events similar to those shown in the figure 2.8.

The stage prior to stage (1) is not shown, in which the node-interface has an empty output-queue and simply moves packets from the input-link to the output-link without using the bypass-queue. In other words, the bypass-queue is empty prior to stage (1).

The token is an idle-symbol with the lg bit-field set (equal to 1), and referred to as a **Go-idle**. An idle-symbol with the lg bit-field reset (equal to 0) is referred to as a **NoGo-idle**.

Stage (1) describes the situation where a packet is inserted into the output-queue, and the node-interface waits for a passing Go-idle. While waiting for a Go-idle it continues to move packets from the input-link to the input-link (the bypass-queue is not used), until a Go-idles passes, whereupon it goes to stage (2). During stage (2) the packet in the output-queue is transmitted on the output-link, and at the same time packets received on the input-link is stored temporarily in the bypass-queue (unless the packets are addressed to the node-interface itself). When the transmission is done, the node-interface goes to stage (3), often referred to as the **recovery stage**, because it is during this stage that the node-interface recover from the packet transmission during stage (2). During stage (3) the node-interface sends solely from the bypass-queue, but packets may well enter the bypass-queue at the same rate as packets are leaving, so the node-interface emits only NoGo-idles between packets from the bypass-queue. The NoGo-idles stop other node-interfaces from transmitting packets from their output-queues, which in turn enables our node-interface to empty the bypass-queue. During stage (2) and (3) the node-interface stores the incoming idle-symbols, by performing an inclusive OR of the incoming idle-symbols (more precisely the lg bit-field). This imply save.go==1 if at least one Go-idle was received on the input-link during stage (2) or (3), and save.go==0 otherwise. When stage (3) ends and the node-interface goes to stage (4), the save.go value is released in an idle-symbol, and the node-interface is back to the situation prior to stage (1). A node-interface goes through these stages repeatedly, during each cycle a packet is transmitted from the output-queue.

Input-queue allocation protocol

This protocol is often referred to as the **AB-retry protocol**, and in an SCI-ring the protocol controls and if necessary restricts the access to the input-queues in the ring. Rules are needed because the input-queue of an node-interface is accessible to all other node-interfaces (every node-interface can transmit to any of the other node-interfaces) and therefore is a shared resource. The input-queue of a node-interface is limited in size, and can therefore only accommodate a limited number of packets. A packet which is addressed to a node-interface where the input-queue is full, will be rejected, and has to be retransmitted until it is accepted. Rejected packets are said to be **busied** by the destination node-interface.

A number of unfortunate situations can arise if the allocation of input-queues is left unrestricted. Consider an SCI-ring where two node-interfaces *P1* and *P2* transmit packets to node-interface C. Assume that *P1* sends a packet to C and that the packet is accepted and the input-queue of C fills up. If *P2* transmits a packet to C, the packet will be rejected and have to be retransmitted. *P1* continue to transmit packets to C, and in the meantime a packet have been removed from the input-queue in C. The packet from *P1* will be accepted and the input-queue fills up again. A packet from *P1* will again be rejected. As a result *P1* will successfully transmit all its packets, while *P2* will transmit none.

The SCI-standard define a solution to this and similar situations. Applied to the above example, this mean that C will not accept new packets if packets have been rejected pre-

VALUE	DESCRIPTION
NOTRY	Make no queue-reservation if packet is rejected
DOTRY	Make queue-reservation if packet is rejected
RETRY-A	Packet previously rejected, during state SERVE-B or SERVE-NA
RETRY-B	Packet previously rejected, during state SERVE-A or SERVE-NB

Table 2.1: Possible values of phase bit-field used in send-packets, related AB-retry

VALUE	DESCRIPTION
BUSY-N	Reserved for future use use NOTRY send-packet when retransmitting
BUSY-D	No space reserved, use DOTRY when retransmitted
BUSY-A	Space reserved, use RETRY-A when retransmitting
BUSY-B	Space reserved, use RETRY-B when retransmitting

Table 2.2: Possible values of phase bit-field used in echo-packets indicating rejected send-packets, related to AB-retry

viously. Space will eventually be created in the input-queue of C, and the rejected packets of *P2* can be accepted because space have been reserved for them. The new packets from *P1* will be rejected until all packets from *P2* (previously rejected) have been accepted.

The SCI-standard define how the send and echo-packets should indicate the various situations that can arise at the receiver and transmitter. The send and echo-packets use the **phase bit-field** in the command symbol (refer to figure 2.5) to indicate whether a send-packet was accepted or rejected. Table 2.1 and and table 2.2 show the different labels used in the AB-retry protocol, in send-packets and echo-packets respectively.

Each node-interface has two input-queues, one for requests and one for response packets. While each queue can be filled up independently, the AB-retry protocol have to be used independently for the request and response input-queue. Figure 2.9 show the AB-retry protocol as a state-change diagram, and it goes through the following cycle:

- A node-interface with a non-full input-queue accept packets as they enter, and an echo indicating a successful transmission is returned for each packet. This state is called the *SERVE-NA*-state.
- The moment a packet is rejected because the queue is full, the input-queue goes to state *SERVE-A*, and an echo is returned indicating this situation.
- An input-queue in the state *SERVE-A* will only accept the packets previously rejected in state *SERVE-NA* and *SERVE-B*. Under no circumstances will new packets be accepted even if space permits, instead they will be rejected and the echo returned will indicate this.
- When all packets that previously were rejected in state *SERVE-NA*, have been accepted, the node goes to the state *SERVE-NB*. In this state all packets, both new and other packets, will be accepted. The moment a packet is rejected because the input-queue is full, the state changes to *SERVE-B*, in which only packets previously

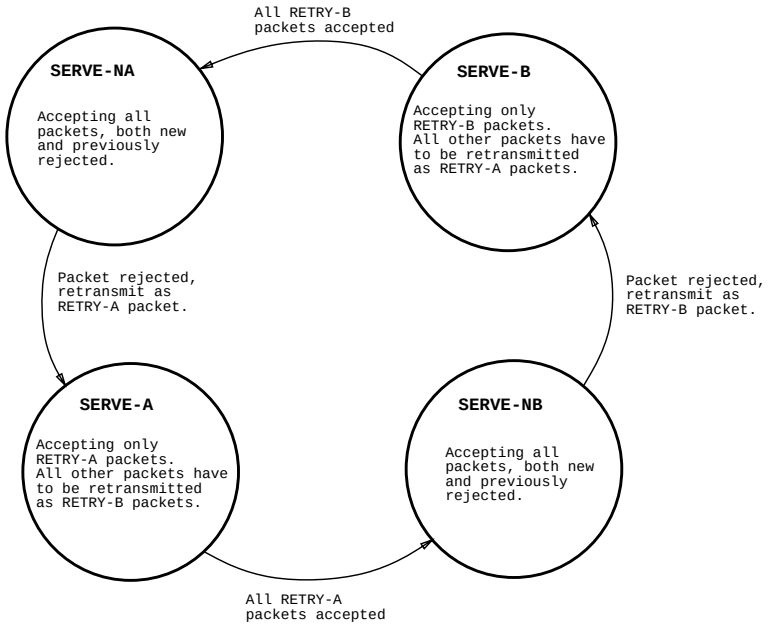


Figure 2.9: The AB-retry protocol

rejected in *SERVE-NB* and *SERVE-A* will be accepted. When all packets previously rejected in *SERVE-NB* and *SERVE-A* have been accepted, the input-queue goes to state *SERVE-NA*.

2.2.4 The physical layer (can be omitted)

The task of the physical layer is to provide services to the packet transportation layer. The services include transmission of symbols from one node-interface to the next. The physical layer is implemented in a unidirectional point-to-point link, and various links have been defined:

- Parallel electrical link - operating at 1Gbyte/sec. Used over short distances (meters)
- Serial optical link - operating at 1Gbit/sec. Used over longer distances (kilometers).
- Serial electrical link - operating at 1Gbit/sec. Used over intermediate distances. (Tens of meters)

The first alternative was chosen as the underlying link-model when the simulator was designed. This link contain 18 lines, 16 of them carry data and the remaining 2 are flag-line and clock line. The flag line is used to determine which symbols are idle-symbols and which are packet symbols. The clock line is used when the data-lines are sampled by the receiving node-interface.

2.2.5 Other concepts

Saturation: An SCI-ring is saturated when the ring is unable to transmit more packets.

Ring circumference: The circumference of an SCI-ring is equal to the number of symbols in the links and within the node-interface' bypass-queue and possible delay queue. The ring circumference depends on the traffic in the ring, and there is a lower bound and an upper bound of the ring circumference. The lower bound is determined by the latency of the links and the fixed minimum latency in the node-interfaces and occurs when the ring is lightly loaded and no symbols is stored in the bypass-queue. The upper bound is determined by the maximum packet size each node-interface can transmit (it is only during transmission of packets from the output-queue that the bypass-queue fills up) and occurs when the ring is saturated.

2.3 SCI/Real-time — modifying the SCI-protocol

This section will give an introduction to the SCI/Real-time (SCI/RT) activity.

An introduction to real-time systems in general is given in section 2.3.1 and will describe very briefly properties of real-time systems. While the SCI/RT-project intends to base its work on the theory of priority-based scheduling, an introduction to that topic is given in section 2.3.2. These sections seek to motivate the reader for the SCI/RT-modifications discussed in section 2.3.3, rather supply him/her with a complete introduction to real-time systems and priority-based scheduling.

Section 2.3.3 will describe some of the proposals on how to modify the SCI-protocol for real-time purposes. Some of these proposals are in accordance with the theory of priority based scheduling and seek to design a system which obey the requirements therein, others are not based on priority based scheduling. There is a large number of modifications proposed in relation to SCI/RT and section 2.3.3 will briefly describe some of them and emphasizing those which are investigated later in this thesis.

When the SCI-project entered the final stage in 1992 and the SCI-standard was awaiting approval, interest had already grown in using the SCI-protocol in real-time environments. Especially the Canadian Department of National Defense (Navy) was interested in the SCI-protocol and hoped to use SCI in some future naval combat systems. The SCI working group considered it more important to have the SCI-protocol approved, rather than delay the process and then try to modify SCI for real-time purposes. This activity therefore branched off into the SCI/RT working group (IEEE P1596.6) and work has progressed since then.

The formal definition of the SCI/RT project is given in the following, as stated in the draft for SCI/RT [IEEE, 1992b]:

Purpose: To define a variation of the IEEE P1596 Scalable Coherent Interface (SCI) For Real-Time Applications (SCI/RT) where the guaranteed forward progress of the SCI, given an unknown computing load, is traded for the guaranteed latency of the SCI/RT, given a known computing load.

Scope: The SCI/RT standard will encompass two changes of the SCI specification that will make it suitable for real-time applications:

1. to specify deterministic arbitration and buffer control protocols that are compatible with the priority-based scheduling theories, such as rate monotonic scheduling, and

2. to extend the error detection and correction capability to provide automatic hardware error detection and correction of a single-point hard fault, to provide automatic hardware sub action fault-retry capability, and to provide backup scrubber support.

This formal definition shows that the goal of SCI/RT is to modify the existing SCI-protocol such that priority based scheduling methodology can be used, and at the same time stay as compatible as possible to the SCI-protocol. In the SCI/RT-draft it is also stated that the future real-time systems are believed to be more dynamic than existing systems, new tasks can be added to the system dynamically by the users.

2.3.1 Real-time systems

One of the requirements to a computer system used in real-time environments, is that the computations not only have to be performed correctly, but have to meet a timing constraint also, i.e. the correctness also depends on when the results are generated. Compared to a time shared system the timing constraint is an added requirement and when a new computer system for real-time purposes is designed this requirement will influence the design process.

Another issue in real-time systems is the importance of stability and security. An example of a real-time system is a computer system monitoring and controlling a nuclear power plant. It is perhaps tolerable that a time shared system experience a breakdown once a year, but in a computer system controlling a nuclear power plant this would be totally unacceptable. In the time shared system the system can be restored to correct status using back-up routines, but in the power plant the nuclear process may come out of control. Therefore a system controlling a nuclear power plant have to avoid a break-down at all cost.

Even if the computer system worked fine, an emergency message from the nuclear kernel of an unexpected heat-up could suddenly be generated. An emergency message should reach the operator, or the process dealing with emergencies, as quickly as possible and without being delayed by other messages of less importance. Therefore priorities are assigned to messages in real-time systems, reflecting their importance.

Another example of a real-time system would the navigation system of airplanes, monitoring and controlling the avionics, and air traffic control system. Common to these examples are the critical situations which would arise if the system broke down.

In a time-shared system the important aspects are fairness among processes, forward progress, and average throughput and latency characteristics. The computer load is rarely known in advance and it also varies considerably, for example by adding or removing computers from the system. A real-time system differs in this respect, because the computer load is assumed to be fully understood and the tasks in the system are often periodical, like sampling operations of temperature, pressure etc. Once in a while an emergency situation arise, and must be handled without delay. The real-time system should be designed in a way such that the timing behavior can be analyzed and predicted. This means that given a set of tasks, it can be guaranteed the system can perform all these tasks in time, but even when the system gets overloaded, the tasks with the highest priority should meet their deadlines at the expense of the other tasks with lower priority. The system also has to be fault tolerant and robust.

2.3.2 Priority based scheduling

A task within a real-time system is usually periodical and it has to be executed and terminated within a certain deadline. It is normal to distinguish between tasks having **hard deadlines** and **soft deadlines**. A deadline is said to be hard if it has to be met in order to ensure system functionality, and it is said to be soft if it is desirable that a task meet its deadline, but it is tolerable that it misses its deadline on occasion [Liu and Layland, 1973].

A **scheduling algorithm** is a set of rules that determine which task should be executed at each instant of time. When all tasks within a real-time system are assigned a priority, indicating their importance, and several tasks are trying to gain access to a shared resource, a **priority-driven scheduling algorithm** is used when the shared resource is allocated to the task with the highest priority. If the task with the higher priority arrives while a lower priority task is using the shared resource, the lower priority task should be interrupted (or suspended), and the shared resource should be allocated to the task with the higher priority [Liu and Layland, 1973].

If the higher priority task has to wait for the lower priority task to terminate or be suspended, a **priority inversion** has occurred. Priority inversion is unavoidable in general, but should be kept at a minimum, because priority inversion reduce the overall performance of the system and its predictability in terms of timing [IEEE, 1992b].

If fixed priorities have been assigned to the tasks (i.e. the priority of a task does not change once it has been assigned), the optimal scheduling algorithm was found to be **rate monotonic** by Liu and Layland in [Liu and Layland, 1973]. According to the the theory of rate monotonic scheduling (RMS) a task is assigned a priority which depends on the length of its period. A task with a shorter period is assigned a higher priority than a task with a longer period. It is assumed that a task is ready to execute at the beginning of a period and its deadline is equal to the end of that period. A task's service-/execution-time should also be shorter than the length of the period.

The theory of rate monotonic scheduling considers only periodical tasks, but real-time systems rarely contain tasks that are purely periodical, so various techniques on how to handle aperiodic tasks have been proposed [Lin and Tarng, 1991].

The mathematical theory of RMS indicates how priorities should be assigned to tasks, how to decide whether it is possible to schedule a certain set of tasks and also how to determine the minimum number of priority levels. It is assumed that the total set of tasks is known, along with their period and execution time. Further details on RMS can be found in [Liu and Layland, 1973] and [Zalewski, 1993].

2.3.3 The proposals on how to modify SCI for real-time purposes

This section will first describe a selection of the modifications proposed in relation to SCI/RT, and then describe the packet preemption protocol in more detail because its performance has been investigated in this thesis.

Some of the proposals are made in accordance with the purpose of the SCI/RT project, while other seem to be made on another basis than RMS. The latter proposals are mostly token-based.

The following list describes a selection of the modification proposed in relation to SCI/RT:

Modifications in accordance with priority-based scheduling, [IEEE, 1992b]: Priority inversion should be avoided because it degrades the predictability of the system, and

there are two main problems we need to consider:

- The buffer overflow problem
- The blocked service problem

The **buffer overflow problem** occur in situations when a packet tries to gain access to an input-queue in a node-interface. If the queue is able to accommodate the packet, it is inserted into the queue and an echo is returned. If the input-queue is full, a priority inversion occur if there are at least one packet in the input-queue that has a lower priority than the incoming packet. To avoid priority inversion when a packet tries to gain access to an input-queue, the queue structure has to be modified.

One solution to the buffer overflow problem is to use so called **preemption**. This means that a high priority packet trying to gain access to a full input-queue will force packets with lower priority to preempt. Within basic SCI, an echo is returned immediately when the send-packet is fully received by the node-interface and it is clear that the packet can be stored in the input-queue. When packets can be preempted in the input-queue, echoes can no longer be returned in this way. Instead a *DONE-echo* is generated and returned only when the send-packet is removed from the input-queue by the application process and a *RETRY-echo* is generated and returned when a send-packet is preempted because of an incoming packet with a higher priority.

The **blocked service problem** occurs in situations when a node-interface tries to transmit packets from the output-queue and the bypass-queue in prioritized order. If the output-queue contains a packet with a higher priority than any packet in the bypass-queue, the output-queue should be served first, but if the bypass-queue is unable to accommodate an incoming packet (with length equal to the packet in the output-queue), then the bypass-queue has to be served first. As a result, priority inversion occurs because a high priority packet in the output-queue has to wait for a packet with a lower priority in the bypass-queue.

Two priority-based arbitration protocols have been proposed to solve the blocked service problem, called **packet deletion protocol** and **packet preemption protocol**. In either protocol the input-queue and bypass-queue are priority queues, and the bypass-queue has additional mechanisms so that packets can be preempted. A send-packet is preempted in the bypass-queue by replacing it with the corresponding RETRY-echo. Because send-packets are larger than echo-packets, preemption creates free space in the bypass-queue. Consequently, echo-packets will not be preempted.

Priority inheritance: Dave James [1] proposes priority inheritance and to apply this to any queue. Packets with low priority blocking a packet with a higher priority, inherit the priority of the blocked packet. In this way the queue will be emptied more quickly and the blocked packet can gain access. The priority inheritance is applied only locally in order to empty the queue, and the packets will keep their original priority after they have left the queue. [Roth, 94].

Token based schemes: Proposals in this group represent an alternative approach to SCI/RT compared to the approach of modifying the SCI-protocol to create a sys-

[1] Dave James — SCI Working Group Vice Chairman, Apple Computer, MS 301-4G, 1 Infinite Loop, Cupertino, CA 95014, USA. Email: dvj@apple.com

tem obeying the priority based scheduling theories. Common to these proposals are the use of a special token which circulate the ring, carrying priority information.

Dave James at Apple Computers and Stein Gjessing at UiO propose to use special **please**-packets. In this scheme a node is negotiating for bandwidth before it is attempting to transmit. The please packets circulate the ring in prioritized order, following an arbitration token indicating which node is allowed to transmit. [Roth, 94].

Tim Scott proposes a strategy called simplified Train Protocol for SCI/RT [Scott, 1995].

The above list describe some of the proposals made in connection with SCI/RT. After having decided to incorporate a performance analysis of some of these proposals, it became clear that it was too ambitious to investigate them all. Some of the proposals are very sketchy, and a lot of details are left out. It was therefore decided to investigate the performance of the following proposal:

Packet preemption protocol A node-interface is allowed to transmit a packet from its output-queue if the following two predicates are both true:

- Output-queue contain a packet with higher priority than any packet in the bypass-queue.
- There is sufficient free space in the bypass-queue, or there is sufficient delete-able space in the bypass-queue.

If so, the node-interface will transmit the packet in the output-queue. If at any time a packet tries to gain access to the bypass-queue, and there is not enough free space in the queue to accommodate it, a preemption operation is carried out. This operation begins with the lowest priority packet which is preempt-able, possibly the incoming packet. In other words, the node-interface will preempt packets only when necessary.

If not both of the above predicates are true, the node-interface has to transmit from the bypass-queue for a while, until these predicates become true.

2.4 Summary

This chapter has given the historical background of SCI in brief terms, given an introduction to the SCI-protocol, real-time systems in general and SCI/RT in particular. Concepts related to SCI and SCI/RT used in the remainder of the thesis are defined in this chapter.

People who previously had been involved in projects trying to increase the speed of the backplane bus (Fastbus, IEEE 960, and Futurebus, IEEE 896) abandoned the bus structure when they realized its physical limitations. As a result, the SCI-project was started in 1988 and a new approach was taken. In 1992, SCI became an IEEE-standard.

The SCI-protocol provides bus-like services to processors and memory. Multiple processors, multiple caches and multiple memory chips can be connected together and will communicate using the SCI-protocol. In this way a parallel computer with distributed shared memory is created. The SCI-protocol is layered and contain a physical layer, a packet transportation layer and a cache coherence layer, and each layer has a specified task. The packet transportation layer has been emphasized in this chapter because its performance, under various conditions, is investigated in the remainder of this thesis. Logically an SCI-interconnect consists of **rings**, either single rings or multiple rings communicating

via **switches**. A ring consists of several **nodes** which communicate using unidirectional point-to-point **links**. Nodes communicate by transmitting **packets** on the interconnect.

The SCI/RT project seeks to modify the SCI-protocol for real-time purposes. Compared to time shared systems like SCI, real-time systems have additional requirements which affect the design process. To meet these requirements, several modifications to the SCI-protocol have been proposed, either within the SCI/RT working group or outside, and some of these proposals have been described briefly in this chapter. Of these proposals, **packet preemption protocol** has been emphasized because its performance is investigated in the remainder of this thesis. At the time of writing it is still not clear how to modify the SCI-protocol for real-time purposes, and the effort of the SCI/RT working group has not yet led to an IEEE standard.

Chapter 3

Issues considered in the thesis

This chapter presents and discusses the issues considered in this thesis, or more precisely, in the remainder of this thesis. It will also be indicated how these issues are planned to be solved in the thesis.

The starting point of this chapter is the thesis' main goal, and section 3.1 will elaborate this, and briefly explain how the initial goal of the thesis was refined during the early stages.

Given the main goal of the thesis, a number of interesting issues arise, but to solve them all is too ambitious. Therefore the logical next step is to decide upon those issues which can be investigated within the framework of a Cand. Scient. thesis. Section 3.2 will present the actual issues considered in this thesis, and how it was planned to solve them.

3.1 The main goal of the thesis

The main goal of the thesis has been to design an object-oriented program in Simula, which simulates a subset of the SCI-protocol (the packet transportation layer) and which can be used when investigating the behavior and performance of this protocol subset. The simulator had to be modifiable and flexible so that modifications to the SCI-protocol itself, like the modifications required within SCI/RT, could be simulated easily by modifying the simulator. When the simulator was finished, behavior and performance of various SCI-interconnects should be analyzed, as well as the SCI/RT modifications.

This has not always been the main goal of the thesis because the very first proposal to a thesis was to design a program which simulated the SCI-protocol, and which could be used as a communication medium, offering the same interface and functionality as the real-life system it simulated. As work progressed during the early stages, it became clear that it was too laborious to design a program according to the above specification and at the same time ensure adequate efficiency within the time-frame of a Cand. Scient. thesis.

Instead only a subset of the SCI-protocol was going to be simulated and its performance and behavior investigated. The protocol subset consisted of the packet transportation layer (section 2.2.3), and simplifying assumptions had been made towards the input of the simulator. Rather than using real world programs as the main source of input, artificial input was going to be used.

Later it was decided to simulate some of the modifications proposed in relation to SCI/RT. Apart from producing results which could be interesting to the SCI/RT working-group, incorporating SCI/RT would also give an indication of how flexible and modifiable the simulator was.

3.2 Issues, and how to resolve them

Once the main goal was settled, a vast number of interesting issues arose, but to solve them all would not have been feasible, and only those issues considered in this thesis are discussed in the remainder of this section. Interesting issues can be found within the design and building process of the simulator, and within the performance of SCI and SCI/RT, and have therefore been grouped accordingly. Final conclusions on these issues are given in chapter 9.

3.2.1 Issues related to the design and building process of the simulator

Several aspects of the design and building process give raise to interesting issues. The first aspect is the programming language Simula which was given from the start because it was hoped that Simula would enhance the correctness of the simulator. The second aspect is the object-oriented programming strategy which was chosen early in the work to be the main design principle, also because it was believed that an object-oriented strategy would enhance the modifiability and flexibility.

The following questions express the issues which are considered in this thesis and which are related to the design and building process of the simulator:

Issue 1: Is it possible to design a simulator for the SCI-protocol which is flexible and modifiable, so that future modifications to the SCI-protocol can be simulated without extensive re-design?

The task of designing a program which simulates both the SCI-protocol and various modifications to the SCI-protocol proposed in relation to SCI/RT may lead to experience that give an indication whether this is possible.

Issue 2: How successful is the object-oriented programming strategy when simulating SCI, when modifications of the SCI-protocol are simulated also?

Incorporating SCI/RT in the SCI-simulator would indicate an answer to the above question. An object-oriented program contains modules whose implementation are hidden to the rest of the program, so a modification of the module structure may not imply extensive re-design. On the other hand, if the underlying model is changed, new entities are introduced, and modifying the program may not be as simple.

Issue 3: How does the Simula programming language affect the design process of the SCI-simulator in general, and the object-oriented programming strategy in particular?

Simula is a programming language which is known to support object-oriented programming. The `class` construct is used when abstract data types are implemented whose implementation should be hidden from direct manipulation, and can be accessed only through a well-defined interface. On the other hand, Simula programs are seldom fast, and long simulations may be needed to ensure reliable simulation results.

3.2.2 Issues related to the performance of SCI

The number of issues related to the performance of SCI are large, so it was decided to focus upon single SCI-rings and a small selection of multi SCI-ring interconnects. When work be-

gan, results had already been published on related aspects, among them [Scott et.al., 1992], and it was decided to compare the results in the latter article to results achieved using the simulator designed in this thesis.

The following questions express the issues which are considered in this thesis and which are related to the performance of SCI:

Issue 4: What affects the performance of single SCI-rings, with less than 16 nodes?

On one hand, an SCI-ring allow multiple nodes to be active and simultaneously transmit packets, while on the other hand, a packet may affect multiple nodes because worm-hole routing is used and as a result several nodes can be blocked. Therefore it is difficult to predict the performance of single SCI-rings.

To find an answer to this question, single SCI-rings will be simulated under various conditions. The exact parameter-values assumed in these simulations are presented and discussed in chapter 5, and the simulation-results itself, related to Issue 4, are presented in chapter 6.

Issue 5: Is the SCI-ring scalable, when the number of nodes are less than 16?

This question is closely related to Issue 4, because the ring allow multiple nodes to transmit, but a packet may block multiple nodes, which leaves us with uncertainty regarding the concurrency of the ring. If the SCI-ring was scalable we would expect that the total throughput of a 16 node SCI-ring would be significantly higher than that of a 4-node SCI-ring.

To find an answer to this question, single SCI-rings of size 4 and 16 will be simulated. The exact parameter-values assumed in these simulations are presented and discussed in chapter 5, and the simulation-results itself, related to Issue 5, are presented in chapter 6.

Issue 6: Does the flow control mechanism specified in the SCI-protocol ensure fairness among the nodes?

It is not obvious that the flow control mechanism described in section 2.2.3, really ensure fairness. There is no direct feedback which throttles a node blocking another node, except for the Go-idles. On the other hand, when a node is in recovery stage it stalls the Go-idles from further progress, and it is reasonable that other nodes stop transmitting once the Go-idles are removed from the ring.

To find an answer to this question, SCI-rings of size 4 and 16 will be simulated under the assumption of a non-uniform load and traffic pattern, and a possible difference in behavior can be observe by using flow control or no flow control. The exact parameter-values assumed in these simulations are presented and discussed in chapter 5, and the simulation-results itself, related to Issue 6, are presented in chapter 6.

Issue 7: Are the results achieved using the simulator developed in this thesis comparable to results in [Scott et.al., 1992]?

Some uncertainty is always associated with results achieved through simulation. Errors can occur on various levels, from the program-code level up to the logical level where the complexity of the real-world problem perhaps is not fully understood. It

would strengthen our confidence in the simulator if the results achieved through this simulator are comparable to other people's results.

To find an answer to this question, SCI-rings will be simulated under conditions which resemble those assumed in [Scott et.al., 1992]. The exact parameter-values assumed in these simulations are presented and discussed in chapter 5, and the simulation-results itself, related to Issue 7, are presented in chapter 6.

Issue 8: Is there a better alternative than using a single SCI-ring interconnect if we were going to connect 16 nodes?

A single SCI-ring may not be the ideal communication medium for 16 nodes. An interconnect with multiple rings provide alternative paths and may increase the throughput. To answer the above question an interconnect consisting of four rings will be simulated and compared to a single SCI-ring. The exact parameter-values assumed in these simulations are presented and discussed in chapter 5, and the simulation-results itself, related to Issue 8, are presented in chapter 7.

3.2.3 Issues related to the performance of SCI/RT

As it was indicated in section 2.3, SCI/RT is still a project in progress, and the effort of the SCI/RT working group has not yet led to an IEEE standard. The modifications to the SCI-protocol proposed are sometimes sketchy, and important details are often left out. Therefore is was decided to investigate modifications proposed in the draft [IEEE, 1992b], and simulate priority output-queue and preemptive bypass-queue. Even though the draft is old, and may no longer represent the current state of the SCI/RT working-group, the draft is still quite detailed and therefore simpler to implement in the simulator. It is also interesting to simulate these modifications because some of the current proposals are based on them.

The following question express the issue which is considered in this thesis and which are related to the performance of SCI/RT:

Issue 9: Does the packet preemption protocol in combination with priority output-queue and preemptive bypass-queue meet the requirements stated in the formal definition of the SCI/RT project (refer to 2.3)?

The packet preemption protocol in combination with priority output-queue and preemptive bypass-queue are only one of several modifications proposed in the draft [IEEE, 1992b], and it is not likely that this modification alone will ensure a system suitable in real-time environments. Nonetheless it is reasonable that this modifications should have the properties which lead to the following kind of results:

1. When load is low, the priority distribution of packets transmitted *and* acknowledged by the receiving node should be approximately equal to the priority distribution of new packets. This mean that all packets, regardless of priority, should reach their destination when load is low and low priority packets should not be preempted if there bandwidth is available.
2. When load is higher, the priority distribution of send-packets transmitted *and* acknowledged by the receiving node should be approximately equal to the priority distribution of new send-packets *only for the highest priority levels*. This

mean that the higher priority packets should reach their destination at the expense of lower priority packets when load is high.

3. The latency of high priority traffic should be less than the latency of low priority traffic, both in general and in overload situations.

General tendencies and trends will be focused rather than exact quantitative measurements. To answer the above question, an SCI-ring of size 4 will be simulated, where each node use the packet preemption protocol in combination with priority output-queues and a preemptive priority bypass-queue. The exact parameter-values assumed in these simulations are presented and discussed in chapter 5, and the simulation-results itself, related to Issue 9, are presented in chapter 8.

3.3 Summary

This chapter has presented and discussed the main goal of the thesis and related issues which have been considered in this thesis.

The main goal of the thesis has been to design a modifiable and flexible simulator for a subset of the SCI-protocol (packet transportation layer) using the programming language Simula, to incorporate some of the SCI/RT modifications proposed by the SCI/RT working group and finally to use the simulator to investigate the performance of certain aspects of SCI and SCI/RT.

A number of interesting issues arise in relation to the above goal, but within the time frame of a Cand. Scient. thesis, these had to be limited in number. Nine issue has been specified in this chapter, expressed as questions, and will be referred to as **Issue 1 - Issue 9** in the remainder of the thesis.

Issue 1 - Issue 3 are related to the design process of the simulator and asks whether it is indeed possible to design a simulator for the SCI-protocol (packet transportation layer) being both modifiable and flexible, and so that future modifications to the protocol (e.g. SCI/RT) can be simulated without extensive redesign. Furthermore these issues ask how successful the object-oriented programming strategy is when designing a simulator for SCI and how Simula affects the design process in general and the object-oriented programming strategy in particular, when designing a SCI-simulator.

Issue 4 - Issue 8 are related to the performance of SCI and wants to know what things affect the performance of single SCI-rings, whether the SCI-ring is scalable, whether the SCI flow control mechanism ensure fairness to the nodes in an SCI-ring, whether the SCI-simulator designed in this thesis produce results which are comparable to those presented in [Scott et.al., 1992] and finally whether there are better ways to connect 16 nodes than using a single ring.

Issue 9 is related to SCI/RT and asks whether an SCI-ring using the packet preemption protocol in combination with priority output-queues and preemptive priority bypass-queues meet the requirements of SCI/RT project, as stated in the definition.

[This page has been intentionally left blank]

Chapter 4

Designing and building the simulator

This chapter describes the design and building process of the program simulating the SCI-protocol, a simulator developed in relation to this thesis and in accordance to the main goal of the thesis as described in section 3.1.

Not being familiar with the SCI-protocol, the first task facing the author was to understand the SCI-protocol to a level where a program simulating it could be designed. Section 4.1 will describe the various sources of information on which the understanding of SCI was based.

The logical next step in the design process, after having understood SCI and various related aspects, would simply be to simulate it, but a strategy is needed. The reason is that the simulator is going to be used in performance analysis of SCI, and to be successful in this intention, the simulator has to meet certain requirements, and it is because of these requirements that a strategy is needed. In the main goal of the thesis (section 3.1) the simulator was required to be flexible and modifiable, but there are additional requirements also. Section 4.2 will explain which requirements the simulator has to meet, which strategy is used and why the strategy is expected to help design a simulator meeting these requirements.

The historical development of the simulator and the final version of the simulator in particular, is described in section 4.3. The implementation of the final version is shown, because it indicates how the strategy was expressed in program code. It is hoped that this section may be of interest to others who also work on SCI-simulations, not because the simulator described there is believed to represent the optimal solution, but because it presents an alternative from which better solutions can be designed.

4.1 Sources of information

When understanding the SCI-protocol, the main source of information has been the SCI-standard [IEEE, 1992a], published by Institute of Electrical and Electronic Engineers (IEEE). An IEEE-standard is developed by a Technical Committee of the IEEE Societies and a Standards Coordinating Committee of the IEEE Standards board, consisting of people who volunteer to participate. A standard begins as a draft, and after having been **approved**, becomes an IEEE-standard. IEEE publishes documents with the intention of presenting state of the art technology as standards, but an IEEE standard does not exclude alternative

approaches. Every five years an IEEE standard will have to be reaffirmed, and a standard more than five years old, not being reaffirmed no longer represent the state of the art within its scope. In the case of SCI, the draft was approved in 1992, and therefore reaffirmation has not been in question yet.

As already explained in section 2.2, the SCI-standard is divided into two parts. The first part is an introduction and tutorial to SCI, and the second part is the SCI-protocol formally specified in C-code. To a non-expert the SCI-standard is both voluminous and complex, and therefore the tutorial part was approached first and has been referred the most. The C-code part was consulted when the tutorial was unclear or ambiguous.

Various articles published regarding SCI have provided an alternative approach to SCI. Articles by [Picker et.al., 1994], [Scott et.al., 1992] and [Bothner and Hulaas, 1991] have been useful.

To understand the intention of the SCI/RT working group and its attempts to modify the SCI-protocol for real-time purposes, the main source of information has been the SCI/RT draft [IEEE, 1992b]. The SCI/RT-draft was written in fall 1992 and has not been altered since. Therefore the draft may no longer represent the current consensus of the SCI/RT working group. The document still provide useful information though, because it gives a brief introduction to real-time systems and priority based scheduling, and because much of the current activity within SCI/RT are based on priority based scheduling (In particular rate monotonic scheduling, refer to section 2.3.2).

Various articles has also provided an alternative approach to real-time systems, in particular [Zalewski, 1993].

Other sources of information have been various mailing lists like `sci@sunrise.scu.edu` and `sci_rt@sunrise.scu.edu`. Especially the latter mailing-list concerning SCI/RT, has brought useful suggestions, comments and proposals. The mailing lists have contained both good and bad information — good because recent discoveries become quickly are visible and many proposals are presented — bad because some of the proposals were very sketchy and a matter of opinion.

During the spring semester of 1994, seminars were held at regular intervals at UiO, either the Department of Informatics or the Department of Physics, where issues related to SCI were presented and discussed. During these seminar-meetings, people from various organizations presented their current work or latest discoveries. Even though these seminars did not present issues which were directly useful in the thesis, it led to a broader understanding of the SCI-technology.

An international SCI-conference were held at UiO 20-23 Sept 1994, where projects and recent discoveries were presented by people from various organizations. Especially the SCI/RT workshop held during the conference, presented issues of particular interest.

When designing the SCI-simulator the books by [Birtwistle et.al., 1982] and [Kirkerud, 1989] have been particularly useful, the latter can also serve as an introduction to object-oriented programming in Simula.

4.2 The programming strategy

This section will describe the strategy which were used when the program simulating SCI was designed. Section 4.2.1 will explain why a strategy was needed, section 4.2.2 will describe the strategy itself and section 4.2.3 will explain why the strategy was chosen. How the strategy finds support in Simula is explained briefly in section 4.2.4. An alternative to

the strategy used in this thesis is described in section 4.2.5.

4.2.1 Requirements to the simulator - a strategy is needed

A strategy was needed because the simulator had to meet certain requirements in order to be useful in performance analysis. In the following, these requirements will be discussed.

As explained in chapter 3, the SCI-simulator had to be both **modifiable and flexible**. In this way modifications of the SCI-protocol related to SCI/RT and various interconnect topologies could be simulated.

The simulator also had to meet a requirement towards **correctness**, because the simulator should simulate SCI-interconnects, and produce results which should apply to real-world SCI-interconnects also. If the simulator should seek to be 100% correct, every possible detail and aspect had to be simulated, and a formal verification of the whole program had to be performed. This was considered too much work within the framework of a Master thesis.

The correctness requirement was therefore relaxed slightly, because a simulator being flexible, modifiable *and* correct would represent too much work. After a closer look at the simulations which were going to be performed later, it became clear that some details could be left out altogether if a simplified model of the real-world was assumed. It was decided that the simulator had to be correct under these assumptions. It was also important that these assumptions, which were made prior to the design of the simulator, were reasonable. For example, a reasonable assumptions would be that a parallel SCI-link with 18 parallel wires never experience skew (Skew is caused by differences in wire-lengths in a parallel link, where a signal on the shorter wire propagate faster than a signal on the longer wire). On the other hand, assuming an SCI-ring without the queue allocation protocol (refer to section 2.2.3) would not be reasonable because it was expected that the queue allocation protocol would come into action during the simulations. Even after having made these simplifying assumptions, a full formal verification of the program was not considered feasible, and an informal reasoning was considered a better alternative.

The requirements toward modifiability and flexibility also implied that the simulator had to be **parameterized**. As described in chapter 3, various ring sizes, topologies and traffic patterns was going to be simulated, and the simulator had to simulate these conditions as precisely as possible. It was also considered important that the parameters specifying a simulation, resembled the characteristics of the corresponding real-life system, e.g. that queue-sizes were specified in bytes.

Another important requirement was that **performance analysis could be made**, which meant that the simulator had to output information concerning behavior and performance of the simulated system. It was also considered important that the statistical measurements was defined in a way that resembled those of the real-world system, e.g. that the points of measurements in the simulator could be recognized in a real-world system. Furthermore, the simulator should be able to provide information indicating the statistical reliability of the measurements. For example, if we wished to determine the average latency of packets in a given interconnect, the sample mean is a statistic which enable us to draw inferences on the population mean, but the sampled mean is only a point estimate, and does not indicate variance of the sampled values. The sample mean should therefore be supplied with the standard error or a confidence interval [Bhattacharyya and Johnson, 1977]. In a book by Jain [Jain, 1991] regarding computer system performance analysis, the usefulness of confidence intervals is emphasized in a discussion on how to compare systems using

sampled data:

> *The basic idea is that a definite statement cannot be made about the characteristics of all systems, but a probabilistic statement about the range in which the characteristics of most system would lie can be made.* - Raj Jain

The last requirement which was considered important was the **efficiency** requirement. If average measurements of throughput and latency were sought, a better estimate of the average value would be produced using longer simulations. Reliable results should therefore be available within reasonable time.

To summarize - the following requirements were considered important when the simulator was designed:

- Modifiable and flexible
- Correct
- Parameterized
- Enabling performance analysis
- Efficient

4.2.2 The design strategy

This section will describe the strategy that has been used when the simulator related to this thesis was designed. In section 4.2.1 the various requirements toward the simulator was described, and consequently a strategy was needed to guide the design process.

Before the strategy is presented, some concepts will have to be defined:

Class: In a program, a class refers to a module whose inner structure is invisible to, and cannot be accessible directly from the remainder of the program. A class can only be accessed through procedures or actions which belong to the interface of the class. The implementation of the procedures in the interface of a class is invisible to the remainder of the program.

Object: An object is an instance of a class, and is generated dynamically. An object resemble the class from which it has been created, because the object's inner structure and implementation is invisible to other objects, and can only be accessed through procedures in the interface of the object. There can also be several objects created from the same class. The behavior of an object is specified by the corresponding class and its behavior should be visible to other objects only through the interface.

The overall strategy has been to design classes, from which objects can be created dynamically and which in structure and behavior resembles the real world components they represent and simulate. By combining several objects of various classes, complex structures can be created. For example by designing classes representing the node-interface and a class representing the link, objects of these types can be created dynamically and then combined into the more complex structure of an SCI-ring. This structure representing an SCI-ring will simulate a real world SCI-ring by activating the objects it contain, and

let the objects interact according to their corresponding class. Various structures can be created by creating various number of objects, and by combining them differently.

If the class is parameterized, objects can be more closely specified, and variation of the objects is ensured. For example, if a class representing a node-interface is parameterized and the bypass-queue size can be specified, different node-interface objects can be created with different bypass-queue size. Further flexibility can be achieved if the class is a sub-type hierarchy, where a super-type have several sub-types. For example, if the class for bypass-queue contain two sub-types, one for FIFO-queue and one for priority queue, bypass-queue objects can be created with different behavior, but with the same interface.

In other words, the strategy seeks to classify all entities in a real-world SCI-ring, and design classes according to this classification. A class should resemble the corresponding entity-class in structure and behavior, and objects are generated from each class so that each object represents an entity in a real-world SCI-ring. Object can be combined into more complex structures afterwards.

The strategy above is an **object-oriented programming strategy**, but have an added requirement towards the implementation of the class because the implementation should resemble the structure of the real world counterpart. The *first* object-oriented programming language was Simula [Wegner, 1990] and it developed during the 1960's by people at Norwegian Computing Center. It contained language-constructs representing classes, sub-classes (with inheritance) and objects [Dahl et.al., 1982].

No formal definition of object-orientation has yet been made, but the following quote (taken from [Wegner, 1990]) gives an indication of what is commonly associated with object-orientation:

> *Modeling entities by their behavior (their response to messages) is a central principle of scientific method in many disciplines: behaviorism in psychology, operationalism in physics, and Platonic ideals in philosophy. Objects are a canonical form of description for any discipline or domain of discourse. Its universality as a representation, modeling, and abstraction technique supports the view that the object-oriented paradigm is conceptually and computationally fundamental* - Peter Wegner

A consequence of object-oriented programming is that programs contain modules (or classes) which hide their complexity and implementation. The implementation of one class can be changed without changing the other classes, and the implementation of one class can be understood without having to understand the implementation of the other classes. In larger programs this benefits modifiability and ease of understanding.

Object-oriented programming is traditionally closely related to a top-down programming strategy. In this strategy the programmer begins with a class which solve the whole problem. If the problem is complex, implementing the class solving the problem is equally complex, but the task is made manageable by identifying sub-problems and divide the implementation into smaller parts. As a result the complexity is reduced, and for each task a class is designed. The process can be repeated several times, but terminates when the implementation of a class is easily expressed in program code.

In practical work on the SCI-protocol, the top-down strategy had to be combined with a bottom-up strategy. A top-down strategy will not necessarily lead to a set of class-definitions which in structure and behavior resemble their real-world counterpart, as required. Moreover the complexity of the SCI-protocol force the programmer to go into details before a final decision can be made on which classes to design.

A combination of the top-down and bottom-up strategies has therefore been used in order to decide which classes the simulator should contain.

4.2.3 Reasons for choosing the strategy

This section will explain why the strategy described in section 4.2.2 is expected to help designing a simulator which meet the requirements in section 4.2.1.

The requirement concerning **modifiability** and **flexibility** is expected to be met because the implementation of a class is hidden from the rest of the program and can be modified without affecting other classes in the rest of the program. If the inner structure of a class has to be modified, it should be possible to perform these modifications without changing the interface of the data type. If it really should be necessary to change the interface of a class, those parts of the program which access objects of this type have to be modified, which is a laborious task. Nevertheless it is hoped that in a well-structured program the number of object accesses is bounded in number and space.

When a set of of classes have been defined, it should be fairly easy to combine them into more complex structures, by creating objects dynamically. The user could specify the desired structure, and the simulator could create the structure dynamically according to the specification. In this way the simulator becomes **parameterized**.

The strategy is also expected to enhance **correctness** because it is easier to reason about the simulator's correctness when the structure is recognized as similar to the real world structure, and because the real-world structure guides the program-design directly. If the correctness of a class is established, which mean that the class behaves according to its specification when accessed through the interface, it is also known, once and for all, that objects of this type maintain their consistency during simulation. A class is protected from direct manipulation and accessible only through interface procedures.

When the structure and behavior of objects correspond to their real world counterparts, it is also easier to **perform and present statistical measurements** that will be comparable to real-world measurements. To illustrate this point consider the following example: Throughput (number of bytes per unit time), can be measured at various places in an SCI-ring. On the links there are a stream of idle symbols and packet symbols, and the throughput can be calculated using either all symbols or only one of the two variants. The throughput can also be measured at a node-interface, using all packets addressed to it or only those packets which are inserted into the input-queue (Packets that are addressed to a node-interface could either be inserted into the queue or rejected, depending on whether the queue can accommodate the packet or not). All these throughput measurements are different and equally interesting, and should be calculated as precisely as possible. This is expected to easier when the structure of the objects resemble the structure of the real world entities they represent.

So far, the strategy is expected to help designing a simulator which meet the requirement in section 4.2.1. It is less certain that the requirement towards **efficiency** is met, because experience indicate that object-oriented programs create a vast number of dynamic objects and use the garbage collector frequently. This will reduce the efficiency of the program.

4.2.4 How the strategy is supported in Simula

The `class` construct in Simula can be used in order to group together a data-structure and the procedures manipulating it. Some Simula-compilers support a mechanism which ensure

that classes are accessed only through their interface, but this has not been the case with the compiler used in this thesis. A Simula class can be parameterized and class-hierarchies can be defined. In this respect the strategy finds support in Simula.

Programs written in Simula are usually not very fast, and this is caused partly by a slow run-time system, and partly by the fact that Simula encourages object-oriented programming, and this will, inevitably, lead to dynamically allocated objects.

4.2.5 An alternative strategy

An alternative to the programming strategy described in the previous sections, was suggested by Stein Gjessing at UiO in the early stage of the work. This alternative strategy focus upon the packet entity in the SCI-interconnect and it is seeks to represent and simulate the life of the packet, rather than the whole interconnect. A packet is represented by an event, and a packet-event is scheduled and rescheduled in a way which resembled the life of the packets in the real-world SCI-interconnect. For example, when a packet pass a node-interface, this could be simulated by rescheduling the packet-event to let it become active N nanoseconds later, where N is the delay in the node-interface.

One advantage of this alternative strategy is that it focuses upon the entity which give raise to throughput and latency, and disregard the rest. In this way we may get less code and a faster simulator. However, it is not obvious how to ensure correctness and the risk is there that we make restricting assumptions toward the issues which we eventually are going to investigate. At the time when the strategy was suggested, a considerable effort was still involved in *understanding* the SCI-protocol and it was reasonable to believe that the strategy required a thorough understanding of the SCI-protocol before a program could be designed. In other words, programming would have to be postponed until the SCI-protocol was fully understood. It was therefore decided to abandon the alternative strategy in favor of the strategy described in section 4.2.2.

4.3 Implementation of the final simulator

This section will describe briefly the development of the simulator, and in particular describe and discuss the final version. The latter part will indicate how the strategy described earlier in section 4.2 was expressed program-code.

4.3.1 The historical development of the simulator

The biggest challenge in the design process was the fact that the SCI-protocol describe the expected behavior, not so much how it should be implemented. A programmer who designs a simulator for SCI has to make a choice on these implementation issues, and has to realize the consequences because they will affect the behavior of the simulator. A good understanding of the SCI-protocol was required especially when faced with different alternatives.

As an example, consider the task of simulating the output-queue structure. In the output-queue there will be various types of packets, *unsent* packets, packets which have to be *retransmitted* and packets sent which *await echoes*. The order in which these packets leave the output-queue cannot be chosen arbitrarily, because forward progress must be ensured. Retry packets (packets which have to be retransmitted) should possibly be transmitted before unsent packets, if an unsent packet is transmitted to the node which

rejected a packet previously, it would certainly be rejected. A packet which awaits an echo should not be retransmitted at all, before an echo is received, but what if the echo is lost in the ring? The example shows that the behavior of an SCI-ring has to be well understood in order to simulate the protocol correctly.

During **stage one**, a very simple simulator was designed, and it was neither correct nor complete, but only used as a tool in order to understand the basic mechanisms in Simula used when simulating parallel processes. The stage one simulator only simulates transmission of complete packets and nothing else. It was clear that the a node-interface should be considered an independent process, and this could be accomplished by using the `class simulation` package in Simula.

The next step was to simulate the packet concept in more detail, and in **stage two** packets were simulated by symbol sequences, and this simulator as well as the succeeding simulators, track every symbol around the ring. On this stage the interesting issue was how to ensure correct receiving of symbols while symbols were simultaneously transmitted. It was realized that a correct way to simulate this would be to design one receiver stage and one transmitter-stage, which behaved independently. On this stage packets and symbols were classes.

In the following stage, considerable effort was involved when the ring bandwidth allocation protocol and the queue allocation protocol were going to be simulated. This effort led to the **stage three** simulator, which implements the go-bit protocol and AB-retry protocol. On this stage, the simulator contain modules which simulated the building blocks of the SCI-interconnect - node, node-interface and link. These entities were represented fairly completely by the corresponding classes.

The strategy described in 4.2 had up to this point been followed diligently, and the simulator had so far been designed without considering the efficiency of the simulator. Nevertheless, when the simulator were tested it became clear that something had to done to speed up the simulator, because the efficiency was appalling. Tests which simulated approximately 1500ns of real system time, spent as much as 80-90% of the time running the garbage collector, and the tendency was clear - the longer the simulation, the slower the execution. Occasionally the program terminated abnormly, with a message from the run-time system telling that no storage was freed by the garbage collector.

The low efficiency of the simulator during stage three was not acceptable, and it was decided to do something about it. As explained above, run-time statistics indicate that the simulator spent as much as 90% of the time running the garbage collector, so attention was drawn to those parts of the simulator which generated most of the objects. This turned out to be the classes representing the packet entity, the multiplexer entity and the stripper entity. More precisely the culprit was the class representing the symbol entity, because a vast number of new symbol objects were created during run time. This was an awkward situation, because those classes which previously had been designed just to simplify the future design process, were now being scrutinized.

This led to **stage four** simulator, which differs from the stage three simulator by simulating symbols by integers (symbols had previously been class-objects) and that the packet class and bypass-queue class were modified. Simulating symbol entities with integers were not entirely according to the strategy, and it was expected that the modifications compared to stage three would reduce the readability and the modularity of the simulator. To some extent, this expectation was later confirmed, in particular when statistical measurements of the latency of packets were implemented. Packet objects had to carry time-stamps and because a packet was a sequence of integers, time-stamps had to stored in *bit-fields* of sev-

eral integers. Nevertheless it became apparent that the modifications during stage 4 were a remedy that increased the efficiency. Comparing test simulations to those previously performed on stage three indicated that efficiency had increased by a factor of 100, sometimes even more. The stage four simulator retained the same functionality as stage three simulator, but garbage collection were drastically reduced.

Having designed the classes representing the building blocks in real-life SCI-interconnects, their structure and behavior, the next step was to enable the user to specify arbitrary SCI-interconnects. A specification language and a parser for the language was designed. The parser would read the specification file and according to this specification file create dynamically the necessary objects of the classes `node`, `nodeinterface` and `link`. These actions were carried out during **stage five**, leading up to the final version of the simulator (refer to section 4.3.2 for more details). Some of the effort was also focused upon statistical aspects regarding the measurements performed in the simulator. One of the requirements towards the simulator was that statistical measurements could be performed and that they corresponded closely to those which could be performed in a real-world SCI-interconnect. It was also important that the simulator gave an indication on how exact the measurements were. Point-estimates (mean value) should therefore be supplied with interval-estimates. The concept of sampling gave the idea to class and objects of this class were assigned to each interesting statistical measurement. In this way the mean, standard deviation and confidence intervals were calculated each time a sampling-operation was performed.

It was also during this stage that SCI/RT was incorporated in the simulator. Appropriate subclasses to the existing classes representing the bypass-queue and output-queue were designed in order to cater for the different behavior and internal structure required by SCI/RT.

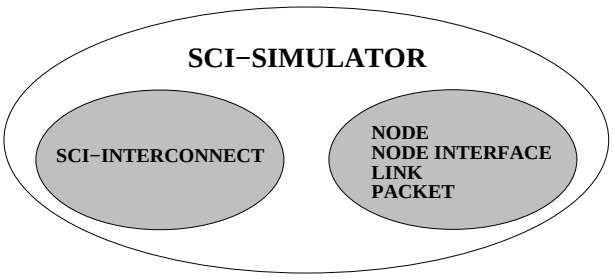
4.3.2 SCIsim - the final version

The program which has been designed for SCI is a program that contain a number of class definitions. The classes can roughly be divided into two sets:

- The first set contain classes representing **structural concepts** like ring and interconnect, and their task is to create the structure dynamically and store the structure.
- The second set contain classes representing **real-world entities** which are the building blocks of SCI-interconnects, for example node, node-interface and link. These classes have been designed according to the strategy described in section 4.2.

Briefly the program behaves as follows: The user writes a file containing the specification of the SCI-interconnect that is going to be simulated. The simulator, henceforth referred to by its name "SCIsim", will let the objects representing the structure (First set) read the specification file, and dynamically create the object accordingly. These latter objects belong to the second set, and they simulate the physical entities of an SCI-interconnects. If the specification file is found correct, the created structure is used in simulation, which is started immediately. During simulation, statistical information is gathered and calculated. Once the simulation is over, and terminates successfully, statistical information is written to file, along with the current state of the interconnect.

An unsuccessful simulation is recognized by an error message, usually caused by unfortunate parameter settings, e.g. specifying an 80 byte bypass-queue inside a node-interface which is capable to transmitting packets up to 256 bytes.



(a) The concept

```
simulation class SCIsimulator(specfile);
  ref(inputfile) specfile;
begin

% LOCAL DATA-TYPE DEFINITIONS:

  element class packet <..>;
  class link <..>;
  process class nodeinterface <..>;
  process class node <..>;

  class network <..>;

% DATA STRUCTURE:

  ref(network) SCIinterconnect;

% DEFINITIONS OF LOCAL PROCEDURES:

  procedure initialize <..>;

% DEFINITIONS OF ACCESSIBLE PROCEDURES:

  procedure WriteSimulationInfo <..>;
  procedure WriteStatisticalInfo <..>;


% STATEMENTS:

  initialize;
  <..>;

end;
```

(b) The class

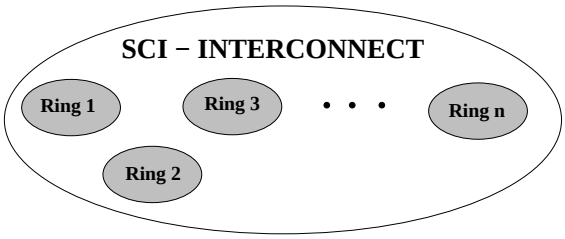
Figure 4.1: The SCI-simulator

The program is textually divided into several smaller files so that each file contain a class or a set of procedures. Classes in Simula can be compiled separately in general, but this is not possible when one class depends on another class' definition *and* vice versa. In the SCIsim simulator there are several classes which are mutually dependent, so the smaller files has to merged into one large file before compilation (Using the C-preprocessor [Kernighan and Ritchie, 1988]).

In the remainder of this section, part of the implementation of the program will be described in some detail, starting at the uppermost level of abstraction with the concept of the SCI-simulator, and proceeding onto modules representing structural entities and building blocks of SCI-interconnects. To indicate how the strategy described in section 4.2 influenced the program structure, a two-part presentation will be given. The first part will describe the real world concept or entity, and the second part will describe the textual representation of this concept or entity in the program. The textual representation is simplified compared to the version used in the program, showing the main structure only. Lines in the textual representation which are comments, begin with '%'.

The SCI-simulator — figure 4.1

Starting at the uppermost level of abstraction we have the **SCI-simulator concept**, and it contains other concepts, like interconnect, link and node. In the SCIsim program the concept of **SCI-simulator** is represented by the `class SCIsimulator`, which is a subclass of the `class simulation`, a standard Simula library class. The latter class enables simple process-simulation in the `class SCIsimulator`.



(a) The concept

```
class network(specfile);
  ref(inputfile) specfile;
begin

% LOCAL DATA-TYPE DEFINITIONS:

  class ring <..>;
  class ring_set <..>;

% DATA STRUCTURE:

  ref(ring_set) RingSet;

% DEFINITIONS OF LOCAL PROCEDURES:

  procedure Initialize <..>;

% DEFINITIONS OF ACCESSIBLE PROCEDURES:

  procedure DisplaySimulation <..>;
  procedure DisplayStatistics <..>;


% STATEMENTS:

  initialize;

end;
```

(b) The class

Figure 4.2: The SCI-interconnect

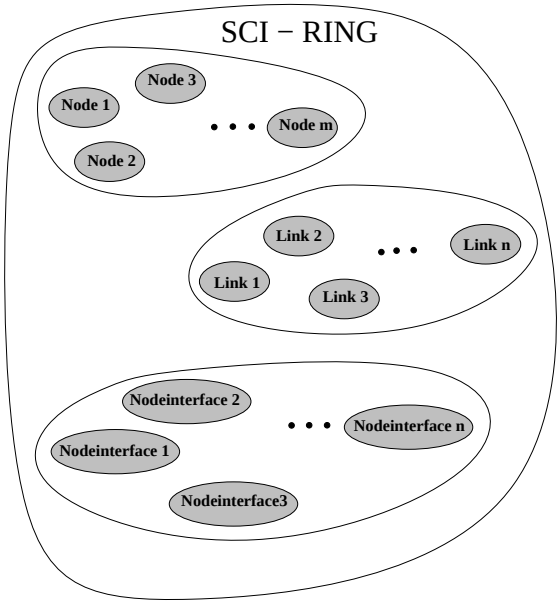
Figure 4.1a illustrates the idea graphically, and shows the concept **SCI-simulator** encapsulating other concepts, like interconnect, link and node. Figure 4.1b shows the textual representation of the `class SCIsimulator` which is used in the program. This class contains two different sets of classes, representing either structural concepts (e.g. `class interconnect`) or physical entities (e.g. `class node`).

The structure of the `class SCIsimulator` is an SCI-interconnect, and its behavior is to open the specification file, pass it onto a generated `interconnect` object and start the simulation. Once the simulation is done, it will write statistical measurements and simulation status-information to file.

The SCI-interconnect — figure 4.2

The **SCI-interconnect concept** relates to the real world SCI-interconnects, consisting of multiple *rings*, communicating through *switches*. The **SCI-interconnect concept** can therefore be described as a set of SCI-rings. Figure 4.2a shows the conceptual idea, and figure 4.2b shows the textual representation in the program, the `class network`. As shown in the latter figure the structure of the module that simulates the concept is a ring-set. Its behavior is to create the rings in the ring-set according to the specification file and store this structure.

The name of the `class network`, does not correspond textually to the concept name **SCI-interconnect**, but is caused by a choice made during the early stages in the design process.



(a) The concept

```
class ring(<..>);
begin

% LOCAL DATA-TYPE DEFINITIONS:

   class link_set <..>;

% DATA STRUCTURE:

   ref(nodeinterface_set) NodeinterfaceSet;
   ref(node_set) NodeSet;
   ref(link_set) LinkSet;

% DEFINITIONS OF LOCAL PROCEDURES:

   procedure Initialize <..>;

% DEFINITIONS OF ACCESSIBLE PROCEDURES:

   procedure DisplaySimulation <..>;
   procedure DisplayStatistics <..>;

% STATEMENTS:

   Initialize;

end;
```

(b) The class

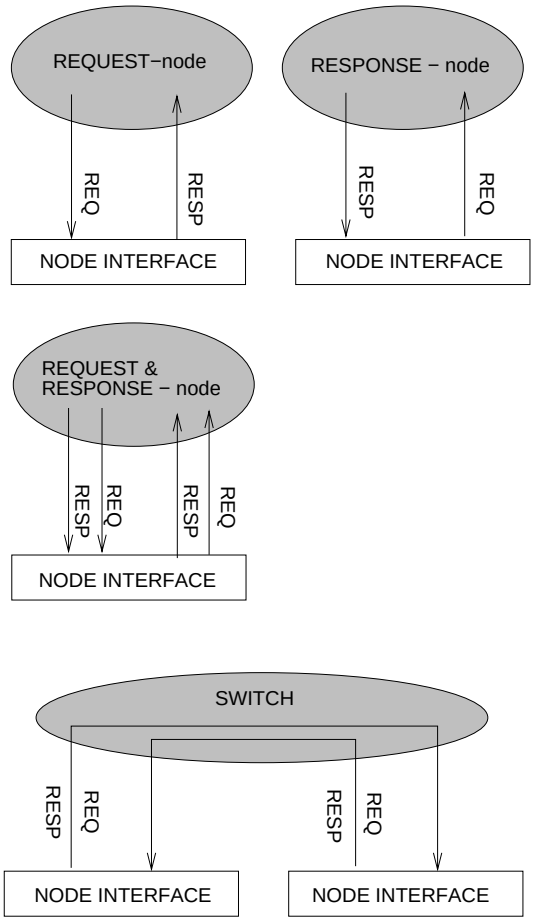
Figure 4.3: The SCI-ring

The SCI-ring — figure 4.3

The **SCI-ring concept** contains the concepts of processor, memory and cache, and the physical ring itself. The processors, memory chips and caches are referred to as application entities, and a processor communicate with a memory chip using the local transfer cloud which handles requests and possibly translating them into packets (refer to section 2.2). These packets are put onto the physical ring using the local node-interface. Between each node-interface there is a link, and to every node-interface there are two links, one input-link and one output-link, thereby forming a ring. In this context the term "node" will refer to a transfer cloud and its local application entities. The **SCI-ring** concept can therefore be described as a set of nodes, a set of node-interfaces and a set of links.

Figure 4.3a shows the **SCI-ring concept** and figure 4.3b shows the `class ring` which is the textual representation. An object of the `class ring` contain three different sets, a node-set, a node-interface-set and a link-set. The node concept is simulated by the `class node`, the node-interface concept is simulated by the `class nodeinterface` and the link concept is simulated by the `class link`, and these classes will be discussed later. Note that in figure 4.3a the number of links and node-interfaces are equal, whereas the number of nodes and node-interfaces may differ. The number of nodes and node-interfaces differ in the case when multiple SCI-rings are connected via switches, because a switch has two or more node-interfaces.

The behavior of the `class ring` is to read a part of the specification file and according to this file create the objects of the classes `node`, `nodeinterface` and `link`.



(a) The entity

```
process class node <..>;
begin
  <..>;
end;

node class nodeapplication <..>;
begin
  <..>;
end;

nodeapplication class RESPnode <..>;
begin
  <..>;
end;

nodeapplication class REQnode <..>;
begin
  <..>;
end;

nodeapplication class REQRESPnode <..>;
begin
  <..>;
end;

nodeapplication class SCIRTnode <..>;
begin
  <..>;
end;

node class SCIswitch <..>;
begin
  <..>;
end;
```

(b) The class

Figure 4.4: The SCI-node

The node — figure 4.4

An application process entity is represented by a processor, a memory chip, a cache or a combination of these entities, and in an SCI-interconnect these entities receive or transmit ordinary requests or responses, using the local transfer cloud. The transfer cloud will handle these requests or responses and possibly translates them into packets which are transmitted on the interconnect (refer to section 2.2).

The `class node` simulates the application process entity and transfer cloud, under some simplifying assumptions. The `class node` does not consider the part of the SCI-protocol related to cache-coherence, and it does not behave in a request-response fashion. The `class node` only generate packets according to a specified size and according to a given load and traffic pattern.

The biggest difference between the `class node` and the previous modules like for example `class interconnect` and `class ring`, is that the `class node` represents one of the main building blocks in an SCI-interconnect. An object of `class node` therefore represents

an application process entity and the local transfer cloud, under the assumption described above. The parameters given to an object of `class node` determines the behavior of this object.

In the SCIsim simulator the **switch entity** is simulated by the `class SCIswitch` which is a subclass of `class node`, and whose behavior is simply to move packets from one *node-interface* to another. More precisely, an object of `class SCIswitch` moves a packet in the input-queue of one node-interface in one ring, to the output-queue in another node-interface in another ring. This mean that the `class SCIswitch` simulates a store-forward switch, and does not use worm-hole routing. Compared to a switch using worm-hole routing, a store-forward switch will represent a higher delay when load is low because packets have to be completely received before they are passed on. When load is high the difference between a switch using worm-hole routing and a switch using store-forward is expected to be small, because in either case, the switches would be saturated and packets have to be temporarily stored. A switch of this type is often referred to as a **bridge** or a 2×2 **switch**.

Figure 4.4a shows the simplified model of the **application process entity and transfer cloud** and figure 4.4b shows the the `class node`, the textual representation.

The node-interface — figure 4.5

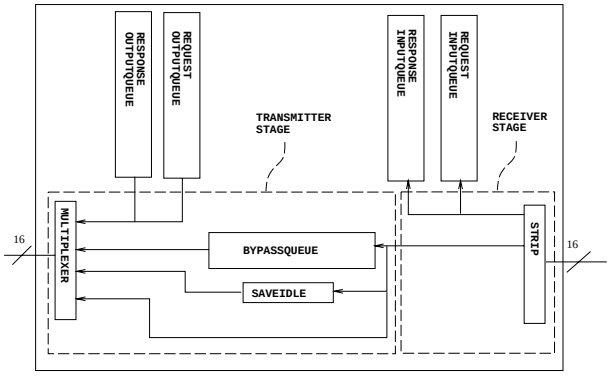
The **node-interface entity** implements the packet transportation layer, and acts as an interface between the transfer cloud and the ring.

The structure of the **node-interface entity** was shown in chapter 2 and is repeated in figure 4.5a. Figure 4.5b shows the textual representation of the `class nodeinterface` which simulates the **node-interface entity**. As illustrated in the figure, the **node-interface entity** has two output-queues, two input-queues, a bypass-queue, a stripper/decoder and a multiplexer. The behavior of an object of `class nodeinterface` is designed to resemble the **node-interface entity**. Packets from either of the two output-queues is transmitted to the next down-stream neighbour according to the ring-bandwidth protocol. Packets and symbols on the incoming link is either passed onto the transmitter-stage if the packet is not addressed to this node-interface, or inserted into the input-queue in accordance with the queue allocation protocol. When a node-interface receives a packet, it strips the packet from the ring, and sends No-Go idles to the transmitter-stage.

The `class nodeinterface` is a subclass of the `class process` (contained in the `class simulation`) and in this way simple process-simulation can be performed. The behavior of the `class nodeinterface` is governed by the incoming symbol, the state of the bypass-queue, the input-queues and the output-queues. The clock-cycle of the **node-interface entity** is simulated by a simple `while-do` loop in the `class nodeinterface`.

When designing the `class nodeinterface` some assumptions were made towards the underlying model of the **node-interface entity** to simplify the design process. Among those assumptions were:

- There is no skew in the links. Skew is caused by differences in wire lengths in parallel links, and limits the maximum signal speed.
- There is no drifting of the internal clocks of the node-interfaces. The internal clocks of the **node-interfaces entity** may drift slightly and this will cause them to become unsynchronized. To cope with drifting, elasticity buffers are used at the input-link.
- There is no prioritized traffic in the ring when basic SCI-rings are simulated. The SCI-protocol defines two priority levels.

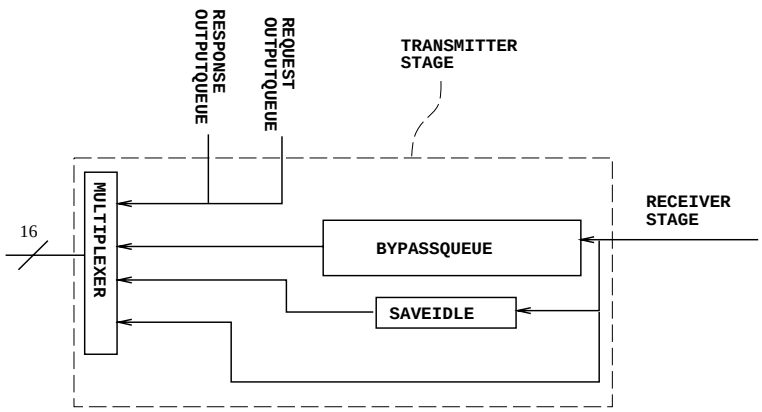


(a) The entity

```
process class nodeinterface(<..>);
begin

% LOCAL DATA-TYPE DEFINITIONS:

  class inputqueue <..>;
  class outputqueue <..>;
  class outputstatemachine <..>;
  statemachine class inputstatemachine <..>;
  class bypassqueue <..>;
  class stripper <..>;
  class multiplexer <..>;
  class saveidle <..>;

  class statemachine <..>;


% DATA STRUCTURE:

  ref(inputqueue) INPUTQREQ, INPUTQRESP;
  ref(outputqueue) OUTPUTQREQ, OUTPUTQRESP;
  ref(outputstatemachine) OUTPUTLOGIC;
  ref(inputstatemachine) INPUTLOGIC;
  ref(bypassqueue) BYPASSFIFO;
  ref(stripper) STRIP;
  ref(multiplexer) MUX;
  ref(saveidle) SAVEIDLEBUFFER;


% DEFINITIONS OF LOCAL PROCEDURES:

  procedure initialize <..>;


% DEFINITIONS OF ACCESSIBLE PROCEDURES:

  boolean procedure TransmitPacket <..>;
  ref(packet) procedure ReceiveResponsePacket <..>;
  ref(packet) procedure ReceiveRequestPacket <..>;

% STATEMENTS:

  initialize;
  while true do
  begin
    InputFromLink;
    TransitInputLogic;
    TransitOutputLogic;
    OutputToLink;
    hold(clock_frequence);
  end;

end;
```

(b) The class

Figure 4.5: The node-interface



(a) The entity - fair bandwidth allocation

```
class outputstatemachine;
begin
  <..>
end;

outputstatemachine class GoBitAlgorithm;
begin
  <..>
end;

outputstatemachine class SimpleAlgorithm;
begin
  <..>
end;

outputstatemachine class SCI_RT_outputstatemachine;
begin
  <..>
end;

SCI_RT_outputstatemachine class PacketPreemptLittleAsPossible;
begin
  <..>
end;
```

(b) The class

Figure 4.6: The transmitter-stage

The transmitter-stage — figure 4.6

The transmitter-stage is located in the node-interface and control the output-queue and the bypass-queue, and has to decide which queue to transmit from. The transmitter-stage therefore behave according to a queue arbitration protocol and there are different alternative which have to be considered. The following describe the alternatives considered in this thesis:

SCI flow control: In an SCI-ring the transmitter-stage must obey the ring bandwidth allocation protocol as described in section 2.2.3.

No flow control: In a ring without flow control the transmitter-stage will transmit from the output-queue only when the bypass-queue is empty. This means that bypassing symbols always have the right of way, and was described in section 2.2.3.

(a) The entity

```
class link(latency);
  real latency;
begin

% LOCAL DATA-TYPE DEFINITIONS

  class delayqueue <..>;

% DATA STRUCTURE:

  ref(delayqueue) symbol_sequence;

% DEFINITIONS OF LOCAL PROCEDURES:

  procedure initialize;

% DEFINITIONS OF ACCESSIBLE PROCEDURES:

  procedure PutSymbol <..>;
  procedure GetSymbol <..>;

% STATEMENTS:

  initialize;

end;
```

(b) The class

Figure 4.7: The link

Packet preemption protocol: In an SCI/RT-ring where the output-queues and priority bypass-queues are preemptive priority queues, the transmitter-stage must choose the queue with the highest priority. However, this is not always possible - if the output-queue has a higher priority than the bypass-queue and the bypass-queue is full, the transmitter-stage has to transmit from the bypass-queue or preempt the bypass-queue. Again there are various alternatives on how and when to preempt, and in this thesis the packet preemption protocol, described in section 2.3.3, is considered.

Figure 4.6a shows the transmitter-stage entity, and this entity is represented by the class-hierarchy of `class outputstatemachine` shown in 4.6b. There has been designed a sub-class for each of the three alternatives above. The `class GoBitAlgorithm` represents SCI flow control, `class SimpleAlgorithm` represents No flow control and `class PacketPreemptionLittleAsPossible` represents Packet preemption protocol.

While figure 4.6a does not indicate how the transmitter-stage should be represented, an underlying model of the transmitter-stage and its behavior had to be defined. Appendix A presents three proposals on how to describe the underlying model of the transmitter-stage, one for each of the three alternatives above.

The link, figure 4.7

The **link entity** represents the physical interconnect between two **node-interface entities**. In the SCI-standard there is defined three different link types - a parallel electrical, an electrical serial and an optical serial link. The SCIsim simulator assumes the first alternative as the underlying model, the parallel electrical link, and assumes that a link transmit complete symbols. Figure 4.7a shows the **link entity**, and figure 4.7b shows the `class link`, which is the textual representation.

When an object of `class link` is created, the signal propagation delay can be specified using the parameter. The structure of the `class link` is a queue whose length is calculated according to the delay parameter. The behavior of the `class link` is governed by the two procedures, `PutSymbol` and `GetSymbol`, which are accessible outside the class.

4.4 Summary

This chapter has described the design and building process of the SCI-simulator.

The main source of information when designing the SCI-simulator has been the SCI-standard [IEEE, 1992a], various articles published regarding SCI, the SCI/RT-draft [IEEE, 1992b] and various proposals on how to modify the SCI-protocol for real-time purposes presented on various mailing-list and on SCI/RT working-group meetings.

When designing the SCI-simulator, there were several requirements which had to be met. The simulator had to be modifiable and flexible, correct, parameterized, efficient and enable performance analysis, and these requirements were considered mandatory to reach the main goal. To meet the requirements a programming strategy was needed, and the choice fell on the object-oriented strategy as the main guiding principle.

In object-oriented programming one seeks to represent the real world entities and concepts as classes, and as a result the real-world structure can be recognized in the program-structure. In this way it is believed that modifiability and flexibility are enhanced because modification to the real-world entities affects the class representing it, and not necessary the entire program. The close resemblance between the structure of real-world system and program-structure enhances correctness, and measurements can be performed at places in the program which correspond to those in the real-world system. While the program contain classes, possibly sub-classes and classes which can be specified individually, objects can be generated at run-time and combined in different ways, and as a results the simulator can be made parameterized. Some uncertainty was associated with the efficiency requirement because experience indicated that object-oriented programs with extensive use of dynamic allocation, sometimes proved to be slow.

Part of the implementation of the SCI-simulator, henceforth referred to as **SCIsim-simulator**, was presented, showing how the real-world concepts and entities were expressed in program-code. The concepts of SCI-simulator, SCI-interconnect and SCI-ring, and the entities of node, node-interface, transmitter-stage and link, have been shown as examples of the implementation.

Chapter 5

Work related to simulation

This chapter describes the work related to the simulation of SCI and SCI/RT, and will explain how the actual measurements presented in chapter 6, 7 and 8 were obtained.

Section 5.1 will, according to Issue 4 - Issue 9 in chapter 3, describe which ring- and interconnect-configurations that have been considered, and in more detail describe the parameter-values assumed during simulation. Section 5.2 will describe and define the measurements emphasized in chapter 6, 7 and 8. Section 5.3 will explain how the actual measurements were obtained.

5.1 Topologies and parameters assumed in SCI- and SCI/RT-simulations

This section will describe the various topologies and parameters assumed during simulation of SCI and SCI/RT, topologies and parameters determined mainly by the issues in chapter 3 related to the performance of SCI and SCI/RT. Section 5.1.1 will define some concepts, section 5.1.2 will present and discuss the parameter-values assumed in SCI-simulations (Issue 4 - Issue 8 in chapter 3) and 5.1.3 will present and discuss the parameter-values assumed in SCI/RT-simulations (Issue 9 in chapter 3).

5.1.1 Defenitions

As explained in chapter 4 the SCIsim simulator is parameterized. Because the **parameters** remain unchanged during simulation, one simulation represents a fixed set of parameters and in the total parameter-space, one simulation represents a singular point. If we were to investigate, for example, how the output-queue size affects the performance and behavior of an SCI-ring, we would have to perform several simulations where the output-queue size was varied from one simulation to another. We will, therefore, distinguish between fixed parameters and varied parameters:

Fixed parameter: A parameter whose value remain fixed from one simulation to another.

Varied parameter: A parameter whose value is varied from one simulation to another.

The **load and traffic pattern** indicate the load in each node and to whom the node transmits its packets. We will distinguish between the following load and traffic patterns:

Uniform: The load is uniform, meaning that all nodes have the same load, and each node transmit to every other node uniformly. A node will never transmit to itself.

Hot-sender: Identical to uniform, except that one node, the so-called **hot-sender**, tries to transmit as much as possible. If no flow control mechanism is used in an SCI-ring, it is expected that a node behaving as a hot-sender will affect the downstream neighbours. The downstream neighbours will be drowned in packets, but the SCI-flow control mechanism (refer to section 2.2.3) should remedy this situation.

Node-starvation: Identical to uniform, except that one node never receives packets from other nodes. If no flow control mechanism is used in an SCI-ring, it is expected that a starved node will be affected. The starved node may be unable to transmit any packet if the load is high, because it will be always busy bypassing packets from other nodes. Again, the SCI-flow control mechanism should remedy this situation.

Both the hot-sender and node-starvation are special cases of the large set of non-uniform load and traffic patterns. The hot-sender and node-starvation load and traffic pattern were chosen because they are simple and intuitive cases of non-uniform load and traffic patterns.

Regarding the **size of send-packets**, we will distinguish between the following three cases (the CRC-symbol included):

16byte: All send-packets are 16 bytes long.

80byte: All send-packets are 80 bytes long.

Mixed: 60% of send-packets are 16 bytes long, 40% are 80 bytes long.

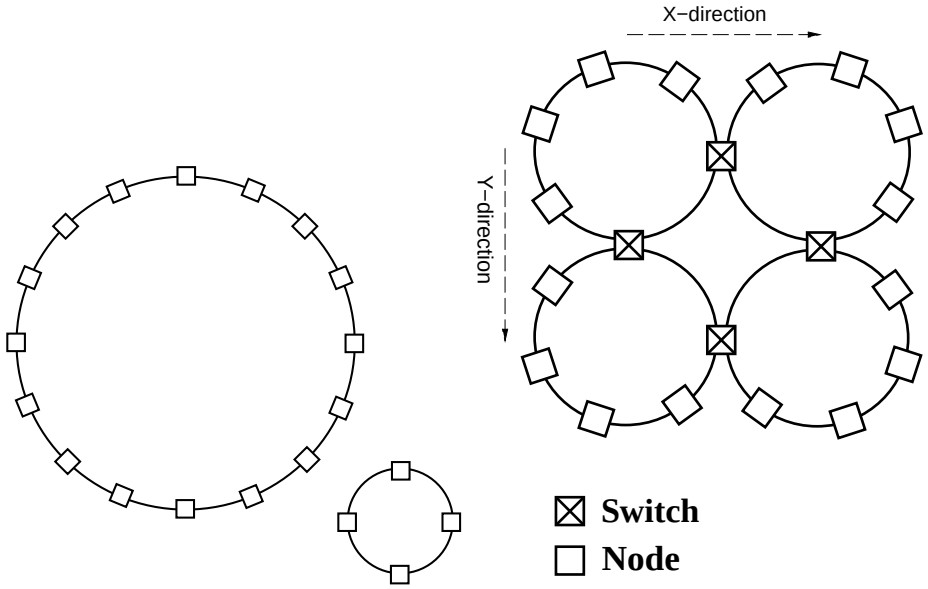
In the SCIsim-simulator, the load is specified in the following way: For each node it is specified how often the node is active and the probability that the it generates a send-packet when it *is* active. While there are only two possible outcomes when a node is active (generate a packet or not), the probability that a packet is generated is unchanged (all parameters remain unaltered during simulation) and the outcome of one trial does not affect the subsequent trials, this mean that the nodes in the SCIsim-simulator perform a sequence of **Bernoulli-trials** [Bhattacharyya and Johnson, 1977].

If we denote the outcome of the above Bernoulli-trial "generate packet", with *success (S)* and the opposite, "do not generate packet", with *failure (F)*, the number of successes X in n Bernoulli trials is binomially distributed. The mean value of X is $Mean = np$ [Bhattacharyya and Johnson, 1977].

When we simulate T nanoseconds of the life of an SCI-ring with N nodes, where each $node_i$ in the ring is active every t_i-th nanosecond and the probability is p_i that the node generates a packet when it is active, the mean number of packets generated by $node_i$ will be:

$$Mean_i = (\frac{T}{t_i})p_i$$

If the simulation-time is high, the number of trials will be high also (in this thesis the number of trials will be 40000 or higher per node). It is therefore reasonable to calculate the load according to the following formula (s_i denotes the size of send-packets generated by $node_i$):



(a) Single-ring (b) 4-ring interconnect

Figure 5.1: Simulated topologies related to SCI

$$Load_i = \frac{Mean_i s_i}{T} = \frac{(\frac{T}{t_i})p_i s_i}{T} = \frac{p_i s_i}{t_i}(byte/ns)$$

In a uniform SCI-ring, we will have for all $node_i$ that $t_i = t$, $p_i = p$ and $s_i = s$, and consequently, the load in each node and in the ring as a whole will be:

$$Load_i = \frac{ps}{t}(byte/ns)$$

$$Load_{ring} = N\frac{ps}{t}(byte/ns)$$

The load will be increased from one simulation to another, starting with a value close to zero and going up to a value where the results seem to stabilize.

5.1.2 Assumptions regarding SCI-simulations

In order to decide on Issue 4 - Issue 7 in chapter 3 (related to the performance of SCI), one of the main topologies will be the single SCI-ring as shown in figure 5.1a. The ring-size will be varied (4 nodes or 16 nodes), the load and traffic pattern will be varied (uniform, hot-sender and node-starvation), the transmitter-stage will be varied (flow control or no flow control), the size of send-packets will be varied (16byte, 80byte or mixed) and the load will be varied (from 0.0byte/ns and upward). By varying all these parameters we will be able to determine whether these parameters affects the performance of the single SCI-ring (Issue 4),

PARAMETER	VALUE
Topology	Single ring, figure 5.1a
Size	4 nodes, 16 nodes
Load and traffic pattern	Uniform, hot-sender, node-starvation
Transmitter-stage	SCI flow control, no flow control
Send-packet size	16byte, 80byte, Mixed
Load	$(0.0, \rightarrow)$ byte/ns
Link delay	1.0 ns
Output-queue size	256 byte
Output-queue type	FIFO
Input-queue size	256 byte
Input-queue type	FIFO
Input-queue allocation opportunity interval	1000.0 ns
Bypass-queue size	256 byte
Bypass-queue type	FIFO
Node-interface clock frequency	2.0 ns
Node-interface minimum delay	8.0 ns
Application process clock frequency	10.0 ns
Application process receiving probability	1.0

Table 5.1: Summary of parameter-values used in single-ring simulations

by varying the ring-size in particular we will get an indication whether the ring is scalable (Issue 5), by varying the load and traffic pattern *and* the transmitter-stage we will be able to get an indication on whether the SCI flow control mechanism ensure fairness (Issue 6), and finally, by using the ring-size of 4nodes/16nodes, packet-size of 16byte/80byte/mixed, flow control or no flow control, and the load and traffic pattern uniform/hot-sender/node-starvation we will get an indication on whether the SCIsim simulator produce results that are comparable to those presented in [Scott et.al., 1992] (Issue 7) because these parameters are identical to those assumed there.

In order to decide on Issue 8 in chapter 3, the second main topology will be an interconnect consisting of 16 nodes, the nodes being equally distributed in 4 rings and 4 switches enable communication between rings (figure 5.1b). Only the load is varied here (from 0.0byte/ns and upward), and this will give an indication whether the throughput and latency is better in the 4-ring interconnect than in the single ring with 16 nodes (Issue 8).

To keep the parameter-space tractable the remaining parameters were fixed. For quick reference the parameters and corresponding values are summarized in table 5.1 and 5.2. In these tables, the value of fixed parameters are singular values whereas the values of varied parameters are interval or sequence of values.

When flow control was used, the standard SCI flow control mechanism was assumed (refer to figure 2.8 in section 2.2.3). If no flow control was used, the strategy was simply to allow a node-interface to transmit *only* when the bypass queue was empty (refer to section 2.2.3).

The parameter controlling the load in each node was also varied, except in the node acting as the hot-sender where the load was fixed throughout all simulations.

PARAMETER	VALUE
Topology	4 rings w/4 switches, figure 5.1b
Size	16 nodes
Load and traffic pattern	Uniform
Transmitter-stage	SCI flow control
Send-packet size	Mixed
Switch type	Store-forward
Routing	Refer to text in section 5.1.2
Load	$(0.0, \rightarrow)$ byte/ns
Link delay	1.0 ns
Output-queue size	256 byte
Output-queue type	FIFO
Input-queue size	256 byte
Input-queue type	FIFO
Input-queue allocation opportunity interval	1000.0 ns
Bypass-queue size	256 byte
Bypass-queue type	FIFO
Node-interface clock frequency	2.0 ns
Node-interface minimum delay	8.0 ns
Application process clock frequency	10.0 ns
Application process receiving probability	1.0

Table 5.2: Summary of parameter-values used in 4-ring interconnect simulation

In the 4-ring interconnect simulation, 2×2 store-forward switches were used. In this kind of switch, packets have to be fully received before they are moved to the next ring. The routing is defined and proved deadlock-free in the following:

Routing in the 4-ring interconnect: Assume a 4-ring interconnect as shown in figure 5.1 and that source-node $N1$ wish to transmit packet S to target-node $N2$. There are three cases we have to consider:

Case 1: If $N2$ is located in the same ring as $N1$, S will not pass through any switch.

Case 2: If $N1$ and $N2$ is located in two neighbouring rings, S will pass through the switch connecting the two neighbouring rings.

Case 3: If $N1$ and $N2$ is located in two diagonally opposite ring, S will first pass through the switch in X-direction, then the switch in Y-direction.

Routing properties: The routing is static and all packets transmitted between the same pair $(N1, N2)$ will pass through the same switch(es). Another consequence of the routing strategy is that a switch in Y-direction will only pass packets destined for a node in the neighbouring ring, whereas a switch in X-direction will also pass packets to the diagonally opposite ring.

In order to prove that the routing is deadlock-free, we have to show that any packet generated by any node, to any node, in the 4-ring interconnect will eventually reach its destination. An underlying assumption in the proof is that the ring bandwidth protocol and the input-queue allocation protocol ensure forward progress.

Proof of Case 1: If $N1$ and $N2$ is located in the same ring we know that S will be accepted by $N2$ because forward progress is ensured by the fair bandwidth allocation protocol and the input-queue allocation protocol. As a result S will reach $N2$.

Proof of Case 2: If $N1$ and $N2$ is located in two neighbouring rings, the two following sub-cases have to be considered:

Case 2': $N1$ and $N2$ is located in two neighbouring rings in Y-direction. According to **Case 1**, packets in the outgoing node-interface of the switch will eventually leave the switch and be accepted by their target-node. This will in turn create space in the switch to accommodate S and the packet will eventually be accepted by the switch, again according to **Case 1**. Because a switch in Y-direction only passes through packets between neighbouring rings, S will reach $N2$.

Case 2": $N1$ and $N2$ is located in two neighbouring rings in X-direction. First step is to prove that S will be accepted by the switch in X-direction. The switch in X-direction passes through packets to the neighbouring ring *and* the diagonally opposite ring, the first kind of packets will leave the switch according to **Case 1** and the second kind of packets will leave the switch because they will eventually be accepted by the switch in Y-direction according to **Case 2'**. This will in turn create space in the switch in X-direction, enabling it to accommodate S, and S will eventually be accepted by the switch i X-direction. Once S is in the switch in X-direction, it will reach its target-node, again according to **Case 2'**. As a result S will reach $N2$.

Proof of Case 3: If $N1$ and $N2$ is located in two diagonally opposite rings, we know that S will first pass through the switch in X-direction and then through the switch in Y-direction, according to **Case 2"**.

Because any packet S, generated by any source-node $N1$ to any target-node $N2$, reaches its destination, the routing in the 4-ring interconnect is deadlock-free.

5.1.3 Assumptions regarding SCI/RT-simulations

In order to decide on Issue 9 in chapter 3 (related to the performance of SCI/RT), the main topology will be a 4-node single-ring. The load and traffic pattern is uniform and packets are generated uniformly on four priority levels.

The output-queue and the bypass queue are both priority queues, and to handle with the situation where a high priority packet tries to gain access to a full queue, both queues use preemption. Preemption in the bypass queue is carried out by *converting* send-packets with a lower priority than the incoming packet, into echo-packet. Preemption in the output-queue is carried out by *deleting* unsent and retry packets with a lower priority than the new packet (send-packets awaiting an echo will never be deleted). The transmitter-stage controls the output-queues and the bypass queue, and behave according to the packet preemption protocol proposed in [IEEE, 1992b] (refer to section 2.3.3).

The size of send-packets were specified as **mixed** (refer to section 5.1.2). The size of the bypass queues was set to 256 bytes and echo-packets inherited the priority of the corresponding send-packet. It is reasonable that the size of the bypass queues and the priority of echo-packets will affect the behavior and performance of an SCI/RT ring, but this has not been investigated in this thesis.

For quick reference the parameters-values are summarized in table 5.3. Note that the input-queues in a node-interface will never fill up because the application process will remove the packets in the input-queue as soon as they enter. Therefore ordinary FIFO queues are used for input-queues.

5.2 Measurements emphasized in simulation

This section will define the measurements emphasized during simulation, and in chapter 6, 7 and 8.

To analyze the performance and behavior of the packet transportation layer, throughput and latency have been measured during simulation, for each node, for the ring in general and, in the case of SCI/RT simulation, for each priority level.

The following subsections will elaborate the concept of throughput, latency and performance.

5.2.1 Throughput

Throughput will be given in bytes per nanosecond $byte/ns$. A throughput-estimate is calculated in the SCIsim simulator by counting the number of bytes passing a certain point and divide it by the simulation time. Throughput is calculated in different ways at various places in the simulator.

The following throughput-measurements have been emphasized in the SCIsim simulator and will be used in the presentation of simulation results in chapter 6, 7 and 8:

PARAMETER	VALUE
Topology	Single ring
Size	4 nodes
Load and traffic pattern	Uniform
Priority distribution of traffic	Uniform w/four pri.levels
Transmitter-stage	Packet preemption protocol
Send-packet size	Mixed
Echo-packet priority	Inherit send-packet
Load	$(0.0, \rightarrow)$ byte/ns
Link delay	1.0 ns
Output-queue size	256 byte
Output-queue type	PRIQ w/preemption
Input-queue size	256 byte
Input-queue type	FIFO
Input-queue allocation opportunity interval	1000.0 ns
Bypass-queue size	256 byte
Bypass-queue type	PRIQ w/preemption
Node-interface clock frequency	2.0 ns
Node-interface minimum delay	8.0 ns
Application process clock frequency	10.0 ns
Application process receiving probability	1.0

Table 5.3: Summary of parameter-values used in SCI/RT simulations

RawThroughput: This throughput-measurement is calculated using the send-packets received by an application process. The whole send-packet, minus the CRC-symbol, is used in the calculation.

The throughput-measurement is calculated for request and response packets separately, for each application process, for the ring as a whole and the interconnect in general (if there are multiple rings).

NetThroughput: This throughput-measurement is calculated using the send-packets received by an application process. Only the data-bytes are used in the calculation.

The throughput-measurement is calculated for request and response packets separately, for each application process, for the ring as a whole and the interconnect in general (if there are multiple rings).

RecThroughput: This throughput-measurement is calculated using the send-packets received by a node-interface that are addressed to the node-interface. The complete send-packet is used in the calculation, and regardless whether the packet is accepted or rejected by the node-interface (The AB-retry protocol control the access of input-queues, refer to section 2.2.3).

This throughput-measurement is calculated for request and response packets separately, for each node-interface, for the ring as a whole and the interconnect in general (if there are multiple rings).

AckRecThroughput: This throughput-measurement is calculated using the send-packets received by a node-interface *and* that are inserted into the input-queue. This mean that only send-packets that are accepted and stored in the input-queue are counted. The whole send-packet, minus the CRC-symbol, is used in the calculation.

The throughput-measurement is calculated for request and response packets separately, for each node-interface, for the ring as a whole and the interconnect in general (if there are multiple rings).

TransThroughput: This throughput-measurement is calculated using the send-packets transmitted by a node-interface. The whole send-packet is used in the calculation, and regardless whether the packet was accepted or rejected by the receiving node-interface.

This throughput-measurement is calculated for each node-interface and request and response packets.

AckTransThroughput: This throughput-measurement is calculated using the send-packets transmitted by a node-interface *and* that are acknowledged by the receiving node-interface. The whole send-packet is used in the calculation.

This throughput-measurement is calculated for each node-interface, for request and response packets separately and for each priority level (if any).

When no packets are rejected by the receiving node-interfaces, total RecThroughput and total *Ack*RecThroughput of an SCI-ring will differ only slightly, because the CRC-symbol is included in the calculation of RecThroughput and not in *Ack*RecThroughput. When packets are rejected by the receiving node-interface, total RecThroughput and total *Ack*RecThroughput of an SCI-ring may differ significantly because all send-packets received

by a node-interface is included in the calculation of RecThroughput whereas in the calculation of *Ack*RecThroughput only those packets which are accepted are included. This also applies to the relationship between TransThroughput and *Ack*TransThroughput.

When single SCI-rings are simulated the total RawThroughput of the ring equals the total AckRecThroughput. On the other hand, when multi ring topologies are simulated, total RawThroughput and total AckRecThroughput will differ, because some of the node-interfaces are associated with switches.

5.2.2 Latency

Latency will be given in nanoseconds *ns*. The concept of latency in the SCIsim simulator is related to the send-packets and to the significant moments during a send-packet's life. In the SCIsim simulator the life of a send-packet begins when it is created in the source-node and ends when it is received by the target-node. A significant moment during a send-packet's life is for example when it is transmitted on the ring.

As explained in section 2.2.3, the **packet transmission protocol** is used when a packet is transmitted from one node to another. If the source-node and target-node are located in the same ring, a **local sub-action** takes place, which means that the send-packet is transmitted on the ring and an echo-packet is returned as an acknowledgment. Otherwise, when the source-node and target-node are located in different rings, the send-packet has to pass through several switches and rings, and a **remote sub-action** is initiated. This mean that a local sub-action is initiated in each ring, between the source-node and the first switch, between intermediate switches and finally, between the last switch and the target-node. To elaborate, a send-packet is transmitted from *one source-node* to *one target-node*, but may pass through several rings and consequently *several node-interfaces*. In each ring the send-packet is transmitted from *one transmitter node-interface* to *one receiver node-interface*.

The following latency-measurements have been emphasized in the SCIsim simulator and will be used in the presentation of simulation results in chapter 6, 7 and 8:

RemoteSubActionLatency: The time it takes from the packet is generated in the source-node by an application process, until it is received by an application process in the target-node.

LocalSubActionLatency: The time it takes from the packet is inserted into the output-queue of the transmitter node-interface, until it is removed from the output-queue by a DONE-echo.

LocalSubActionNoEchoLatency: The time it takes from the packet is inserted into the output-queue of the transmitter node-interface, until it is fully received by the receiver node-interface, regardless whether it is accepted or rejected.

RingLocalSubActionLatency: The time it takes from the packet is transmitted on the ring by the transmitter node-interface (counting from the first symbol), until the corresponding echo is received, regardless whether it was a DONE-echo or RETRY-echo.

RingLocalSubActionNoEchoLatency: The time it takes from the packet is transmitted on the ring by the transmitter node-interface (counting from the the first symbol),

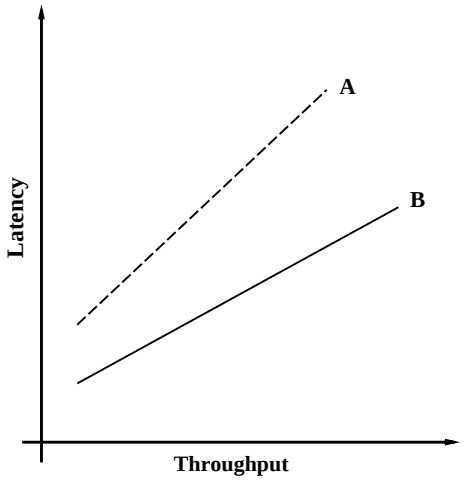


Figure 5.2: Comparing the performance, an example

until it is fully received by the receiver node-interface, regardless whether it is accepted or rejected.

There is a significant difference between the RemoteSubActionLatency and the other four latency-measurements, because the former is calculated by the application processes, while the other four are calculated by the node-interfaces. The RemoteSubActionLatency may include the passing of several switches and rings, while the other four latency-measurements are ring-local.

There is also a significant difference between LocalSubActionLatency (LocalSubActionNoEchoLatency) and *Ring*LocalSubActionLatency (*Ring*LocalSubActionNoEchoLatency) because the first latency-measurement include the time spent in the output-queue. It is reasonable to expect that when load is low, the quantitative difference between LocalSubActionLatency (LocalSubActionNoEchoLatency) and *Ring*LocalSubActionLatency (*Ring*LocalSubActionNoEchoLatency) will be small, but when load is high the quantitative difference will increase because send-packets would spend more time in the output-queue.

During simulation, send-packets will be generated and transmitted according to the user-provided specification, and each send-packet will give raise to the latency-measurements described above. These measurements are calculated locally in each node, for each ring and for the interconnect in general, and the mean-value, variance, standard deviation and confidence intervals are calculated also.

5.2.3 Performance

The concept of performance will in this thesis refer to the relationship between throughput and latency. Good performance is associated with high throughput and low latency. The term will be used relatively, indicating whether the performance in one case is better than the performance in another. In example in figure 5.2 it is therefore correct to state: «The performance of B is better than A».

5.3 How the measurements were obtained

The measurements, which will be presented later in this thesis, were obtained through the following process: First a number of specification files were written, then the SCIsim simulator was executed several times with a new specification file as input each time, and finally, when all simulations were terminated, the files containing the statistical measurements were analyzed and interesting data compiled into a more appropriate form (e.g. a graph).

The specification files were written so that they would, eventually, cover all combinations of parameter-values described in table 5.1, 5.2 and 5.3. As stated in the introduction to section 5.1, one simulation represents a *fixed* set of parameter values, and therefore to investigate the impact of increasing load in an SCI-ring, several simulations had to be performed. A typical task in this part of the work, would be to write 30-40 specification files which would differ only in the load-parameter. This was rather cumbersome and indicate that a prize had been paid, in terms of more work when specifying a simulation, when the simulator was made parameterized.

Preliminary simulations were performed in order to determine the length of the simulations. If results from these simulations indicated a too broad confidence interval, usually more than $\pm 5\%$, the simulations were rejected and a longer simulation-time was required. In hindsight it might have been better to have designed the SCIsim simulator so that simulations were run for the necessary amount of time to ensure a given level of precision, rather than the other way around.

The simulator were usually executed at times when computer-load was low, e.g. at night, in order to take advantage of idle computers and thereby execute longer simulations. In the SCIsim simulator a pseudo-random number generator was used in order to achieve randomization, and this pseudo-random generator has to be initialized with a seed-value. If the same seed is used every time the pseudo-random generator will produce the same sequence of "random" numbers. In the SCIsim simulator a new seed is ensured each time a simulation is started because it uses the value returned by the system sub-routine `time()` as the seed-value. The `time()` sub-routine is available within the UNIX-environment and returns the time since 00:00:00 GMT, Jan. 1, 1970, measured in seconds.

The simulator also produced a large amount of statistical information which were written to multiple files. The files had to analyzed and interesting measurements extracted and combined. Again this part of the work proved to be rather cumbersome.

5.4 Summary

This chapter has described the work related to simulating SCI and SCI/RT, described the various topologies and parameter-values assumed during simulation and which measurements have been emphasized in the remainder of the thesis.

The issues described in chapter 3, related to the performance of SCI and SCI/RT (Issue 4 - Issue 9), determined the main topologies and parameter-values. The main topologies have been the single SCI-ring, a 4-ring interconnect (figure 5.1) and a 4-node SCI/RT-ring. The SCIsim simulator was parameterized, and by varying the value of parameters (e.g. the load), one could examine how that particular parameter affected the simulated topology (e.g. increase in throughput). To help decide on Issue 4 - Issue 9, the ring-size was varied, the packet-size was varied, the type of flow control was varied, the load and traffic pattern

was varied and the load itself was varied. The exact parameter-values are summarized in table 5.1, 5.2 and 5.3.

This chapter has also defined the concepts of **throughput**, **latency** and **performance** in the context of SCI and SCI/RT. There are different ways to calculate throughput and latency, and those used in the remainder of the thesis have been defined in this chapter. Performance is defined as the relationship between throughput and latency, and good performance is associated with high throughput and low latency.

Simulation-results were obtained by using the SCIsim-simulator (chapter 4) with the parameters in table 5.1, 5.2 and 5.3 as input. Because a single simulation represent a fixed set of parameters, a sequence of simulations had to be performed where one parameter-value was changed from one simulation to another. During simulation, the SCIsim-simulator gathered data and presented the final results when it terminated. Simulation results were sometimes rejected if they proved too imprecise or the confidence intervals were too broad. In these cases, the simulations had to be re-run.

[This page has been intentionally left blank]

Chapter 6

Results from the simulation of SCI single-rings

This chapter presents and discusses results from the simulation of single SCI-rings, results which will help decide on Issue 4 - Issue 7 (chapter 3).

Section 6.1 will discuss some aspects regarding the presentation of results in this chapter and in the subsequent chapters 7 and 8.

There are three main sections in this chapter and apart from presenting and discussing results which relate to Issue 4 - Issue 7, these sections will also try to emphasize various properties of the SCI-ring. Section 6.2, related to uniform load and traffic pattern, will emphasize average values for throughput and latency, and these results will help us decide on Issue 4, Issue 5 and Issue 7. Section 6.3 and 6.4, related to hot-sender and node-starvation load and traffic pattern, will emphasize the properties of the SCI flow control mechanism, and these results will help us decide on Issue 4, Issue 6 and Issue 7.

Some of the results presented in this chapter are compared to results in [Scott et.al., 1992] and in brief terms, that article presents simulation-results of single SCI-rings assuming various ring-sizes, packet-sizes, flow control, load and traffic patterns, where both a simulator *and* a mathematical model were used.

6.1 Aspects regarding the presentation of results

The SCIsim simulator described in chapter 4 was used, and its input was the parameters described in chapter 5.

As explained in chapter 5, simulations were run for a considerable amount of time in order to ensure reliable results. If the results indicated a too broad confidence intervals (more than approximately $\pm 5\%$), these simulations were re-run for a longer period of time. In this way acceptable precision was ensured and consequently, acceptable precision are associated with the results in this chapter and in the subsequent chapters 7 and 8. To avoid drowning the reader in details, point estimates are shown, but as an example, interval estimates are presented in section 6.2.1.

Throughput this chapter and the subsequent chapters 7 and 8, results will be presented in graphs. In these graphs, a line is drawn between two singular points if it is reasonable to interpolate between them. There are some exceptions though where the results are rather jumbled, and lines are drawn in order to visualize related results more clearly (graph 6.14 and graph 8.2a).

When a graph is presented and discussed in this chapter, and the subsequent chapters, 7 and 8, the presentation and discussion will take place in the following way:

1. Presentation of the actual results.

2. Discussion of which consequences the results in **1** have for the properties of the real-world structure.

3. Discussion of whether the results are reasonable.

An underlying assumption in the discussion in **3** is that the SCIsim-simulator is correct, an assumption which is reasonable considering the discussion in chapter 4.

6.2 Uniform load and traffic pattern in single SCI-rings

This section will present and discuss results from the simulation of single SCI-rings where the load and traffic pattern was **uniform**. These results will help us decide on Issue 4, Issue 5 and Issue 7 presented in chapter 3. The following list describes the overall conditions assumed during simulation:

- **Topology:** Single ring.
- **Size:** 4 nodes or 16 nodes.
- **Load and traffic pattern:** Uniform.
- **Transmitter-stage:** SCI flow control, no flow control.
- **Send-packet size:** 16bytes, 80bytes or Mixed.

The load was increased from one simulation to another - starting with a value close to zero and going up to a level where the throughput and latency measurements had stabilized. The length of one simulation was determined after some preliminary simulations, some configurations and topologies required longer simulations than others to ensure reliable results (Refer to section 5.3). Therefore simulated time lay in the interval from $400000ns$ to $900000ns$. The remaining parameter-values assumed during simulation, can be found in table 5.1 in section 5.1.2.

In order to indicate the throughput of a uniform, single SCI-ring, this section will emphasize the **RecThroughput** as defined in section 5.2. This throughput-measurement includes all send-packets received by a node-interface that are addressed to the node-interface itself. Total and average values will be presented - the former include all node-interfaces in the ring, while the latter equals the total throughput divided by the number of nodes.

In order to indicate the latency, this section will emphasize **LocalSubActionLatency**, **LocalSubActionNoEchoLatency**, **RingLocalSubActionLatency** and **RingLocalSubActionNoEchoLatency** as defined in section 5.2. As indicated in the definition, these latency measurements are ring-local. Average and maximum values of these latency measurements will be presented - the former being the mean value of the sampled values.

Section 6.2.1 will present and discuss results from the simulation of SCI-rings of size 4 displaying a uniform load and traffic pattern, and which do not use the flow control mechanism.

Section 6.2.2 will present and discuss results from the simulation of uniform SCI-rings of size 4 which use the SCI flow control mechanism. While adding flow control to a ring will affect some measurements, other measurements are unaffected. Section 6.2.2 will therefore present and discuss results, which are related to the flow control mechanism, and compare them to results in section 6.2.1.

Section 6.2.3 will present and discuss results from the simulation of uniform SCI-rings with 16 nodes, which either use the SCI flow control mechanism or not. Again, increasing the size of an SCI-ring will affect some measurements, while other measurements are unaffected. Section 6.2.3 will therefore present and discuss results which are related to the larger size, and compare them to the ring in section 6.2.2.

Section 6.2.4 will give a summary of the main results related to uniform load and traffic pattern.

6.2.1 Results related to uniform SCI-rings with 4 nodes, no flow control

This section will present and discuss the simulation-results in the following order:

- Throughput and latency as a function of load for the whole ring, graph 6.1a-c.
- The relationship between throughput and latency for the whole ring, graph 6.2a-c and 6.3.
- Statistical properties of the simulation-results, graph 6.4a.

The three graphs in 6.1 shows throughput and latency as a function of load, and includes the three cases of 16byte, 80byte and mixed send-packets. The results will be discussed in the following:

Total RecThroughput as a function of total load, graph 6.1a.

This graph shows the total RecThroughput as a function of total load, when the send-packets are either 16byte, 80byte or mixed. Total load and total RecThroughput is specified in $byte/ns$ along the x-axis and y-axis respectively. Observing the results in graph 6.1a it is reasonable to state the following:

- When load is low, total RecThroughput as a function of total load approximates the linear function f(x)=x. This mean that total RecThroughput is approximately equal to total load, when total load is less than a certain limit **L**. In the 16byte case **L** is approximately $1.25 byte/ns$, in the 80byte and mixed case **L** is approximately $1.5 byte/ns$.
- When load is higher than **L**, total RecThroughput as a function of total load will drop below f(x)=x, and will increase more and more slowly as total load increase. When load is very high, total RecThroughput as a function of total load, will approximate a constant function f(x)=K. In the 16byte case K is approximately $1.15 byte/ns$, in the 80byte case approximately $1.70 byte/ns$ and in the mixed case approximately $1.55 byte/ns$.

The above observations indicate properties of the total RecThroughput in *real-life* SCI-rings. When total load is low and less than a certain limit, total RecThroughput equals

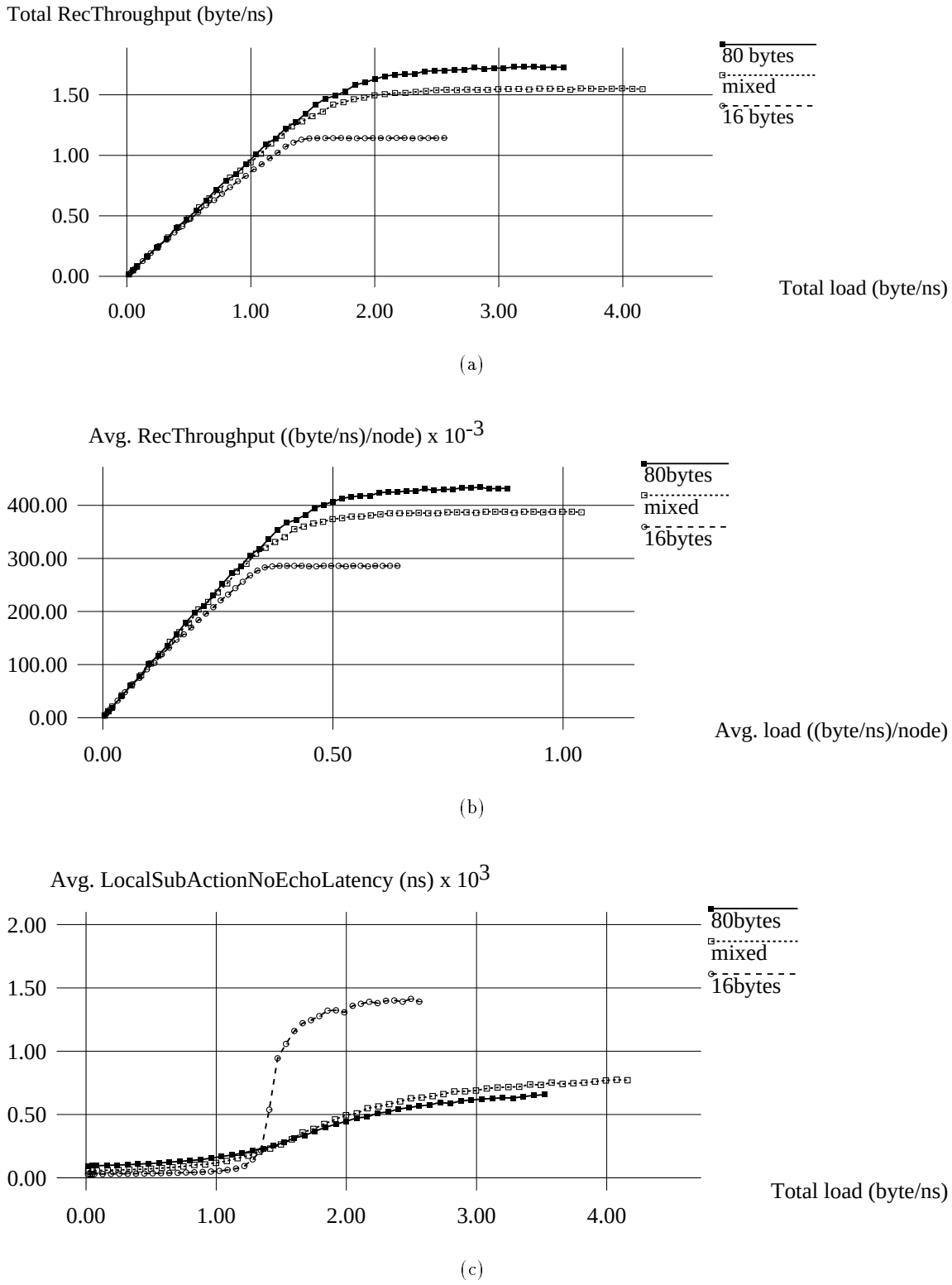


Figure 6.1: Throughput and latency as a function of load. (Uniform, 4 nodes, no flow control).

the total load. If the total load exceeds this limit, total RecThroughput will be less than the total load, and if total load increases further, total RecThroughput will approach a maximum value. Even if the total load should increase further, the total RecThroughput will remain stable and will not degrade.

This a reasonable conclusion on properties of the total RecThroughput in SCI-rings. The following rationale explains why:

- When the load is low in an SCI-ring, few packets circulate the ring. When a node-interface has transmitted a send-packet from the output-queue, it must empty the bypass-queue before it can transmit more send-packets (This process of emptying the bypass-queue is often referred to as **recovery**). In a lightly loaded SCI-ring the probability that packets try to pass a node-interface already busy transmitting from the output-queue, is small. Consequently a node-interface rarely has to empty the bypass-queue after transmission and send-packets inserted into the output-queue can be transmitted almost immediately. The output-queues will rarely fill up and a node-interface can transmit packets at the same rate as they are generated by the corresponding application process.

- When the load increase, more packets will circulate the ring, and consequently a node-interface is more often in recovery stage, and therefore unable to transmit packets from the output-queue. The probability that a node-interface is blocked in this way for a long time, has increased and once in a while, the output-queue of the node-interface is full, because the node application process has generated more packets than the node-interface can transmit. When the output-queue is full, the application process has to *discard* packets (or stop generating new packets). When packets are discarded, the total RecThroughput will be less than total load.

- When load is very high, the ring is filled with packets. Under these conditions a node-interface will be constantly busy, either transmitting packets or bypassing packets. The application processes generate packets at a higher rate than the node-interface can transmit and eventually, the output-queues will fill up and the application processes have to discard packets. The ring transmits at full speed and the total RecThroughput has reached its maximum value.

As we can observe from graph 6.1a, the point when total RecThroughput starts to approach the maximum total RecThroughput and the maximum value itself, depends very much on the size of send-packets. The reason is that smaller send-packets (16byte case) will experience a higher relative overhead compared to the larger packets (80byte case). To every send-packet transmitted on the ring, a 2byte idle-symbol will be appended and an 8-byte echo-packet will be generated (once the send-packet has arrived at the receiving node-interface). An idle-symbol and an echo-packet represent a higher relative overhead to a 16 byte send-packet, than to an 80 byte send-packet. Consequently, more bandwidth is spent on echo-packets and idle-symbols than on send-packet in the 16byte case, compared to the 80byte case. A combination of 16 byte and 80 byte packets, as in the mixed case, will lead to an intermediate case. Therefore a ring transmitting only 16 byte packets will be saturated more quickly (when total load is lower) than a ring transmitting only 80 byte packets, and will experience a lower maximum total RecThroughput.

An important thing to notice about an SCI-ring without flow control, is that once the ring is saturated, a node-interface can only transmit send-packets if it also receive packets

from other node-interfaces in the ring. It is only by receiving packets from other node-interfaces that a node-interface is able to empty the bypass-queue and eventually, transmit new send-packets.

Average RecThroughput as a function of average load, graph 6.1b

This graph shows the average RecThroughput as a function of average load when the send-packets are either 16byte, 80byte or mixed. Average load and average RecThroughput are specified in $(byte/ns)/node$ along the x-axis and y-axis respectively.

The intention of showing the graph is to present the exact values of average RecThroughput and because these values will be used later. All application processes are request-response type in these simulations, and this mean that they all produce and consume packets. Therefore the average load is equal to the total load averaged over the nodes and the average RecThroughput is equal to the total RecThroughput averaged over the nodes. Consequently the graph 6.1b is a scaled-down copy of the graph 6.1a. The maximum average RecThroughput in the 80byte case is $0.43(byte/ns)/node$, in the mixed case $0.39(byte/ns)/node$ and in the 16byte case $0.29(byte/ns)/node$

Refer to the discussion of graph 6.1a for further details.

Average LocalSubActionNoEchoLatency as a function of total load, graph 6.1c

This graph shows the average LocalSubActionNoEchoLatency as a function of total load when the send-packets are either 16byte, 80byte or mixed. Total load is specified in $byte/ns$ along the x-axis and LocalSubActionNoEchoLatency is specified in ns along the y-axis. The results presented in this graph have the following properties:

- When total load is less than approximately $1.4byte/ns$, the average LocalSubActionNoEchoLatency of the 16byte case is smaller than the average LocalSubActionNoEchoLatency of the 80byte case and the mixed. When total exceed $1.4byte/ns$, average LocalSubActionNoEchoLatency of the 16byte case is higher than the average LocalSubActionNoEchoLatency of the other two cases.

  When total load is less than approximately $1.6byte/ns$, the average LocalSubActionNoEchoLatency of the mixed case is less than the 80byte case, but when total load exceed $1.6byte/ns$ the situation is reversed and the average LocalSubActionNoEchoLatency is higher in the mixed case than in the 80byte case.

- In all three cases the average LocalSubActionNoEchoLatency increase as the total load increase, but the rate of growth in any point depends very much on the the total load in that particular point and the size of send-packets.

  This is easily visible in the 16byte case and when load is less than $1.0byte/ns$, a small increase in total load leads to a small increase in average LocalSubActionNoEchoLatency. Between $1.0byte/ns$ and $2.0byte/ns$, a small increase in total load leads to a large increase in average LocalSubActionNoEchoLatency. When the total load exceed $2.0byte/ns$ a small increase in total load again leads again to a small increase in average LocalSubActionNoEchoLatency. The average LocalSubActionNoEchoLatency in the 16byte case therefore display an asymptotic behavior, and the simulation results in graph 6.1c indicate that the lower asymptote is approximately $30ns$ and the upper asymptote is approximately $1400ns$.

An asymptotic property is also present in the results of 80byte case and the mixed case. The simulation results in graph 6.1c seem to indicate that the lower and upper asymptote in the mixed case is approximately $55ns$ and $700ns$ respectively. In the 80byte case the lower and upper asymptote is approximately $90ns$ and $800ns$.

The above results indicate some properties of the LocalSubActionNoEchoLatency in *real-life* SCI-rings. First, the results indicate that average LocalSubActionNoEchoLatency increase when total load increase. Second, there is a lower bound as well as an upper bound for the average LocalSubActionNoEchoLatency. The average LocalSubActionNoEchoLatency will approach the lower bound as total load approaches zero, and will approach the upper bound when total load grow higher and higher. Third, an increase in total load will lead to an increase in average LocalSubActionNoEchoLatency, but the rate of growth depends on the total load. If the ring transmits only 16 byte send-packets and the total load is less than $1.0byte/ns$ or greater than $2.0byte/ns$, a small increase in total load will lead to a small increase in average LocalSubActionNoEchoLatency, and when total load is between $1.0bytes/ns$ and $2.0byte/ns$ a small increase in total load will lead to a large increase in average LocalSubActionNoEchoLatency.

When total load is less than approximately $1.3byte/ns$ the average LocalSubActionNoEchoLatency of packets in an SCI-ring transmitting only 16 byte packets, is less than the average LocalSubActionNoEchoLatency of packets in an SCI-ring transmitting only 80 byte packets. The average LocalSubActionNoEchoLatency of packets in an SCI-ring transmitting mixed send-packets, fall between the 16byte case and 80byte case. When total load exceed approximately $1.6byte/ns$, the situation is reversed, and packets in an SCI-ring transmitting only 80byte packets experience a smaller average LocalSubActionNoEchoLatency than packets in SCI-rings transmitting only 16 byte send-packets, or SCI-rings transmitting mixed send-packets.

The above conclusion regarding LocalSubActionNoEchoLatency of packets is reasonable. The following rationale will explain why:

- An increase in load will imply an increase in average LocalSubActionNoEchoLatency, because an increase in load will lead to more packets circulating the ring. As a result, it becomes more likely that a node-interface will experience a longer recovery stage after a packet-transmission, and consequently it becomes more likely that new send-packets have to wait in the output-queue before the node-interface is to transmit again. Note that the LocalSubActionNoEchoLatency is the time from a send-packet is inserted into the output-queue until it has been received completely by the receiving node-interface.

  The reason behind the rapid increase in average LocalSubActionNoEchoLatency in the interval from $1.0byte/ns$ to $2.0byte/ns$, is that the output-queues starts to fill up.

- It is reasonable that there is a lower bound and an upper bound of the average LocalSubActionNoEchoLatency. When the load is low, less than $1.0byte/ns$ in the 16byte case, the application processes generate few packets, and few packets will circulate the ring. As a results, the node-interfaces are rarely busy bypassing packets (few packets circulate the ring) or in recovery stage (the application process generates few packets which imply a long time interval between packets). A send-packet inserted into the output-queue can therefore be transmitted almost instantaneously and once the send-packet is transmitted on the ring, it will experience a minimum of delay

because there are no other packets ahead of it. The lower bound of average LocalSubActionNoEchoLatency is therefore determined by the fixed minimum delay in the ring structure.

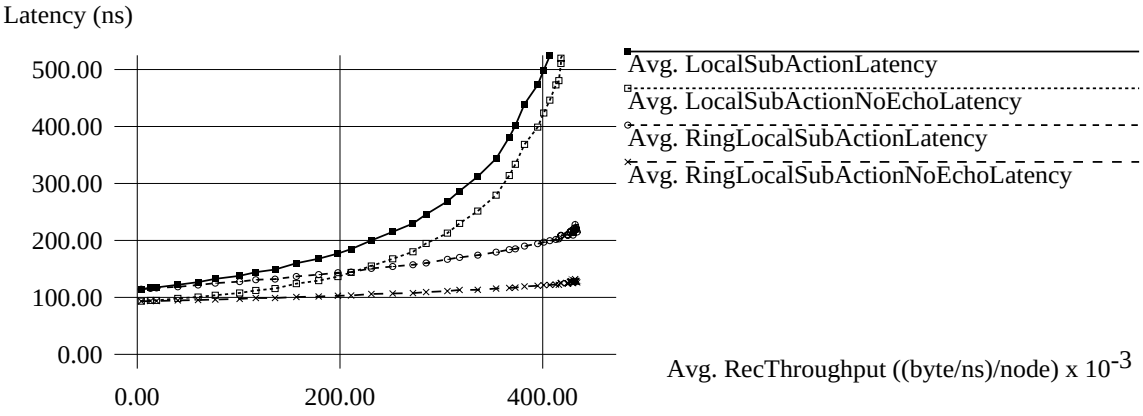
When load is very high, the application processes generate a large number of packets and a lot of packets will circulate the ring. The ring, consisting of links and bypass-queues, will be completely filled with send-packets and echo-packets, and the ring circumference is at its largest. When new send-packets are inserted into the output-queue, the node-interface is most likely busy bypassing packets or recovering from an earlier transmission. If the load is sufficiently high, an application process will generate packets at a higher rate than the node-interface can transmit, and after some time the output-queue will fill up. When the output-queue in all node-interfaces are full, the average LocalSubActionNoEchoLatency will not increase further because no new packets can be inserted into the output-queue until space is freed in the output-queue. The ring continue to transmit packets at full speed so eventually space is freed in the output-queues. While the average LocalSubActionNoEchoLatency depends on the output-queue size, the upper bound will depend on the maximum output-queue size. If the size of the output-queues had been infinite the average LocalSubActionNoEchoLatency would not have been bounded.

- It is reasonable that the upper and lower bound of average LocalSubActionNoEchoLatency depends on the size of send-packets used in the ring. When load is low, a packet will experience almost no delay except for the fixed delay in links and node-interfaces. It will take more time to transmit a large packet than a smaller packet, for example an 80 byte send-packet needs $80ns$ to leave the transmitting node-interface, whereas a 16 byte send-packet only needs $16ns$. When load is very high, the output-queues are filled up completely and the ring is transmitting at full speed. As explained in the discussion of graph 6.1a, a 16 byte send-packet has a higher relative overhead than an 80 byte send-packet. Consequently, more bandwidth is spent on echo-packets and idle-symbols and less bandwidth would be available for packet-transmission in an SCI-ring transmitting only 16byte packets, compared to an SCI-ring transmitting only 80 byte send-packets. In other words, it will take more time to transmit the same amount of bytes if only 16byte send-packets are used, than it would if 80 byte send-packets are used.

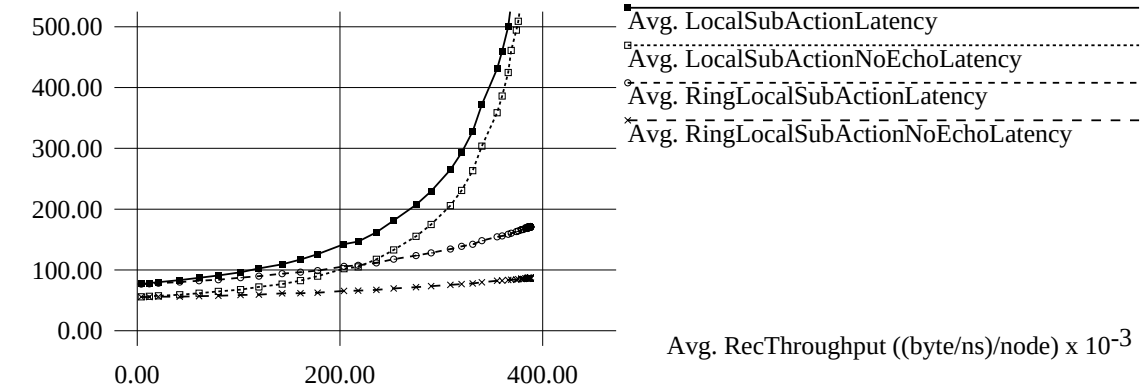
We have now discussed RecThroughput and latency as a function of load. Up to a certain point, the total RecThroughput increase approximately linearly when total load increase, but beyond this point the total RecThroughput begins to level off and approach a **stable maximum value**. This maximum value depends on the size of send-packets transmitted in the ring. The average LocalSubActionNoEchoLatency also increase as the total load increase, and are **bounded upward and downward**. The exact values of these boundaries depend on the size of send-packets transmitted in the ring. In addition the upper bound of average LocalSubActionNoEchoLatency depends on the maximum output-queue size.

The relationship between average RecThroughput and average latency, graph 6.2a-c and 6.3.

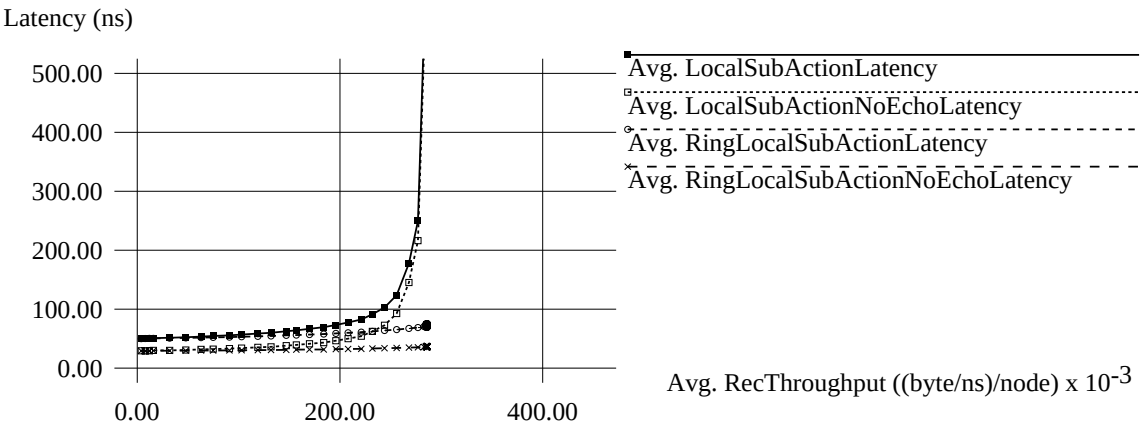
The three graphs 6.2a-c show the relationship between average RecThroughput and various latency measurements, when the send-packets are either 80byte, mixed or 16bytes



(a) 80byte



(b) mixed



(c) 16byte

Figure 6.2: The relationship between throughput and latency (Uniform, 4 nodes, no flow control).

respectively. The average RecThroughput is specified in $(byte/ns)/node$ along the x-axis and the latency is specified in **(ns)** along the y-axis. The four latency measurements **LocalSubActionLatency**, **LocalSubActionNoEchoLatency**, **RingLocalSubActionLatency** and **RingLocalSubActionNoEchoLatency** are shown. Refer to section 5.2 for further details on these latency measurements.

The three graphs 6.2a-c have several properties in common. They will be discussed in the following:

1. When average RecThroughput is low, the average LocalSubActionLatency is very close to the average *Ring*LocalSubActionLatency, but as the RecThroughput increase, the average LocalSubActionLatency increase a lot more than the average *Ring*LocalSubActionLatency. This is the case in graph 6.2a (80byte) and 6.2b (mixed) when average RecThroughput is less than $0.02(byte/ns)/node$, and in graph 6.2c (16byte) when average RecThroughput is less than $0.1(byte/ns)/node$.

   This indicate that in uniform SCI-rings, the average LocalSubActionLatency and average *Ring*LocalSubActionLatency are approximately equal when average RecThroughput is low, but will diverge when load increase because the average LocalSubActionLatency increase more rapidly than the average *Ring*LocalSubActionLatency.

   This is a reasonable conclusion because when RecThroughput is low, packets spend little time in the output-queues. Consequently, the LocalSubActionLatency and *Ring*LocalSubActionLatency of a send-packet will be equal in the average case. Note that the only difference between LocalSubActionLatency and *Ring*LocalSubActionLatency is that the former include the time spent in the output-queue.

   We also observe that the average *Ring*LocalSubActionLatency increase slowly compared to the average LocalSubActionLatency, and then stops abruptly when the RecThroughput reaches its maximum value. This indicates that the average *Ring*LocalSubActionLatency does not increase indefinitely once the ring is saturated. This is reasonable because once the ring is saturated, the ring circumference is at its largest and the *Ring*LocalSubActionLatency depends on the ring circumference. Also the average LocalSubActionLatency will stop abruptly (not shown in the three graphs 6.2a-c but is shown in graph 6.1c) because when the ring is saturated the ring circumference has reached its maximum size ***and*** the output-queues are full. The exact point where this situation occurs depends on whether 16 byte packets, 80 byte packets or mixed packets have been used.

2. When average RecThroughput is low, the average LocalSubActionNoEchoLatency is very close to the average *Ring*LocalSubActionNoEchoLatency, but as RecThroughput increases the average LocalSubActionNoEchoLatency will increase a lot more than the average *Ring*LocalSubActionNoEchoLatency.

   This indicate that in uniform SCI-rings, the average LocalSubActionNoEchoLatency and average *Ring*LocalSubActionNoEchoLatency are approximately equal when average RecThroughput is low, but will diverge when load increase, because the average LocalSubActionNoEchoLatency increase more rapidly than the average *Ring*LocalSubActionNoEchoLatency. There is also an upper bound for the *Ring*LocalSubActionNoEchoLatency.

   This is analogous to the above result and the reader should refer discussion in **1** for further details. Note that the only difference between LocalSubActionNoEchoLatency

and *Ring*LocalSubActionNoEchoLatency is that the former include the time spent in the output-queue.

3. In the 16byte case (graph 6.2c) we observe that for any given RecThroughput, the average RingLocalSubActionLatency is approximately twice the corresponding RingLocalSubAction*NoEcho*Latency.

   This indicate that in a uniform SCI-ring transmitting only 16 byte send-packets, the RingLocalSubActionLatency of the packets is approximately twice the RingLocalSubAction*NoEcho*Latency.

   This is reasonable because a send-packet has to traverse half-way around the ring in the average case, and the corresponding echo-packet has to traverse the other half. Note that RingLocalSubActionLatency include the time to traverse both the send-packet *and* the echo-packet (refer to section 5.2 for more details), while the RingLocalSubAction*NoEcho*Latency the time to traverse the send-packet. The RingLocalSubActionLatency of a 16 byte send-packet is not exactly twice the RingLocalSubAction*NoEcho*Latency because an echo-packet is only 8 bytes long.

4. For any given RecThroughput, the difference between average LocalSubActionLatency and average LocalSubAction*NoEcho*Latency is equal to the difference between average RingLocalSubActionLatency and average RingLocalSubAction*NoEcho*Latency.

   This is reasonable because in these simulations, no send-packets are rejected by the receiving node-interface and echoes will always indicate a successful transmission. Both the LocalSubActionLatency and RingLocalSubActionLatency include the time it takes to send the echo-packet back to the transmitting node-interface, whereas LocalSubAction*NoEcho*Latency and RingLocalSubAction*NoEcho*Latency does not.

In graph 6.3, the 16byte, 80byte and mixed case are compared, and the relationship between average RecThroughput and average LocalSubActionNoEchoLatency is shown. When average RecThroughput is less than approximately $0.27(byte/ns)/node$ the average LocalSubActionNoEchoLatency in the 16byte case is less than that of the 80byte case and the mixed case. If we with the term "good performance" associate high RecThroughput and low latency, the performance of the average node in an SCI-ring transmitting 16 byte packets are *better* than the performance of the average node in an SCI-ring transmitting 80 byte packets, when average RecThroughput is less than $0.27(byte/ns)/node$.

When average RecThroughput exceed $0.27(byte/ns)/node$, the situation is reversed, and the performance of an SCI-ring transmitting only 16 byte packets is *worse* than the performance of SCI-rings transmitting only 80 bytes packets.

This is reasonable because an SCI-ring with 4 nodes transmitting only 16 byte packets is saturated when the average RecThroughput approaches $0.29(byte/ns)/node$. Generating even more packets will only add to the output-queue length and thereby the time packets spend in the queue. The result is that the LocalSubActionNoEchoLatency will increase.

This also apply to the mixed packet case and the 80 byte packet case. The performance is better in the mixed case than in the 80 byte case, when the average RecThroughput is less than $0.32(byte/ns)/node$. When average RecThroughput exceeds $0.32(byte/ns)/node$, the situation is reversed and the packets in the 80byte case experience a lower LocalSubActionNoEchoLatency than packets in the mixed case.

These results can be compared to results presented in [Scott et.al., 1992], and graph 6.3 is directly comparable to a graph there. The results in graph 6.3 resemble the re-

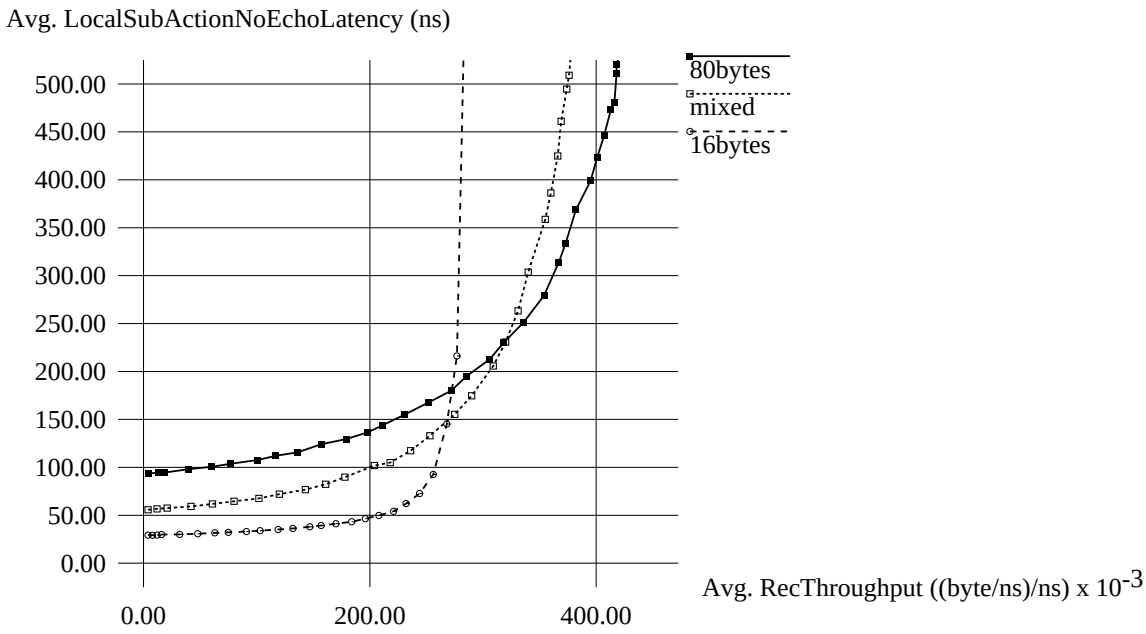


Figure 6.3: Performance comparison of different packet sizes (Uniform, 4 nodes, no flow control)

	Graph 6.3	**[Scott et.al., 1992]**	**Deviation**
16 byte vs. mixed	$(0.27, 150)$	$(0.26, 160)$	$(-4\%, +7\%)$
80 byte vs. mixed	$(0.32, 230)$	$(0.31, 250)$	$(-3\%, +9\%)$

Table 6.1: Points of intersection

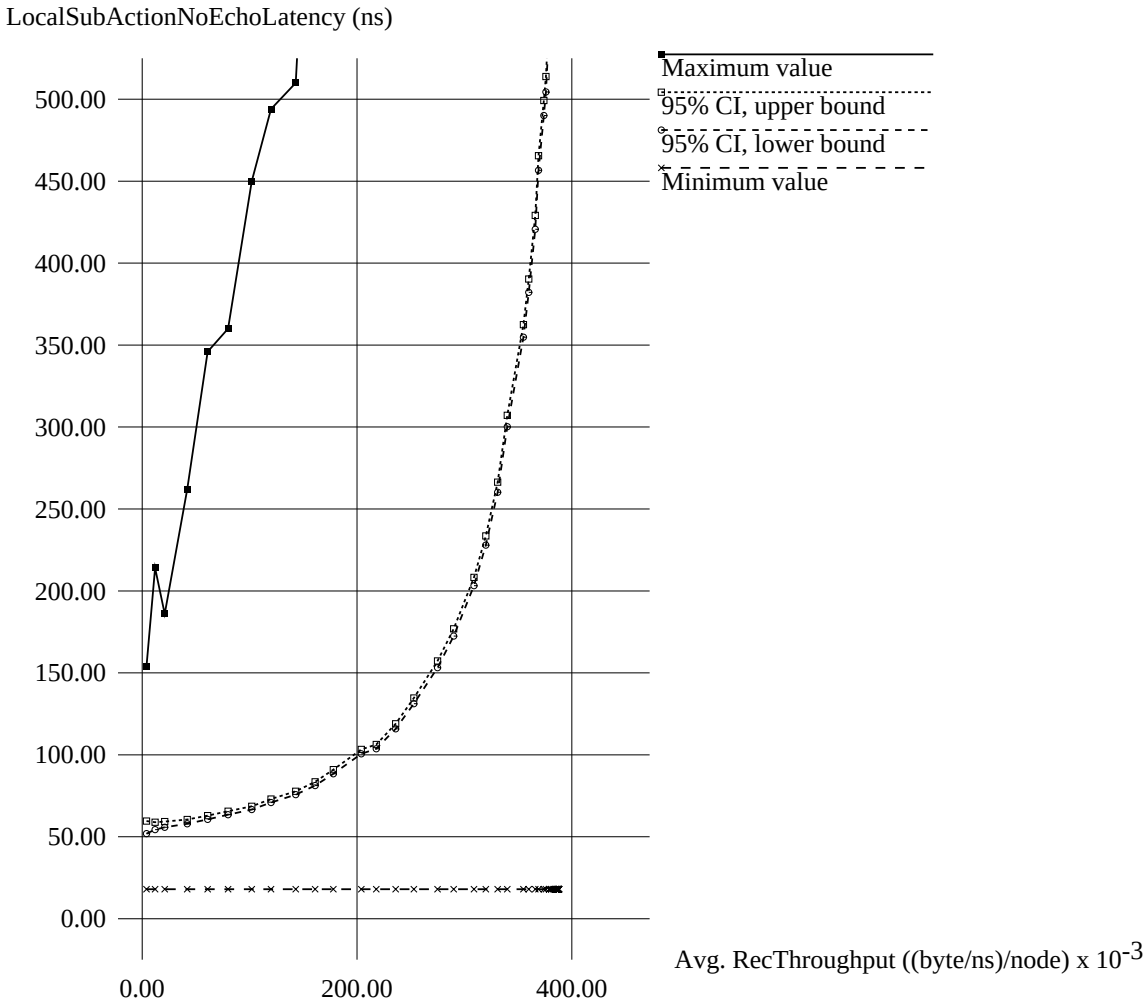


Figure 6.4: Statistical properties of the simulation-results (Uniform, 4 nodes, no flow control, mixed packets).

sults in [Scott et.al., 1992], though the actual measurements differ slightly. Compared to [Scott et.al., 1992], the graph 6.3 indicate a slightly better performance of uniform SCI-rings with 4 nodes without flow control. The intersection points between the 16byte case and the mixed case, and the mixed case and the 80byte case, in graph 6.3 are compared to [Scott et.al., 1992] in table 6.1.

Statistical properties of LocalSubActionNoEchoLatency, graph 6.4

To indicate the reliability of the measurements presented so far, figure 6.4 display some statistical properties associated with the LocalSubActionNoEchoLatency in the mixed case. The graph will show the following statistical properties of the LocalSubActionNoEcho-Latency:

- The relationship between average RecThroughput and the **minimum LocalSub-ActionNoEchoLatency sampled** during simulation.

- The relationship between average RecThroughput and the **maximum LocalSubActionNoEchoLatency sampled** during simulation.
- The relationship between average RecThroughput and the **95% confidence interval (C.I.) for the population mean** of LocalSubActionNoEchoLatency. The lower and upper bound of the confidence interval is shown. This illustrates that there is a 95% probability that the **population mean** of LocalSubActionNoEchoLatency lay between the lower and upper bound.

The abovementioned **population mean** refers to the **true, yet unknown,** mean value of LocalSubActionNoEchoLatency within the simulated system. Through simulation an estimate of the population mean have been calculated and presented i graph 6.1 through 6.3. The 95% confidence interval shown in graph 6.4 will therefore indicate whether this estimate are plausible.

The sample space, from which the estimate of the mean-value of LocalSubActionNoEchoLatency was calculated, was all the send-packets received by any node-interface during simulation. In the 80byte case the size of the sample space varied from 151 to 15955 depending the load, in the mixed case the size varied from 288 to 27319, and in the 16byte case it varied from 400 to 28583. The sample size is smallest when the load is low and the highest when the load high.

If we consider the 95% confidence interval in graph 6.4, we observe that the interval is between $\pm 0.6\%$ and $\pm 6.8\%$ of the estimated mean value presented in graph 6.2.1b and 6.3. In the 80byte case the 95% confidence interval is between $\pm 1\%$ and $\pm 1.5\%$ and in the 16 byte case it is between $\pm 0.6\%$ and $\pm 3\%$ of the estimated mean value (80byte and 16byte case is not shown in graph 6.4).

It may come as a surprise that the confidence interval does not vary more than is the case. This is reasonable though because simulation within the same set (80byte, mixed and 16byte) were performed for same amount of simulated time. At small loads the number of packets were small so the sample size was small too, which would imply a higher standard deviation and a broader confidence interval, but at the same time the traffic was so small that packets seldom experienced delay in the output-queues or the bypass-queues. The sampled values at low loads therefore did not diverge much. When the load grew higher the number of packets increased and thereby the sample space increased. A larger sample space usually imply a smaller standard deviation and a narrower confidence interval, but at the same time the traffic had increased so the packets more often experience a delay in the output-queue and the length of this delay is not deterministic because no flow control mechanism was used. Therefore the sampled values was spread a lot more.

We will also observe in graph 6.4 that the minimum sample of LocalSubActionNoEchoLatency is the same for any RecThroughput, and this apply to other cases as well. This is reasonable because it represents the situation where a node transmits to its downstream neighbour. The maximum value of LocalSubActionNoEchoLatency sampled during simulation seem to increase as the average RecThroughput increase, though in a non-monotonic fashion. This is also reasonable because when RecThroughput increase, the number of packets in the ring increase, and it becomes more likely that a node is blocked for a considerable time. Consequently, send-packets spend more time in the output-queues compared to when RecThroughput was lower.

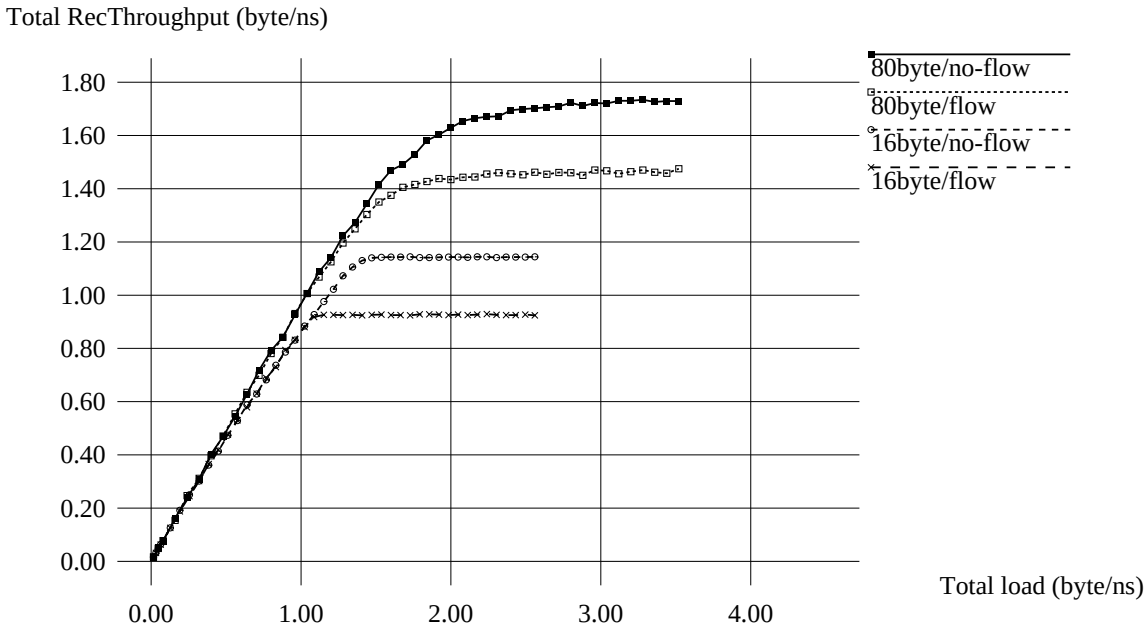


Figure 6.5: Throughput as a function of load (Uniform, 4 nodes, no flow control versus flow control).

6.2.2 Results related to uniform SCI-ring with 4 nodes, standard SCI flow control

This section will present and discuss results that are particular to uniform SCI-rings *with flow control*. The thorough discussion of RecThroughput and latency which were carried out on section 6.2.1 related to the graphs 6.1, 6.2 and 6.3 has been omitted, because the results described there apply to uniform SCI-rings *with flow control* as well. Still, there are quantitative differences between SCI-rings with flow control and SCI-rings without flow control, so this section will focus upon these differences. The results will be presented in the following order:

- Throughput as a function of load for the whole ring, comparison of flow control to no flow control. Graph 6.5.
- Latency as a function of load for the whole ring, comparison of flow control to no flow control. Graph 6.6.
- The relationship between throughput and latency for the whole ring, comparison of flow control to no flow control. Graph 6.7.

Total RecThroughput as a function of total load, graph 6.5

This graph shows the total RecThroughput as a function of total load when the send-packets are either 16byte or 80byte, and the ring either uses the standard SCI flow control mechanism or no flow control at all. Total load and total RecThroughput is specified in $byte/ns$ along the x-axis and y-axis respectively.

The interesting thing here is to compare total RecThroughput in the two cases **80byte/flow** and **80byte/no-flow**, and the two cases **16byte/flow** and **16byte/no-flow**. If we first

consider 80byte send-packets, we observe that the total RecThroughput in the flow case and the no-flow case are approximately equal when total load is less than $1.25 byte/ns$. When total load exceeds $1.25 byte/ns$, the total RecThroughput of the flow case and the no-flow case differ significantly as they approach different maximum values. The maximum total RecThroughput of the ring without flow control it is approximately $1.70 byte/ns$, while in the ring with flow control it is approximately $1.45 byte/ns$, representing a 15% reduction.

If we then consider 16byte send-packets we will observe almost the same relationship as in the 80byte case. When total load is less than $1.10 byte/ns$, the total RecThroughput in the flow case and the no-flow case are approximately equal. When total load exceed $1.1 byte/ns$ the total RecThroughput of the flow case and the no-flow case differ significantly, and the maximum total RecThroughput of the ring without flow control is approximately $1.15 byte/ns$, while in the ring with flow control it is $0.95 byte/ns$, which represent a 17% reduction.

This indicate some properties of the total RecThroughput in uniform SCI-rings. When load is less than a certain limit **L**, the total RecThroughput is not affected by the flow control mechanism, and is approximately equal to the total RecThroughput if no flow control was used. When load exceed **L**, the total RecThroughput in the ring with flow control will be less than the total RecThroughput in the ring without flow control. The maximum total RecThroughput in the ring with flow control will also be less than the maximum total RecThroughput in the ring without flow control.

It is reasonable that the total RecThroughput is approximately equal when load is low, because few packets will circulate the ring and a node-interface is rarely busy bypassing packets or recovering from a prior transmission. A ring using the SCI flow control mechanism is filled with Go-idles when the load is low, and a node-interface does not have to wait long for a passing Go-idle after a packet is inserted into the output-queue. If the ring does not use flow control, the node-interface simply transmits the packet without waiting for any Go-idle. Therefore the delay before a packet can be transmitted is almost equal in rings with flow control and rings without flow control. The output-queues will rarely fill up, and the node-interfaces transmit at the same rate as the application processes generate packets.

It is also reasonable that maximum total RecThroughput is less in SCI-rings with flow control than in SCI-rings without flow control. The following rationale explain why:

SCI-rings without flow control: Assume a uniform load and traffic pattern, and a load sufficiently high so that the output-queues are non-empty at all times. A node-interface in this ring will be constantly busy, either bypassing packets, transmitting packets from the output-queue or recovering from a previous transmission (Note that **recover** means to empty the bypass-queue after a transmission).

In an SCI-ring without flow control, a node-interface can transmit a packet only if the bypass-queue is empty. The node-interface goes through a cycle of stages and can transmit packets only in some of them. Let us first assume that the bypass-queue is empty and before the packet can be transmitted, the packet currently bypassed has to come to an end. Then the packet in the output-queue is transmitted, and at the same time incoming symbols are stored in the bypass-queue. When the transmission is done, the node-interface goes to recovery stage, in which the node-interface tries to empty the bypass-queue by transmitting packets from it, but packets may well enter the bypass-queue at the same rate as the packets are leaving. Therefore considerable time can be spent recovering from a transmission, in fact the recovery stage will

not end *unless* the node-interface receives a packet addressed to it. When receiving a packet, the node-interface strips it from the ring and passes idle-symbols to the bypass-queue, and a sequence of idle-symbols are collapsed into one. In this way the bypass-queue is emptied, or at least shrunken in size, and after the node-interface has received one or several packets addressed to it, the bypass-queue is once again empty and the node-interface can again transmit a packet from the output-queue.

The details are not important, but what *is* important, is that a node-interface in an SCI-ring without flow control at high load, can only transmit packets from the output-queue if it also receives packets addressed to it. If the node-interface does not receive a packet, it will be stuck in a never-ending recovery stage. In a uniform SCI-ring with 4 nodes, one out of three packets on the input-link is addressed to the node-interface, and this will enable the node-interface to transmit new packets.

SCI-rings with flow control: Assume a uniform load and traffic pattern and a load sufficiently high so that the output-queues are non-empty at all times. As in the SCI-ring without flow control, a node-interface will be constantly busy, either bypassing packets, transmitting packets from the output-queue or recovering from a previous transmission. In the flow-controlled ring, a node-interface can also be blocked for some time, awaiting a Go-idle. This blocked-stage is the crux of the SCI-flow control mechanism, because it stops a node-interface from transmitting send-packets from the output-queue, as long as another node-interface in the ring is trying to empty its bypass-queue.

A node-interface in an SCI-ring with flow control can only transmit packets from the output-queue if the bypass-queue is empty *and* a Go-idle has passed. The node-interface will go through a cycle of stages, and can only transmit in some of them. Let us assume that the bypass-queue is empty. The node-interface is most likely busy bypassing a packet (without storing it in the bypass-queue), and the node-interface must wait until this packet has come to an end, and then it goes to the blocked stage where it awaits a Go-idle. When it observes a passing Go-idle, perhaps wedged in between the packets, it appends the packet. At the same time incoming packets are stored in the bypass-queue. When the transmission is done, the node-interface goes to recovery stage, where it tries to empty its bypass-queue by transmitting packets from it, and between packets the node interface emits NoGo-idles. In this way other node-interfaces are blocked until the bypass-queue is empty.

When the node-interface is recovering it will emit NoGo-idles between the packets transmitted from the bypass-queue, and in this way end the recovery stage. Unfortunately the NoGo-idles affect the whole ring, not only the node-interfaces which cause the bypass-queue of the first node-interface to fill up, but also node-interfaces that communicate without affecting the first node-interface. Communication which do not pass the first node-interface, is also stopped as a result of the NoGO-idles and some bandwidth is eventually lost. Therefore the maximum total RecThroughput is less than in the no-flow case. It is also important to note that a node-interface in an SCI-ring with flow control does not send out packets during all stages. In the blocked-stage, the node-interface may well emit a long sequence of NoGO-idles, because another node-interface in the ring tries to empty its bypass-queue.

This explains why the maximum total RecThroughput is less when flow control is used in uniform SCI-rings. The reduction in maximum total RecThroughput is partly caused

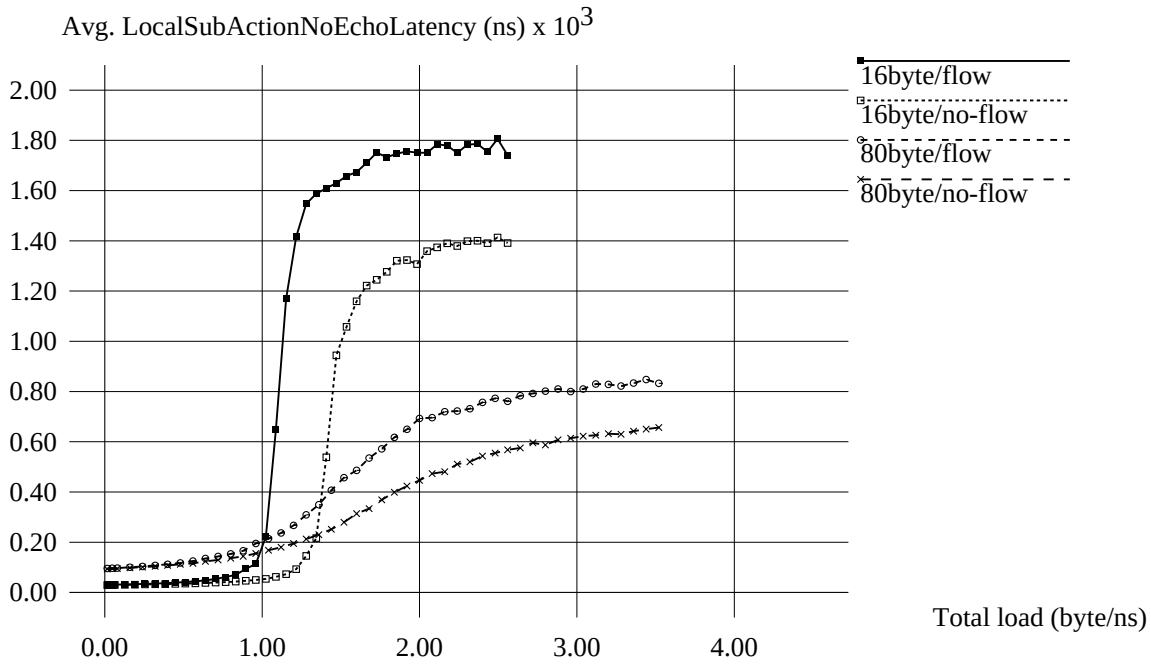


Figure 6.6: Latency as a function of load (Uniform, 4 nodes, no flow control versus flow control)

by the flow control mechanism (blocking non-interfering node-interfaces) and partly caused by how a uniform SCI-ring without flow control behave (node-interfaces are always sending something).

This seem to speak against the use of a flow control mechanism in the SCI-ring, but there are other important aspects too. The SCI-standard emphasize forward progress and consequently, it must guarantee fair distribution of bandwidth. The uniform SCI-ring without flow control may seem fair, because all nodes transmit approximately the same amount, but this is caused by the uniform load and traffic pattern. As we will se later, things will be different in a non-uniform load and traffic pattern.

Average LocalSubActionNoEchoLatency as a function of total load, graph 6.6

This graph shows the average LocalSubActionNoEchoLatency as a function of total load when the send-packets are either 16byte or 80byte, and the ring either use the standard SCI flow control mechanism or no flow control at all. Total load is specified in $byte/ns$ along the x-axis and average LocalSubActionNoEchoLatency is specified in ns along the y-axis.

The interesting thing here is to compare average LocalSubActionNoEchoLatency in the two cases **80byte/flow** and **80byte/no-flow**, and the two cases **16byte/flow** and **16byte/no-flow**. If we first consider 80byte send-packets, we observe that the average LocalSubActionNoEchoLatency in the flow case and the no-flow case are approximately equal when total load is less than $0.5byte/ns$. When total load exceeds $0.5byte/ns$, the average LocalSubActionNoEchoLatency of the flow case and the no-flow case diverge and will approach different maximum values. The maximum average LocalSubActionNoEcho-Latency of the ring without flow control it is approximately $700ns$, while in the ring with

flow control is approximately $850ns$, which represent an increase of 21%.

If we then consider 16byte send-packets, we will observe almost the same relationship as in the 80byte case. When total load is less than $0.5byte/ns$, the average LocalSubActionNoEchoLatency in the flow case and the no-flow case are approximately equal. When total load exceeds $0.05byte/ns$, the average LocalSubActionNoEchoLatency of the flow case and the no-flow case differ significantly, and the maximum average LocalSubActionNoEchoLatency of the ring without flow control is approximately $1400ns$, while in the ring with flow control it is $1800ns$, which represent an increase of 28%. We also observe that the rapid increase in average LocalSubActionNoEchoLatency (eg. in the 16byte case when the when total load exceeds $1.0byte/ns$) takes place when the maximum total RecThroughput is reached in graph 6.5.

This indicate some properties of the average LocalSubActionNoEchoLatency in real-life SCI-rings. When load is less than a certain limit **L**, the average LocalSubActionNoEchoLatency is approximately equal, regardless whether flow control is used or not. If load exceeds **L**, the average LocalSubActionNoEchoLatency will increase a lot faster when flow control is used, than it would do if no flow control is used. The maximum average LocalSubActionNoEchoLatency is also significantly higher when flow control is used.

It is reasonable that the average LocalSubActionNoEchoLatency is approximately equal at low loads, and that the average LocalSubActionNoEchoLatency increase faster when flow control is used when load increase. The following will explain why:

- When load is low, few packets circulate the ring. In an SCI-ring with flow control, the ring will be filled with Go-idles (except for an occasional packet) because a node-interface rarely has to empty the bypass-queue after a transmission. A packet inserted into the output-queue can therefore be transmitted almost immediately, because the bypass-queue is rarely full and Go-idles fill the ring.

  In an SCI-ring without flow control, a packet inserted into the output-queue can also be transmitted almost immediately because all that is required is an empty bypass-queue.

- When load increase further, more packets circulate the ring. A node-interface will therefore experience longer and more recovery stages.

  In an SCI-ring without flow control a node-interface can transmit if the bypass-queue is empty and the recovery stage ends whenever the node-interface receives a packet. In an SCI-ring with flow control a node-interface can transmit if the bypass-queue is empty *and* a Go-idle passes, and the recovery stage ends whenever the node-interface receives a packet addressed to it or a sequence of idles. Consequently, the LocalSubActionNoEchoLatency will be higher when flow control is used than when no flow control is used, because of the additional requirement of the passing Go-idle.

It is also reasonable that the maximum average LocalSubActionNoEchoLatency is higher when flow control is used. From discussion of figure 6.5 we know that the maximum total RecThroughput is lower when flow control is used and this mean that less bandwidth is available when send-packets are transmitted. Consequently, when flow control is added to an SCI-ring, more time will be spent transmitting the same amount as before.

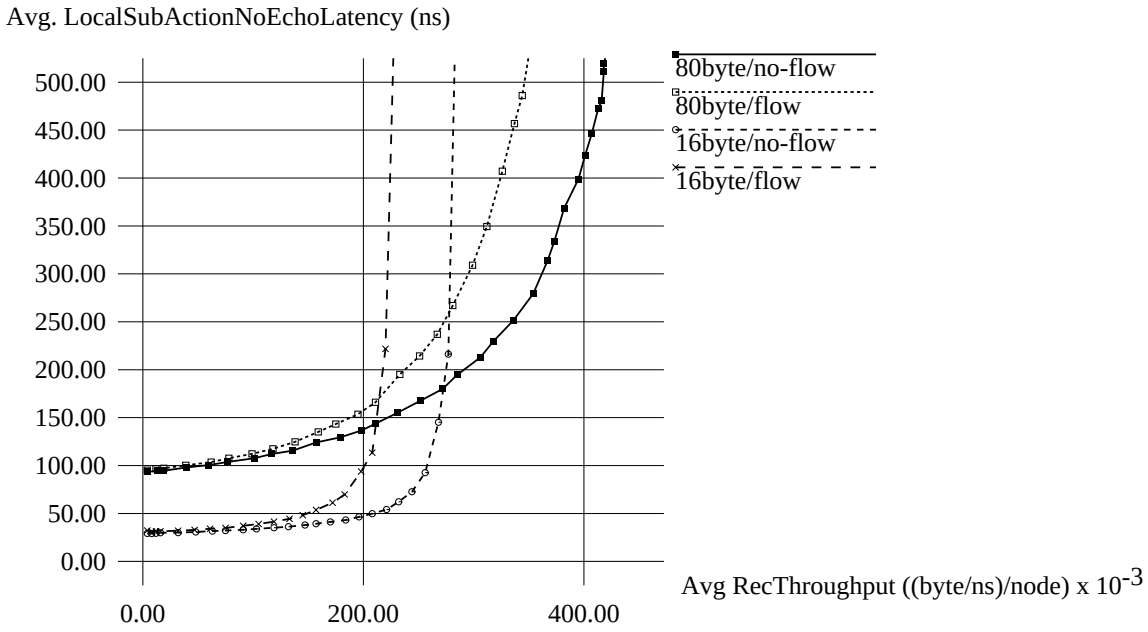


Figure 6.7: The relationship between throughput and latency (Uniform, 4 nodes, no flow control versus flow control)

The relationship between average RecThroughput and average LocalSubAction-NoEchoLatency, graph 6.7

This graph shows the relationship between average RecThroughput and average Local-SubActionNoEchoLatency, when the send-packets are either 16byte or 80byte, and the ring either use the standard SCI flow control mechanism or no flow control at all. The Rec-Throughput is specified in $(byte/ns)/node$ along the x-axis and the latency is specified in **(ns)** along the y-axis.

If we first consider 80byte send-packets, we observe that the average LocalSubAction-NoEchoLatency is almost identical in the flow case and the no-flow case, when average RecThroughput is less than $0.05(byte/ns)/node$. When average RecThroughput exceed $0.05(byte/ns)/node$, the flow case and the no-flow case diverge quickly and the average LocalSubActionNoEchoLatency of the flow case is higher than the no-flow case. If we then consider 16byte send-packets, we observe that the average LocalSubActionNoEchoLatency is almost identical in the flow case and the no-flow case, when average RecThroughput is less than $0.10(byte/ns)/node$. When average RecThroughput exceed $0.1(byte/ns)/node$, the flow case and the no-flow case diverge quickly and the average LocalSubActionNoEcho-Latency of the flow case is higher than the no-flow case.

This indicate that an SCI-ring without flow control, has a higher average RecThroughput and a lower average LocalSubActionNoEchoLatency than an identical SCI-ring with the flow control added. When we consider uniform SCI-rings under similar conditions, the performance of the average node in rings without flow control is higher than in rings with flow control.

This is a reasonable conclusion, if we consider the results presented in graph 6.5 and 6.6:

- From graph 6.5 we know that the total RecThroughput of the **16byte/no-flow** case and the **16byte/flow** case is almost identical when the total load is less than $0.5 byte/ns$. As a result the total RecThroughput is less than $0.5 byte/ns$. From graph 6.6 we know that the average LocalSubActionNoEchoLatency is almost identical when total load is less than $0.5 byte/ns$. Therefore the **16byte/flow** and the **16byte/no-flow** is almost identical when average RecThroughput is less than $0.125 (byte/ns)/node$ (SCI-ring with 4 nodes).

- When total load exceeds $0.5 byte/ns$, but remain less than $1.1 byte/ns$, the total RecThroughput of the **16byte/no-flow** case and the **16byte/flow** case continue to increase and remain close in graph 6.5. The average LocalSubActionNoEchoLatency on the other hand, increase more in the **16byte/flow** case than in the **16byte/no-flow** case, as shown in graph 6.6. Therefore the **16byte/flow** case and the **16byte/no-flow** case diverge in graph 6.7, when average RecThroughput is in the interval from $0.125 (byte/ns)/node$ to $0.225 (byte/ns)/node$.

- When total load exceeds $1.1 byte/ns$, the average LocalSubActionNoEchoLatency of the **16byte/flow** case and the **16byte/no-flow** case continue to increase, as shown in graph 6.5. In graph 6.6 the total RecThroughput of the **16byte/flow** case will not increase further, while the **16byte/no-flow** case will continue to increase until total load exceed $1.5 byte/ns$. Therefore the **16byte/flow** case lay above the **16byte/no-flow** in graph 6.7.

These results can be compared to results presented in [Scott et.al., 1992], and graph 6.7 is directly comparable to a graph there. We will focus upon the **80byte/flow** case and the **16byte/flow** case, as the other two have been discussed previously in section 6.2.1.

The graph 6.7 resembles the results presented in [Scott et.al., 1992], though there are quantitative differences. Compared to [Scott et.al., 1992], the graph 6.7 indicate a slightly worse performance of uniform SCI-rings with 4 nodes using flow control and when 16byte send-packets are used. When 80byte send-packets are considered the results in graph 6.7 indicate approximately the same performance as that presented in [Scott et.al., 1992].

6.2.3 Results related to SCI-rings of size 16

This section will present and discuss results from the simulation of uniform SCI-rings with 16 nodes, which either use the SCI flow control mechanism or not. Results that are directly related to the larger ring structure will be emphasized, and results from section 6.2.1 and 6.2.2 will be used in order to compare 16 node SCI-rings to 4 node SCI-rings. The following results will be discussed in this section:

- Throughput as a function of load for the whole ring, comparison of 4 node SCI-rings to 16 node SCI-rings. Graph 6.8.

- Latency as a function of total load for the whole ring, comparison of 4 node SCI-rings to 16 node SCI-rings. Graph 6.9.

- The relationship between throughput and latency for the whole ring, comparison of 4 node SCI-rings to 16 node SCI-rings. Graph 6.10.

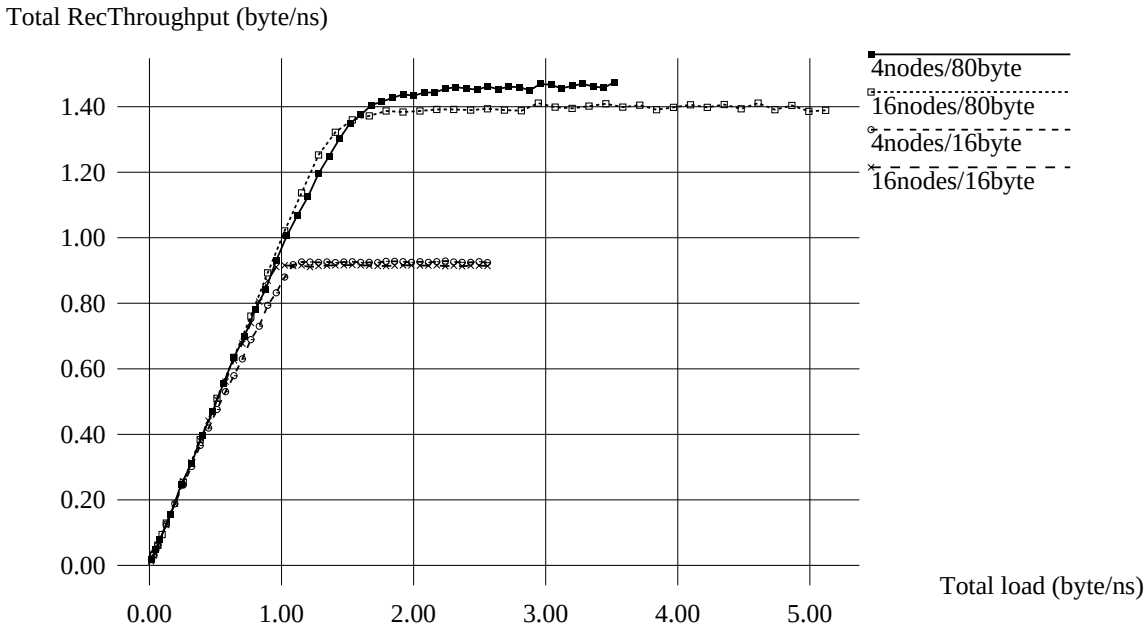


Figure 6.8: Throughput as a function of load (Uniform, 4 node versus 16 node, flow control)

Simulation results indicate that adding flow control to a uniform 16 node SCI-ring will affect the RecThroughput and latency in the same as it will in a uniform 4 node SCI-ring. When load is low, the flow control mechanism does not affect the total RecThroughput, and total RecThroughput is equal regardless whether flow control is used or not. When load is higher, the total RecThroughput is less when flow control is used and the maximum total RecThroughput is reduced with 20% when either 16byte or 80byte send-packets are used. The average LocalSubActionNoEchoLatency is not affected when load is low, but will be affected when load is high. In general the average LocalSubActionNoEchoLatency is higher when flow control is added to the ring, and in particular the maximum average LocalSubActionNoEchoLatency is increased with 37% when 80byte packets are used and 28% when 6byte packets are used. For further details on flow control in uniform SCI-rings, refer to the discussion of flow control in uniform 4 node SCI-rings in section 6.2.2.

Total RecThroughput as a function of total load, graph 6.8

This graph shows the total RecThroughput as a function of total load when the send-packets are either 16byte or 80byte, the ring uses flow control and the ring contains either 4 nodes or 16 nodes. Total load and total RecThroughput is specified in $byte/ns$ along the x-axis and y-axis respectively.

The interesting thing here is to compare total RecThroughput in the two cases **4nodes/80byte** and **16nodes/80byte**, and the two cases **4nodes/16byte** and **16nodes/16byte**. If we first consider 80byte send-packets, we observe that the total RecThroughput in the 4 node ring and the 16 node ring is almost identical when total load is less than $1.5byte/ns$. When total load exceeds $1.5byte/ns$, the total RecThroughput in the 4 node ring is higher than the total RecThroughput in the 16 node ring. The maximum total RecThroughput in the 4 node ring is 4% higher than the maximum in the 16 node ring.

If we then consider 16byte send-packets, we observe that the total RecThroughput in

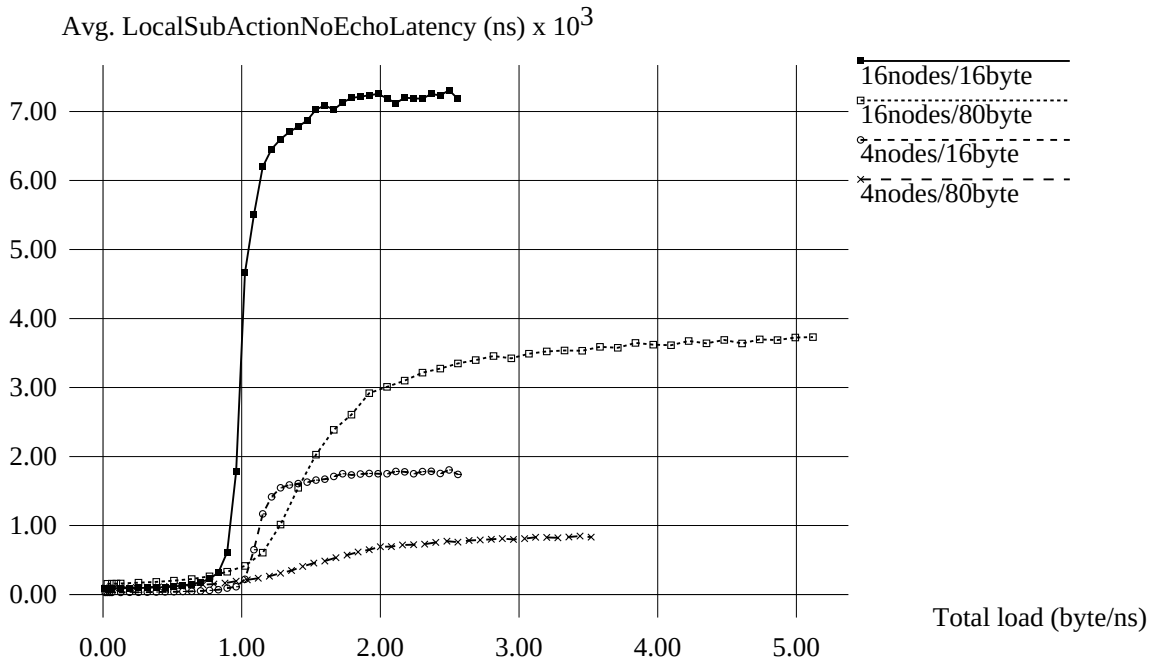


Figure 6.9: Latency as a function of load (Uniform, 4 node versus 16 node, flow control).

the 16 node ring is slightly higher, no more than 9%, than the total RecThroughput of the 4 node SCI-ring, when total load is less than $1.1 byte/ns$. When total load exceed $1.1 byte/ns$ the situation is reversed and the total RecThroughput of the 4 node ring is slightly higher than in the 16 node ring. The maximum total RecThroughput of the 4 node ring is 1% higher than the maximum total RecThroughput of the 16 node ring.

Assuming that the ring display a uniform load and traffic pattern, and that it uses flow control, the above observation indicate that the total RecThroughput as a function of load in a 4 node SCI-ring and a 16 node SCI-ring does not differ more than 10%, and that the maximum total RecThroughput of 4 node ring in some cases is higher than in the 16 node ring..

This result is somewhat disappointing, because it means that the total RecThroughput does not increase as the number of nodes in a uniform SCI-ring increase. Consequently the SCI-ring structure cannot be scalable up to 16 nodes.

Average LocalSubActionNoEchoLatency as a function of total load, graph 6.9

This graph shows the average LocalSubActionNoEchoLatency as a function of total load when the send-packets are either 16byte or 80byte, the ring use flow control and the ring contain either 4 nodes or 16 nodes. Total load is specified in $byte/ns$ along the x-axis and LocalSubActionNoEchoLatency is specified in ns along the y-axis.

The interesting thing here is to compare average LocalSubActionNoEchoLatency in the two cases **16nodes/16byte** and **4nodes/16byte**, and the two cases **16nodes/80byte** and **4nodes/80byte**. If we consider 16byte send-packets, we observe that the average LocalSubActionNoEchoLatency in the 4 node ring is less than in the 16 node ring at any load. When load is low, less than $0.5 byte/ns$, the average LocalSubActionNoEchoLatency is approximately 300% higher in the 16 node ring than in the 4 node ring, but when load

is higher, greater than $2.0 byte/ns$, the average LocalSubActionNoEchoLatency in the 16 node ring is approximately 400% higher than in the 4 node ring.

If we then consider 80byte send-packets we will make the same observation. In general, the average LocalSubActionNoEchoLatency is less in the 4 node ring than in the 16 node ring. When load is less than $0.5 byte/ns$, the average LocalSubActionNoEchoLatency is approximately 16% higher in the 16 node ring than in the 4 node ring. When load is greater than $2.0 byte/ns$, the average LocalSubActionNoEchoLatency is 450% higher in the 16 node ring than in the 4 node ring.

This indicate that the average LocalSubActionNoEchoLatency is higher in a 16 node SCI-ring than in a 4 node SCI-ring, assuming similar conditions. This apply to any given load, and in particular at high loads when the maximum average LocalSubActionNoEcho-Latency is approximately 4 times higher in the 16 node ring than in the 4 node ring.

It is reasonable that the average LocalSubActionNoEchoLatency is higher in the 16 node ring than in the 4 node ring, because the ring-structure and the total output-queue size is larger. The LocalSubActionNoEchoLatency includes the time from a send-packet is inserted into the output-queue, until it has been received by the receiver node, and when the ring-size increase, the LocalSubActionNoEchoLatency will also increase. It is also reasonable that the maximum LocalSubActionNoEchoLatency of the 16 node ring is approximately 4 times higher than the 4 node ring because the total output-queue space is 4 times larger in the 16 node ring than in 4 node ring, and because the maximum LocalSubActionNoEchoLatency depends on total output-queue space.

The relationship between average RecThroughput and average LocalSubAction-NoEchoLatency, graph 6.10

This graph shows the relationship between average RecThroughput and average LocalSub-ActionNoEchoLatency, when the send-packets are either 16byte or 80byte, the ring use flow control and the ring contain either 4 nodes or 16 nodes. The RecThroughput is specified in $(byte/ns)/node$ along the x-axis and the latency is specified in **(ns)** along the y-axis.

The interesting thing here is to compare average LocalSubActionNoEchoLatency in the two cases **16nodes/16byte** and **4nodes/16byte**, and the two cases **16nodes/80byte** and **4nodes/80byte**. The results indicate that each node have a higher RecThroughput and a lower LocalSubActionNoEchoLatency in the 4 node ring than in the 16 node ring. If we with high throughput and low latency associate good performance, the average node in the smaller ring has a higher performance than the average node the larger ring.

6.2.4 Summary of results related to uniform load and traffic patterns

The following is summary of the main results related to uniform SCI-rings, results which will help us decide on Issue 4, Issue 5 and Issue 7 (chapter 3):

- The total **RecThroughput** equals the total load when total load is less than a certain limit **L**. This limit depends on the size of send-packets used in the ring, eg. when 16byte send-packets are used, $L = 1.25 byte/ns$.
- Maximum total **RecThroughput** is a stable value even when total load increase. The maximum value is approached when load exceed **L**.

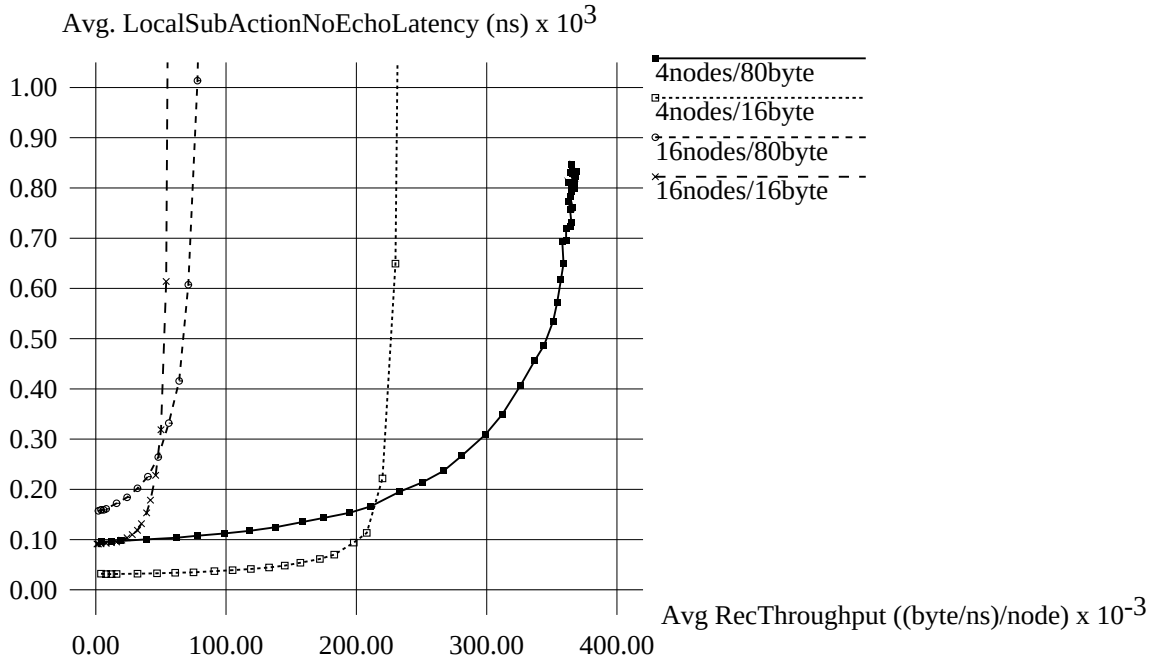


Figure 6.10: The relationship between throughput and latency (Uniform, 4 node versus 16 node, flow control)

- Maximum total **RecThroughput** depends on the size of send-packets used in the ring. Large send-packets gives a higher maximum total **RecThroughput** than smaller send-packets.
- The average **LocalSubActionLatency** and **LocalSubActionNoEchoLatency** increase as total load increase, but is bounded downward and upward. The lower bound depends on the minimum fixed delay in links and node-interfaces, and the size of send-packets used in the ring. The upper bound also depends on the minimum fixed delay in links and node-interfaces, but in addition it depends on the maximum output-queue size.
- Flow control will reduce the maximum total **RecThroughput**. When either 16byte or 80byte send-packets are used the reduction amounts to 15%.
- Flow control will increase the maximum average **LocalSubActionNoEchoLatency**. When 16byte send-packets are used in the ring the increase amounts to 28%.
- Increasing the number of nodes from 4 to 16 in a uniform SCI-ring, will not increase the total **RecThroughput** of the ring.
- Increasing the number of nodes from 4 to 16 in a uniform SCI-ring, will increase the average **LocalSubActionNoEchoLatency** in general, and the *maximum* average **LocalSubActionNoEchoLatency** in particular. When 16byte send-packets are used, the increase of the maximum average LocalSubActionNoEchoLatency amounts to 400%.

- The performance of the average node is higher in a 4-node SCI-ring than in a 16-node SCI-ring.
- The SCIsim-simulator produce results which resemble those presented in [Scott et.al., 1992] when the same configuration are simulated. Properties of the SCI-ring related to ring-size, packet-size and flow-control (as indicated above in this list) corresponds to those indicated in [Scott et.al., 1992].

6.3 Hot-sender load and traffic pattern in single SCI-rings

This section will present and discuss results from the simulation of single SCI-rings where the load and traffic pattern is referred to as **hot-sender**. These results will help us decide on Issue 4, Issue 6 and Issue 7 as presented in chapter 3. The following list describes the overall conditions assumed during simulation:

- **Topology:** Single ring.
- **Size:** 4 nodes or 16 nodes.
- **Load and traffic pattern:** hot-sender.
- **Transmitter-stage:** SCI flow control, no flow control.
- **Send-packet size:** Mixed.

Except from the hot-node, where the load was fixed to $4.2 byte/ns$, the load was increased from one simulation to another - starting with a value close to zero and going up to a level where the throughput and latency measurements had stabilized. The length of one simulation was determined after some preliminary simulations (Refer to section 5.3) and for the 4-node ring and the 16-node ring the simulation time was $1200000ns$ and $2500000ns$ respectively. The remaining parameter-values which were assumed during simulation, can be found in table 5.1 in section 5.1.2. The nodes in the smallest ring are labeled **P0 through P3** for convenience, and **P0 will refer to the hot node**, P1 to its immediate downstream neighbour etc.

In order to indicate the throughput of each node in a hot-sender SCI-ring, this section will emphasize the **TransThroughput** as defined in section 5.2. This throughput-measurement includes all send-packets transmitted by a node-interface.

In order to indicate the latency of each node, this section will emphasize **LocalSubActionNoEchoLatency**. Both average and maximum values of LocalSubActionNoEchoLatency will be presented for each node.

Section 6.3.1 will present and discuss results from the simulation of hot-sender SCI-rings with 4 nodes, where the ring either use the SCI flow control mechanism or not. A 16 node SCI-ring has also been simulated under the assumption of hot-sender load and traffic pattern, but the results resemble closely the results of the 4 node ring, so results related to the 16 node ring will not be presented.

Section 6.3.2 will give a summary of the main results related to hot-sender.

6.3.1 Results related to hot-sender in SCI-rings with 4 nodes

This section will present and discuss the simulation-results in the following order:

- Throughput as a function of load for each individual node, comparison of no-flow control to flow control. Graph 6.11a-b.
- Latency as a function of load for each individual node, comparison of no-flow control to flow control. Graph 6.12a-b.
- The relationship between throughput and latency for each individual node, comparison of no-flow control to flow control. Graph 6.13a-b.
- Worst case latency as a function of load for the downstream neighbour P1, comparison of no-flow control to flow control. Graph 6.14.

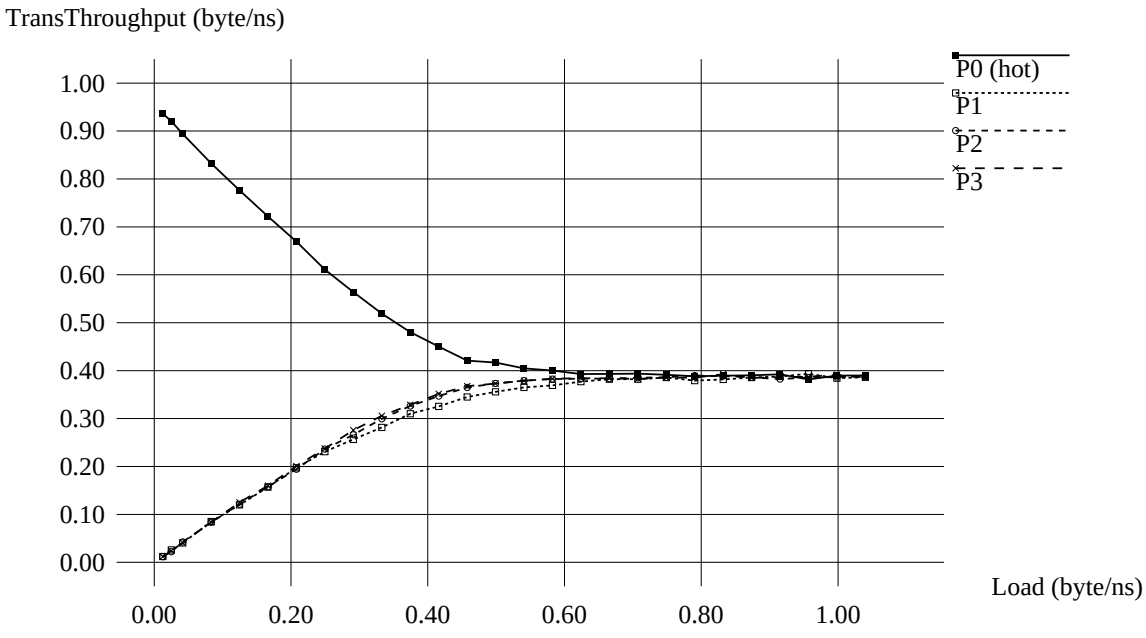
TransThroughput as a function of load, graph 6.11a-b

This figure contains two graphs, one related to SCI-rings without flow control (graph 6.11a) and the other related to SCI-rings with flow control (graph 6.11b). Each graph shows for each node P0-P3, the TransThroughput as a function of load in P1-P3. Note that the load in P0 is fixed. Load and throughput is specified in $byte/ns$ along the x-axis and y-axis respectively.

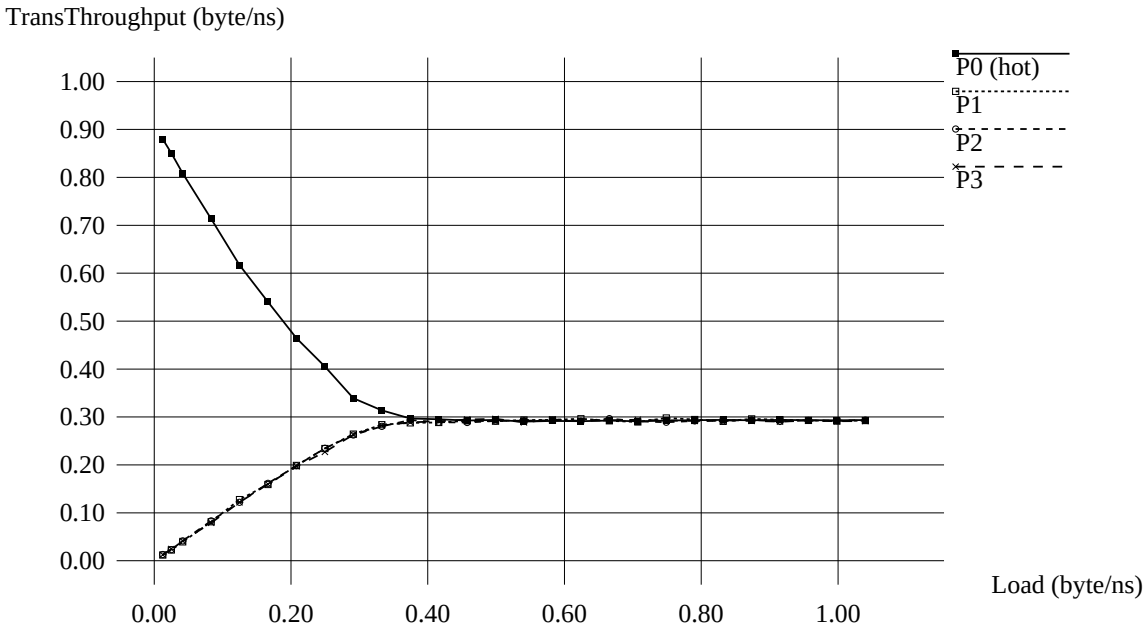
We observe in graph 6.11a that the TransThroughput of P0 is significantly different from that of node P1-P3. When load in P1-P3 approach $0.0byte/ns$, the TransThroughput of P0 will approach $0.95byte/ns$. When load in P1-P3 increase, the TransThroughput of P0 will decrease, while the TransThroughput of P1-P3 will increase. The TransThroughput of all nodes converge when load in P1-P3 increase, and when it exceeds $0.60byte/ns$ the TransThroughput of P0-P3 is approximately equal. A more careful observation reveal that node P1-P3 experience an approximately linear growth of TransThroughput when load increase in the interval from $0.0byte/ns$ to $0.20byte/ns$, while the TransThroughput of P0 decrease approximately linearly in the same interval. When load in P1-P3 exceeds $0.20byte/ns$, the TransThroughput start to level out, approaching $0.39byte/ns$. We will also observe that the TransThroughput of P1, the immediate downstream neighbour of the hot node P0, is slightly lower than the TransThroughput of P2 and P3, when load is in the interval from $0.3byte/ns$ to $0.60byte/ns$.

The results presented in graph 6.11a indicate that in an SCI-ring of size 4, displaying a hot-sender load and traffic pattern, the hot node has a higher TransThroughput at low loads, than the remaining nodes. When load in the remaining nodes increase (the load in P1-P3 are equal) the TransThroughput of P1-P3 increase, while the TransThroughput of P0 decrease, and will approach the same stable value. When load in each node P1-P3 exceed $0.60byte/ns$, all nodes have almost the same TransThroughput. Node P1, the immediate downstream neighbour of P0 has the same transmitted throughput as node P2-P3, except when load lay in the interval $0.3byte/ns$ to $0.65byte/ns$ where P1 transmit slightly less.

The results in graph 6.11a and the above conclusion is reasonable assuming a hot-sender load and traffic pattern. When load in P1-P3 is very low, P0 has the ring entirely to itself, and will transmit packets as fast as it can. Because very few send-packets from P1-P3 are passing P0, and almost all echo-packets entering P0 on the input-link are addressed to P0 itself, P0's output-link will transmit (almost) only send-packets from P0's own output-queue. The average size of send-packets transmitted from P0 is 0.6*16 + 0.4*80 = 41.6byte



(a) No flow control



(b) Flow control

Figure 6.11: Throughput as a function of load (Hot-sender, 4 nodes, mixed packets).

and an idle-symbol will be appended. Therefore the TransThroughput of P0 will approach $41.6/(41.6+2) = 0.95 byte/ns$ when load in P1-P3 is very low. This corresponds to the result in graph 6.11a.

When the load in P1-P3 increase from a small value, P1-P3 will transmit more packets and this will in turn increase their TransThroughput. The hot node will have to bypass more packets to P2 and P3 than before, and consequently the TransThroughput of P0 will decrease. P1-P3 are able to increase their TransThroughput, despite P0's constant flow of packets, because all nodes generate packets with uniform destination addresses. As explained in section 6.2 (where flow control were compared to no-flow control in a uniform SCI-ring), a node-interface in an SCI-ring without flow control can transmit only if also receives packets addressed to it. Therefore P1 is not more affected by the hot node than P2 and P3 in terms of TransThroughput, though the TransThroughput of P1 is slightly less than that of P2-P3 when load is in the interval 0.30 to $0.60 byte/ns$.

When the load in P1-P3 is very high, greater than $0.60 byte/ns$, the output-queues will be constantly filled with packets, and P1-P3 will start to behave like P0 and try to transmit as much as possible. Therefore the hot-sender load and traffic pattern will resemble the uniform load and traffic pattern. Consequently, the TransThroughput of P0-P3 is approximately equal, and will approach the maximum average throughput observed the uniform SCI-ring shown in graph 6.1b.

In graph 6.11b, we observe results that resemble those results presented in graph 6.11a, with some exceptions. Where the flow control and the no flow control differ, is in the TransThroughput of node P1. In graph 6.11b we observe the same TransThroughput of P1-P3 at any load, whereas in the no flow control case the TransThroughput of node P1 was slightly less than that of node P2 and P3.

The TransThroughput of P1-P3 increases when load in P1-P3 increase, and the growth is almost linear when load is less than $0.20 byte/ns$. When load exceeds $0.20 byte/ns$, the TransThroughput of P1-P3 will level off, and when load exceeds $0.40 byte/ns$, it will have reached the same maximum value. The TransThroughput of P0 is very high at low loads, but decreases when load in P1-P3 increase, and will approach a stable value when load exceeds $0.40 byte/ns$. The TransThroughput of P0 is approximately equal to that of node P1-P3 when load is high.

The results in graph 6.11b indicate some properties of an SCI-ring, displaying a hot-sender load and traffic pattern. The hot node has a high TransThroughput when load in P1-P3 is low, whereas P1-P3 have a low TransThroughput. When load in P1-P3 increase, the TransThroughput of the hot node will decrease while the TransThroughput of the other nodes will increase. When load in P1-P3 is very high, the TransThroughput of all nodes are almost equal, approximately $0.29 byte/ns$.

The results in graph 6.11b and the above conclusion is reasonable assuming an SCI-ring with flow control displaying a hot-sender load and traffic pattern. When load is low, the other nodes transmit few packets and P0 has the ring almost entirely to itself. While there are few send-packets from P2 and P3 that are passing P0, and almost all echo-packets on P0's input-link are addressed to P0 itself, the bypass-queue rarely fills up and P0 rarely has to empty the bypass-queue after a transmission. The ring is therefore filled with Go-idles, except for the send-packets transmitted by P0. P0 is therefore able to transmit packets continuously.

When load in P1-P3 increase, P1-P3 will transmit more packets, and this will in turn increase their TransThroughput. The TransThroughput of P0 will decrease, because packets

from P2 and P3 have to pass through node P0. The flow control mechanism also eliminates the difference in TransThroughput between P1 and P2/P3.

When load is very high, the output-queues in P1-P3 are constantly filled with send-packets, and the behavior of P1-P3 will resemble the behavior of P0. Consequently the hot-sender load and traffic pattern starts to resemble the uniform, because all nodes transmit packets uniformly and have the same effective load. As a result, the TransThroughput of each node are approximately equal when load is high, and is almost identical to the maximum average throughput in a uniform 4-node SCI-ring.

It is also reasonable when load is high, that the TransThroughput of each node is reduced compared to a no-flow ring. The hot-sender load and traffic pattern resembles the uniform when load is high, and the flow control mechanism affects the whole ring. Refer to the discussion of figure 6.5 in section 6.2.2 for further details.

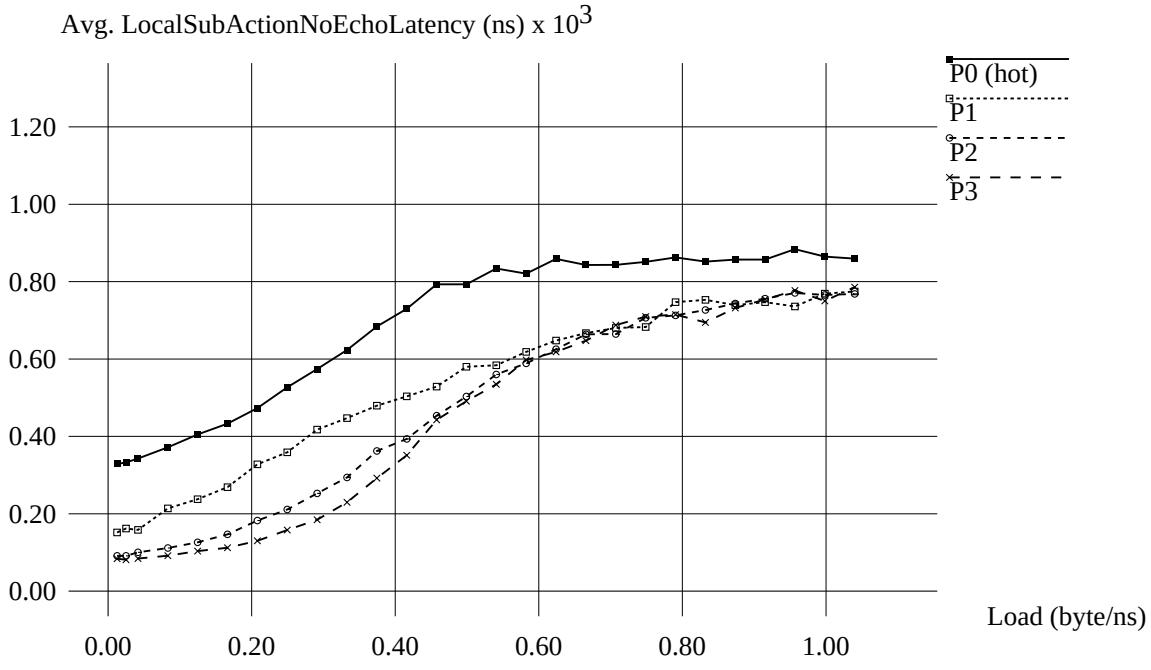
If we consider the results in graph 6.11a-b without considering the latency, the behavior of the SCI flow control mechanism is not convincing. The slight difference in TransThroughput among P1-P3 in the no-flow ring is eliminated when flow control is used, but the prize that has been paid is a reduction in TransThroughput when load is high. In fact, P0 has a generally higher TransThroughput in the no-flow ring than in the flow controlled ring.

Average LocalSubActionNoEchoLatency as a function of load, graph 6.12

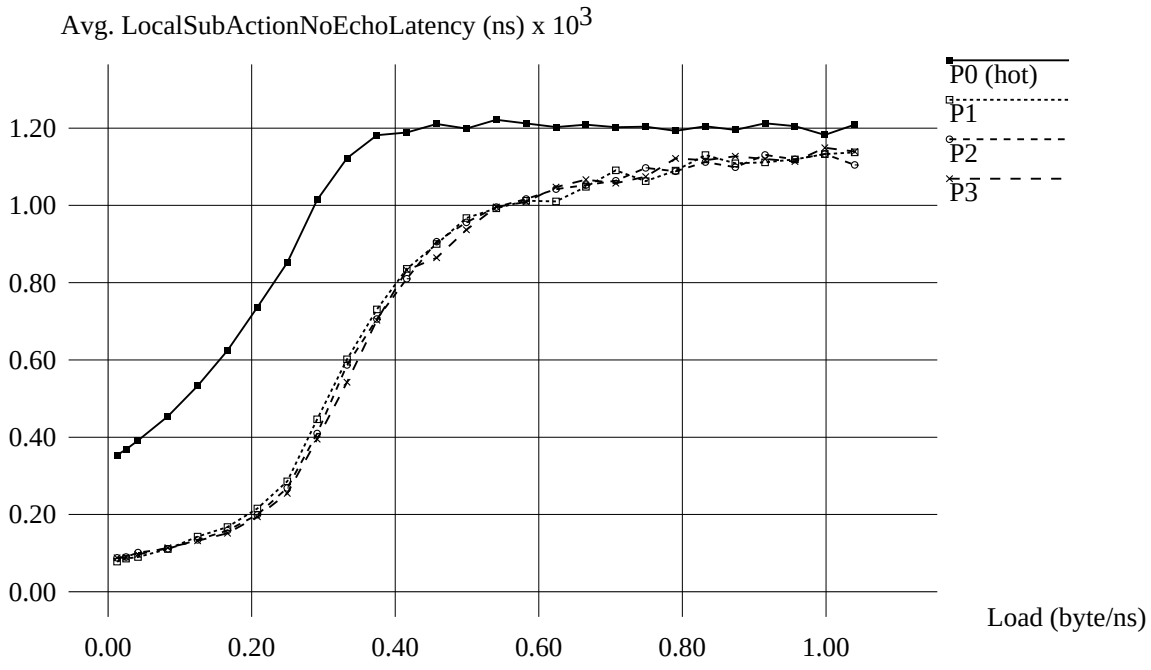
This figure contains two graphs, one related to SCI-rings without flow control (graph 6.12a) and the other related to SCI-rings with flow control (graph 6.12b). Each graph shows for each node P0-P3, the average LocalSubActionNoEchoLatency as a function of load in P1-P3. Note that the load in P0 is fixed. Load is specified in $byte/ns$ along the x-axis and LocalSubActionNoEchoLatency is specified in ns along the y-axis.

We observe from graph 6.12a that the average LocalSubActionNoEchoLatency is highest for packets transmitted from P0, regardless the load in P1-P3. Among the nodes P1-P3, we observe that the average LocalSubActionNoEchoLatency are highest for P1, second-highest for P2 and smallest for P3, when load is less than $0.5byte/ns$. When load exceeds $0.5byte/ns$, the results are somewhat jumbled, but seem to converge. For all nodes the LocalSubActionNoEchoLatency increase when load in P1-P3 increase. When load exceeds $0.5byte/ns$, the results are somewhat jumbled, but the general tendency seem to indicate a maximum value.

This indicate that in 4 node SCI-ring without flow control and which displays a hot-sender load and traffic pattern, the average LocalSubActionNoEchoLatency is highest for the hot node. Of the remaining nodes, the immediate downstream neighbour of the hot node experience the second-highest average LocalSubActionNoEchoLatency, the next downstream neighbour the third-highest average LocalSubActionNoEchoLatency, and the node furthest away experience the lowest average LocalSubActionNoEchoLatency. Therefore the nodes are affected by the hot-sender depending of the distance from the hot-sender. The nodes which are closest (in downstream direction) to the hot-sender are more affected than the nodes furthest away. As we observed in 6.11a, the TransThroughput of P1-P3 are almost identical when load is less than $0.20byte/ns$, and this mean that when load is low, P1-P3 will transmit the same amount of bytes, but for the nodes closest to the hot node this will take more time than for the nodes furthest away.



(a) No flow control



(b) Flow control

Figure 6.12: Latency as a function of load (Hot-sender, 4 nodes, mixed packets).

It is reasonable that the hot node has the highest average LocalSubActionNoEchoLatency because its output-queue is always full and the last packet in the output-queue must wait until all the other packets have left the queue. When the load in the remaining nodes are low, their output-queues rarely fill up, and they only have to wait for the packet from P0 to pass or the bypass-queue to empty. On P1's input-link, one out of three packets are addressed to the P1, while on P2's input-link, one out of two packets are addressed to P2 and there are a lot of empty spaces (idle-symbols) between packets. On the input-link of P3, all incoming send-packets are addressed to it and in addition there are some echo-packets. This mean that P1 has to wait longer than P2, and P2 has to wait longer than P3.

When load increase, P1-P3 have new packets ready in the output-queue before they have been able to empty the bypass-queue, and as a result the LocalSubActionNoEchoLatency will increase. When load in P1-P3 is very high the output-queues are constantly filled with send-packets, in which case the average LocalSubActionNoEchoLatency of each node has reached its maximum level.

When load is high, the hot-sender load and traffic pattern approach the uniform load and traffic pattern, and as a consequence the maximum average LocalSubActionNoEchoLatency of any node in graph 6.12 is approximately equal to the maximum average LocalSubActionNoEchoLatency of a uniform SCI-ring as shown in graph 6.1c.

When we observe graph 6.12b, we will see that the average LocalSubActionNoEchoLatency is different in the flow controlled ring than in the no-flow ring (graph 6.12a). The average LocalSubActionNoEchoLatency is higher for the hot node than for any of the other three nodes (P1-P3), and it will increase rapidly when load increase. When load in P1-P3 exceed $0.4 byte/ns$, the average LocalSubActionNoEchoLatency levels out and will approach $1200ns$. For P1-P3 on the other hand, the average LocalSubActionNoEchoLatency is approximately equal. Even when load is low, where we previously have observed a significant difference in average LocalSubActionNoEchoLatency in the no-flow ring, the average LocalSubActionNoEchoLatency of P1-P3 is almost equal in the flow controlled ring.

The average LocalSubActionNoEchoLatency also seem to converge when load exceed $1.0 byte/ns$, and the average LocalSubActionNoEchoLatency of P0-P3 seem to approach the same maximum value. When load in P1-P3 approach $0.0 byte/ns$ the LocalSubActionNoEchoLatency in any node seem to approach a minimum value. The lower bound is significantly higher in P0 than in P1-P3. The lower bound in P0 is approximately $350ns$ and in P1-P3 approximately $80ns$.

This indicate that in an SCI-ring with flow control displaying a hot-sender load and traffic pattern, the average LocalSubActionNoEchoLatency is higher for the hot node than in the remaining nodes, but the difference will decrease when load increases. The average LocalSubActionNoEchoLatency is almost identical in the remaining group of nodes, regardless the load. The average LocalSubActionNoEchoLatency of all nodes also approach the same maximum value.

The results presented in graph 6.12b and the above conclusion is reasonable assuming an SCI-ring with flow control displaying a hot-sender load and traffic pattern. When load in P1-P3 is low, the load in P0 is still fixed to a high value and its output-queue is completely filled with send-packets. Therefore the LocalSubActionNoEchoLatency are higher in P0 than in P1-P3, which on the other hand, rarely generate packets. The flow control mechanism controls the recovery stage in P1-P3, and node P1 is not more affected than P2-P3 by the hot node. It is also reasonable that the average LocalSubActionNoEchoLatency

in P0-P3 approach a stable maximum value because at high loads the output-queues are filled, and the LocalSubActionNoEchoLatency will not increase further. The maximum average LocalSubActionNoEchoLatency of any node is identical to the maximum average LocalSubActionNoEchoLatency of a *uniform* SCI-ring under similar conditions.

These results concerning average LocalSubActionNoEchoLatency are more encouraging than the results concerning the TransThroughput presented in 6.11a-b. The average LocalSubActionNoEchoLatency of P1-P3 was severely affected by the hot node in the ring without flow control, whereas in the ring with flow control, P1-P3 has approximately the same average LocalSubActionNoEchoLatency. The results in graph 6.11a-b therefore indicate that SCI flow control mechanism can ensure fairness in an SCI-ring with a hot node. The price that has been paid, is a higher average LocalSubActionNoEchoLatency in general.

The relationship between TransThroughput and average LocalSubActionNoEchoLatency, graph 6.13a-b

This figure contains two graphs, one related to SCI-rings without flow control (graph 6.12a) and the other related to SCI-rings with flow control (graph 6.12b). Each graph shows for each node P0-P3, the relationship between TransThroughput and average LocalSubActionNoEchoLatency. TransThroughput is specified in $byte/ns$ along the x-axis and LocalSubActionNoEchoLatency is specified in ns along the y-axis.

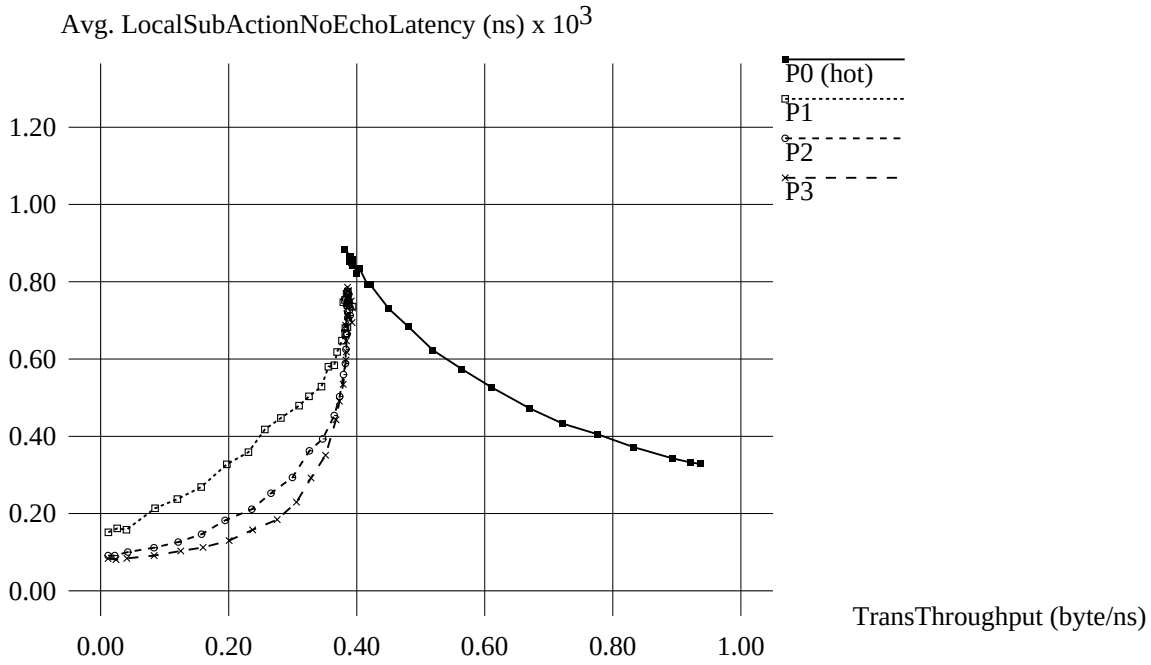
If we observe the graph 6.13a we will see that the results of each node P0-P3 differ significantly. Given the same transmitted throughput, the average LocalSubActionNoEchoLatency of P1 is higher than that of P2, and the average LocalSubActionNoEchoLatency of P2 is higher than that of P3. The results of P0 is significantly different from P1-P3.

This indicate that the performance of the hot-node's first downstream neighbour is worse than the performance for any other node in the ring. It will have a higher average LocalSubActionNoEchoLatency given the same TransThroughput. The second downstream node is less affected than the first, and have a higher performance, the third node is less affected than the second and so forth. When load in P1-P3 is high, the TransThroughput of P1-P3 approach $0.4byte/ns$ and the performance of each node is approximately the same. This is reasonable result considering the results presented in graph 6.11a-b and 6.12a-b.

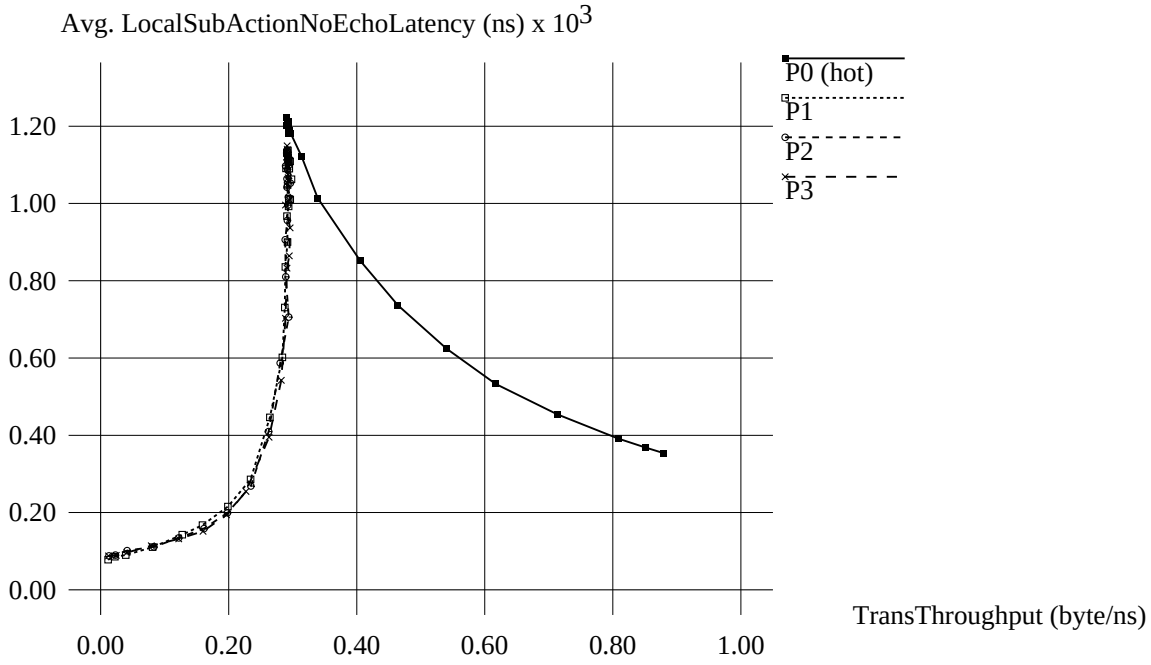
When we observe graph 6.13b, which represent the flow controlled ring, we will see that the performance of each node is quite different in this ring than in the no-flow ring. The relationship between TransThroughput and average LocalSubActionNoEchoLatency for P1-P3 is approximately equal.

This indicates that the performance of each node P1-P3 are approximately equal in an SCI-ring with flow control and a hot-sender load and traffic pattern. When load in P1-P3 is high, the TransThroughput of each node approach $0.3byte/ns$ and the performance of all nodes are approximately equal. The results therefore indicate that adding flow control to an SCI-ring displaying a hot-sender load and traffic pattern, will equalize the difference in performance. This is reasonable result considering the results presented in graph 6.11a-b and 6.12a-b.

Results presented in graph 6.13a-b can be compared to results presented in [Scott et.al., 1992]. Both [Scott et.al., 1992] and the graph 6.13a-b indicate that node P1 is more affected by the hot-node than P2-P3 in an SCI-ring without flow control, and in both cases flow control will eliminate the difference.



(a) No flow control



(b) Flow control

Figure 6.13: The relationship between throughput and latency (Hot-sender, 4 nodes, mixed packets).

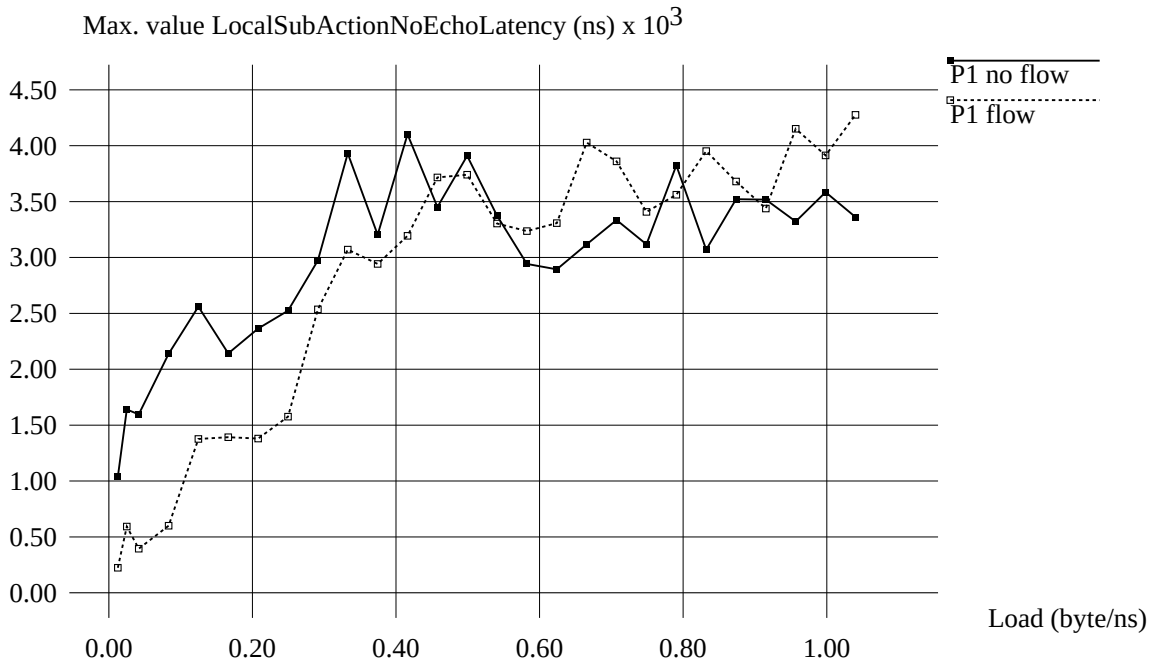


Figure 6.14: Worst case latency as a function of load for the downstream neighbour P1 (Hot-sender, 4 nodes, mixed packets).

The quantitative results in [Scott et.al., 1992] differ from the results in graph 6.13a-b especially at low loads. In [Scott et.al., 1992], the performance of P1-P3 is almost identical when TransThroughput is low, regardless whether flow control is used or not. In graph 6.13a-b we observe a significant difference between node P1 and nodes P2-P3 in the ring without flow control. Nevertheless, it is considered reasonable that there *is* a difference in average LocalSubActionNoEchoLatency when transmitted throughput is low because on P1's input-link, two out three packets have to be bypassed, while on P2's input-link only one out of two packets must be bypassed. On P3's input-link, there are only send-packets addressed to P3 itself and P3 only has to bypass echo-packets, which are smaller in size compared to send-packets.

Maximum value of LocalSubActionNoEchoLatency as a function of load, graph 6.14

This graph shows for node P1 (The hot node's immediate downstream neighbour), the maximum value of LocalSubActionNoEchoLatency sampled during simulation, as a function of load in P1-P3. Load is specified in *byte/ns* along the x-axis and LocalSubActionNoEchoLatency is specified in *ns* along the y-axis.

In graph 6.11a-b, 6.12a-b and 6.13a-b average values of throughput and latency have been emphasize, and average values are interesting in order to show that the flow control mechanism ensure fairness in the average case. In the context of SCI, it must be **guaranteed** that a node receive a minimum amount of bandwidth and that there exist an upper bound for the LocalSubActionNoEchoLatency. To indicate whether the flow control mechanism indeed reduce the worst case latency, figure 6.14 show the maximum value of

LocalSubActionNoEchoLatency of P1 as a function of load, comparing an SCI-ring with flow control to an SCI-ring without flow control.

The interesting situation to consider here arise when load is low, because it is then that P1 is drowned in packets from the hot node. In graph 6.14 we observe that when load is less than $0.40 byte/ns$, the maximum value of LocalSubActionNoEchoLatency of P1 is **less in the ring with flow control** than in the ring without flow control. When load is less than $0.20 byte/ns$, the maximum value of LocalSubActionNoEchoLatency in the ring with flow control is approximately 1000-1500ns less than in the ring without flow control.

This indicate that adding flow control to a ring displaying a hot-sender load and traffic pattern will reduce the worst case latency of P1 when load is low. This also indicate the existence of an upper bound for the LocalSubActionNoEchoLatency when flow control is used.

6.3.2 Summary of results related to hot-sender

The following is summary of the main results related to SCI-rings displaying a hot-sender load and traffic pattern, results which will help us decide on Issue 4, Issue 6 and Issue 7 (chapter 3):

- In an SCI-ring with a hot-sender and without flow control, the «non-hot» nodes are affected depending on the distance (downstream) from the hot node, and the nodes closest (in downstream direction) to the hot node is affected more than the nodes furthest from the hot node. The affected nodes have approximately the same **TransThroughput**, but the nodes closest to the hot node have a higher **LocalSubActionNoEchoLatency**.

- In an SCI-ring with a hot-sender and with flow control, the «non-hot» nodes have approximately the same **TransThroughput** and the same average **LocalSubActionNoEchoLatency**. The worst case **LocalSubActionNoEchoLatency** is also reduced for the immediate downstream neighbour compared to the no-flow case. Adding flow control to the ring with a hot-sender will therefore ensure fairness. The fairness is attained at the expense of the maximum total **TransThroughput** of the ring. When mixed packets are used, the reduction of maximum total **TransThroughput** amounts to 25%.

- When load in the «non-hot» nodes increase, the **TransThroughput** of the hot-sender will decrease.

- When load in the «non-hot» nodes are very high, the hot-sender load and traffic pattern will resemble the uniform load and traffic pattern.

- In relation to hot-sender, the SCIsim-simulator produce results which resemble those presented in [Scott et.al., 1992] - both cases indicate that the «non-hot» nodes are affected depending on their distance (downstream) from the hot-sender in the ring with no flow control, whereas the performance of the «non-hot» nodes are equalized when SCI flow control is used. There are quantitative differences between the hot-sender results in [Scott et.al., 1992] and results presented in this section, in particular in the no-flow case when load is low.

6.4 Node-starvation load and traffic pattern in single SCI-rings

This section will present and discuss results from the simulation of single SCI-rings where the load and traffic pattern is referred to as **node-starvation**. These results will help us decide on Issue 4 and Issue 6 as presented in chapter 3. The following list describes the overall conditions assumed during simulation:

- **Topology:** Single ring.
- **Size:** 4 nodes or 16 nodes.
- **Load and traffic pattern:** node-starvation.
- **Transmitter-stage:** SCI flow control, no flow control.
- **Send-packet size:** Mixed.

The load in each node was increased from one simulation to another - starting with a value close to zero and going up to a level where the throughput and latency measurements had stabilized. The length of one simulation was determined after some preliminary simulations (Refer to section 5.3) and for the 4-node and 16-node ring the simulation time was 1200000ns and 2000000ns respectively. The remaining parameter-values which were assumed during simulation, can be found in table 5.1 in section 5.1.2. The nodes in the smallest ring are labeled **P0 through P3** for convenience, and **P0 will refer to the starved node**, P1 to its immediate downstream neighbour etc.

In order to indicate the throughput of each node in a node-starvation SCI-ring, this section will emphasize the **TransThroughput** as defined in section 5.2. This throughput-measurement includes all send-packets transmitted by a node-interface.

In order to indicate the latency of each node, this section will emphasize **LocalSubActionNoEchoLatency**. Both average and maximum values of LocalSubActionNoEchoLatency will be presented for each node.

Section 6.4.1 will present and discuss results from the simulation of node-starvation SCI-rings with 4 nodes, where the ring either uses the SCI flow control mechanism or not. A 16-node SCI-ring has also been simulated under the assumption of a node-starvation load and traffic pattern, but results here resemble closely the results of the 4 node ring, so the results of 16-node SCI-ring will not be presented.

Section 6.4.2 will give a summary of the main results related to node-starvation.

6.4.1 Results related to node-starvation in SCI-rings with 4 nodes

This section will present and discuss the simulation-results in the following order:

- Throughput as a function of load for each individual node, comparison of no-flow control to flow control. Graph 6.15a-b.
- Latency as a function of load for each individual node, comparison of no-flow control to flow control. Graph 6.16a-b.
- The relationship between throughput and latency for each individual node, comparison of no-flow control to flow control. Graph 6.17a-b.

- Worst case latency as a function of load for starved node P0, comparison of no-flow control to flow control. Graph 6.18.

TransThroughput as a function of load, graph 6.15a-b

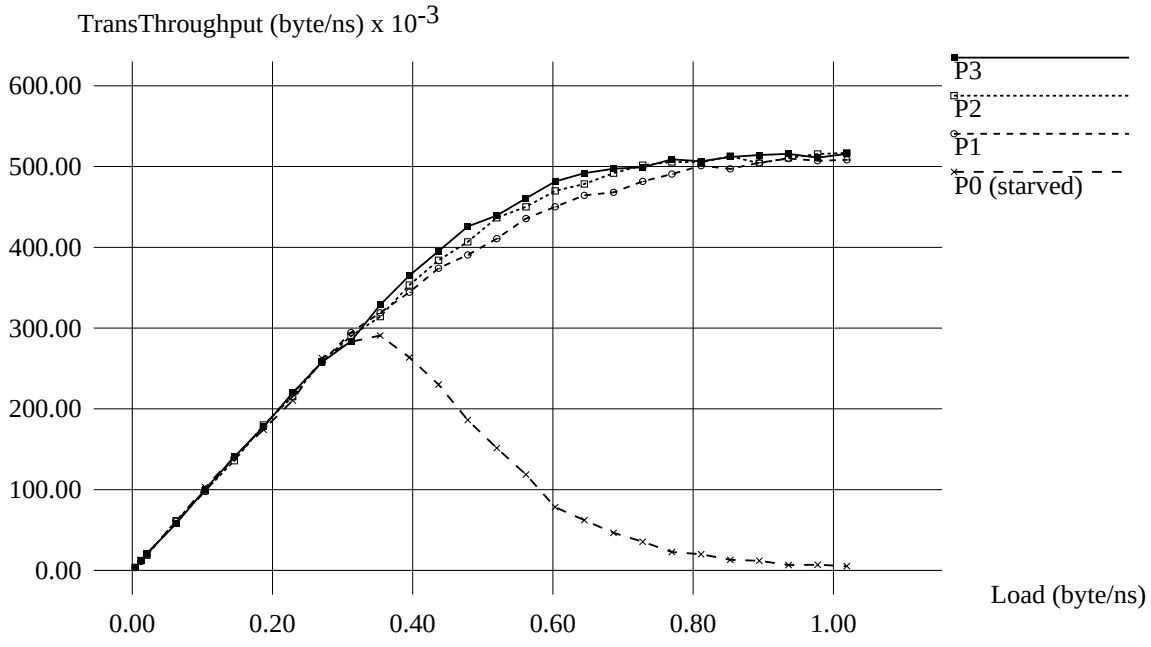
This figure contains two graphs, one related to SCI-rings without flow control (graph 6.15a) and the other related to SCI-rings with flow control (graph 6.15b). Each graph shows for each node P0-P3, the TransThroughput as a function of load in P0-P3. Load and throughput is specified in $byte/ns$ along the x-axis and y-axis respectively.

Observing graph 6.15a we will see that the TransThroughput differ significantly among the nodes, especially between P0, the starved node, and the rest. When load in each node is less than $0.3byte/ns$, the TransThroughput of each node increases linearly when load increases. When load exceeds $0.3byte/ns$, the TransThroughput of P0 levels out and starts to decrease. When load exceeds $1.0byte/ns$, the TransThroughput of P0 approaches $0.0byte/ns$. The TransThroughput of P1-P3 continue to increase even when load exceeds $0.3byte/ns$, but there are minor quantitative differences. When load exceeds $0.8byte/ns$, the TransThroughput of P1-P3 have reached the maximum value of $0.51byte/ns$.

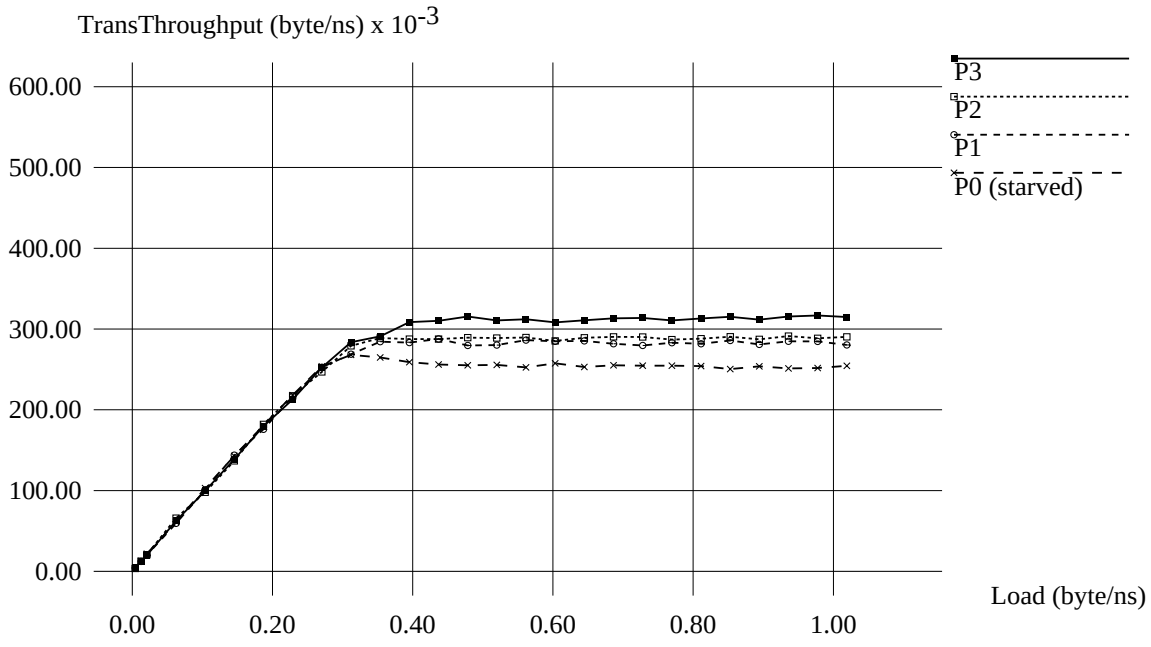
The results presented in graph 6.15a indicate that the TransThroughput of each node in a 4-node SCI-ring without flow control displaying a node-starvation load and traffic pattern, differ significantly. When load is low, less than $0.3byte/ns$ (mixed send-packets), there is no difference in TransThroughput of the four nodes P0-P3, and the TransThroughput equals the load. When load exceed $0.3byte/ns$, the TransThroughput of the starved node starts to decrease and will be reduced to $0.0byte/ns$ when load is sufficiently high. The TransThroughput of the non-starved nodes will continue to increase even when load exceeds $0.3byte/ns$ and up to $0.8byte/ns$ when maximum TransThroughput is reached.

The results in graph 6.15a and the above conclusion is reasonable assuming an SCI-ring without flow control displaying a node-starvation load and traffic pattern. In an SCI-ring without flow control a node can transmit only when the bypass-queue is empty. In order to transmit several packets the node must empty the bypass-queue between transmissions and in the absence of flow control, this can be attained only when the node receives packets addressed to it or when the ring is lightly loaded. When load is low, few packets circulate the ring. Therefore there is enough free space between packets to enable the starved node to empty the bypass-queue between transmissions. It is only the free space between the packets which enables the starved node to transmit send-packets, and when load is high, packets will fill the ring at the expense of free space. As a result the starved node will struggle to empty the bypass-queue before new send-packets enter the output-queue. If the load is sufficiently high, the starved node will spend a very long time in the recovery stage, and only occasionally manage to empty the bypass-queue. Consequently, the starved node will transmit less than the other nodes and its output-queue will fill up.

The non-starved nodes will not be affected in the same way as the starved node because they all receive packets from each other, in fact the non-starved nodes receive packets from the starved node also. As a result the non-starved nodes increase their TransThroughput until the ring is fully saturated. In the load interval form $0.3byte/ns$ to $0.8byte/ns$ there is a slight difference in TransThroughput among the non-starved nodes and this is reasonable. In the same load-interval, the starved node is still able to transmit some packets, and it is P0's packets, transmitted uniformly to the non-starved nodes, that affect the non-starved nodes and their TransThroughput. P1, the downstream neighbour of the starved node, is more affected than P2 and P3 because it has to bypass packets to P2 and P3 (from P0).



(a) No flow control



(b) Flow control

Figure 6.15: Throughput as a function of load (Node-starvation, 4 nodes, mixed packets).

Node P2 only has to bypass packets to P3 (from P0), whereas P3 does not have to bypass any packets from the starved node. As a result, P1 transmit slightly less than P2, and P2 transmit slightly less than P3. When load exceed $0.8byte/ns$ the starved node transmit very few packets, and it will behave almost as a bypass-queue. The non-starved nodes transmit uniformly among themselves and as a result they have approximately the same TransThroughput.

Observing graph 6.15b, we will see that the results of the flow controlled ring differ significantly from the ring without flow control (graph 6.15a). When load is less than approximately $0.25byte/ns$, the TransThroughput of each node increases linearly as load increase, and TransThroughput equals the load. When load exceed $0.25byte/ns$ and increase further, the TransThroughput will approach a maximum value. For P0 (the starved node) the maximum TransThroughput is approximately $0.25byte/ns$, and for P1, P2 and P3 (the non-starved nodes) the maximum transmitted throughput is $0.28byte/ns$, $0.29byte/ns$ and $0.32byte/ns$ respectively.

The results presented in graph 6.15b indicate that in a 4-node SCI-ring with flow control displaying a node-starvation load and traffic pattern, all nodes, including the starved node, have a minimum TransThroughput. When load is low, less than approximately $0.25byte/ns$, all nodes have approximately the same TransThroughput, and the TransThroughput will equal the load. When load exceed $0.25byte/ns$ the TransThroughput of all nodes will approach a maximum value. The maximum TransThroughput of the starved node is less than the maximum TransThroughput of any of the non-starved nodes.

The results in graph 6.15b and the above conclusion is reasonable assuming an SCI-ring with flow control displaying a node-starvation load and traffic pattern. The flow control mechanism stop other nodes from transmitting packets when a node is in recovery and in this way, the length of the recovery stage is bounded. When the starved node P0 has transmitted a packet and goes into the recovery stage, it will actively reduce the bypass-queue's size by emitting NoGo-idles and thereby drain Go-idles from the ring. When no Go-idles are left in the ring, the other nodes will not be able to transmit packets and P0 will therefore empty its bypass-queue.

Because P0 never receive packets addressed to it, its only way to reduce the bypass-queue during recovery is to emit NoGo-idles. It is therefore reasonable that the starved node transmit less than the non-starved nodes (which receive packets).

The difference in maximum TransThroughput of P1-P3 is also reasonable because node P0 transmit uniformly to all other nodes. P1 will have to bypass packets to P2 and P3 (from P0), P2 will have to bypass packets to P3 (from P0) whereas P3 will not have to bypass any packets from P0. Since P1 has to bypass more packets than P2, and P2 has to bypass more packets than P3, P1 will transmit less than P2 and P2 transmit less than P3.

Considering the results presented in graph 6.15a related to SCI-rings without flow control, where the starved node was unable to transmit at high loads, flow control seem to have a promising effect on the starved node. The simulation results in graph 6.15b indicate that even the starved node receive a minimum amount of bandwidth. The prize that has been paid is a 25% reduction in maximum TransThroughput of the ring.

Average LocalSubActionNoEchoLatency as a function of load, graph 6.16a-b

This figure contains two graphs, one related to SCI-rings without flow control (graph 6.16a) and the other related to SCI-rings with flow control (graph 6.16b). Each graph shows for each node P0-P3, the average LocalSubActionNoEchoLatency as a function of load in P0-P3. Load is specified in $byte/ns$ along the x-axis and LocalSubActionNoEchoLatency is specified in ns along the y-axis.

Observing graph 6.16a we see that the average LocalSubActionNoEchoLatency differ significantly among the nodes. When load is less than $0.20byte/ns$ the starved node P0, has the same average LocalSubActionNoEchoLatency as P1-P3, the non-starved nodes. When load exceeds $0.20byte/ns$, the average LocalSubActionNoEchoLatency of P0 increases very fast, and diverges from that of P1-P3. Moreover the results do not indicate whether there is an upper bound for the average LocalSubActionNoEchoLatency of P0.

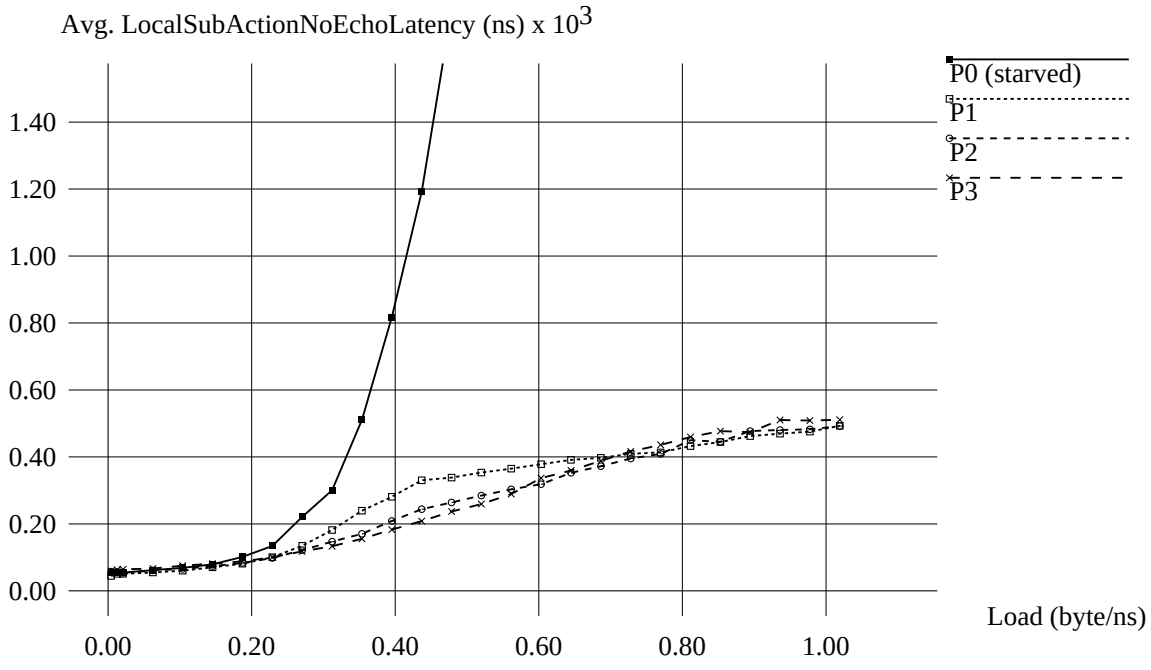
The results of the non-starved nodes indicate an upper and a lower bound for average LocalSubActionNoEchoLatency of P1-P3, and when the load exceeds $0.80byte/ns$, the results of P1-P3 converge and seem to level out. When load is in the interval from $0.3byte/ns$ to $0.80byte/ns$, the average LocalSubActionNoEchoLatency differ among the non-starved nodes and P1 has a higher LocalSubActionNoEchoLatency than P2, and P2 has a higher LocalSubActionNoEchoLatency than P3.

The results presented in graph 6.16a indicate that the nodes in a 4-node SCI-ring without flow control displaying a node-starvation load and traffic pattern, have a significantly different LocalSubActionNoEchoLatency. The starved node will experience the same average LocalSubActionNoEchoLatency as the non-starved nodes when load is low, but when load increase the average LocalSubActionNoEchoLatency increase rapidly and apparently without an upper bound. The average LocalSubActionNoEchoLatency of the non-starved nodes are bounded upward and downward. In the load interval from $0.20byte/ns$ to $0.70byte/ns$, P1 will have a slightly higher average LocalSubActionNoEchoLatency than P2, and P2 will have a slightly higher LocalSubActionNoEchoLatency than P3, but when load is either very high or very low, P1-P3 have almost the same average LocalSubActionNoEchoLatency.

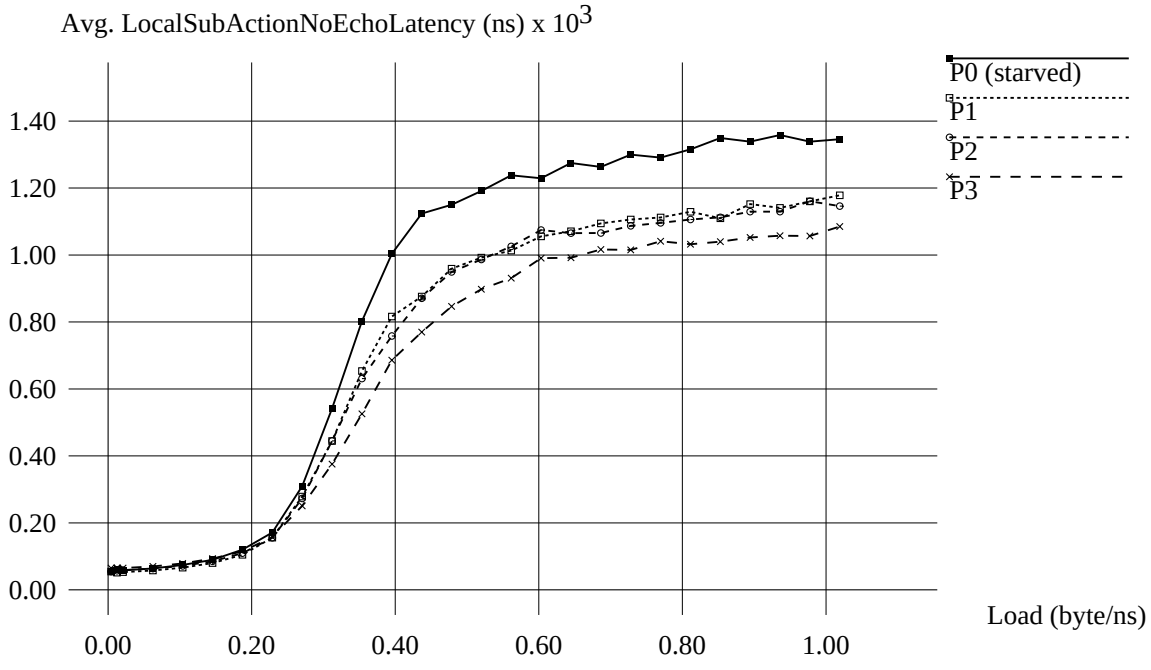
The results in graph 6.16a and the above conclusion is reasonable assuming an SCI-ring without flow control displaying a node-starvation load and traffic pattern. When load is low, few packets circulate the ring, and there are a lot of free space between then. Therefore, the bypass-queue rarely fills up during transmission and packets inserted into the output-queue can be transmitted almost immediately without having to wait in the output-queue. As a result, the average LocalSubActionNoEchoLatency is identical in all nodes P0-P3 when load is low.

When load is in the interval from $0.2byte/ns$ to $0.3byte/ns$, we know from graph 6.15a that the starved node increase its TransThroughput, but the average LocalSubActionNoEchoLatency increase more than P1-P3. This is reasonable because the free space between packets are few and far between, so P0's recovery stage will take more time, and P0's packets will have to wait in the output-queue. When load exceeds $0.3byte/ns$, packets will be generated faster in P0 than can be transmitted, and in addition the TransThroughput of P0 is actually reduced (refer to figure 6.15a). The output-queue of P0 fills up rapidly and packets have to wait a long time before they can be transmitted. As a result, the average LocalSubActionNoEchoLatency of P0 increases quickly when load is high.

The non-starved nodes transmit packets uniformly among themselves, and their bypass-queues will be emptied when they receive packets. In the average case the length of the



(a) No flow control



(b) Flow control

Figure 6.16: Average latency as a function of load (Node-starvation, 4 nodes, mixed packets).

recovery stage, and consequently the time between transmissions, are almost identical in the three nodes. The difference in average LocalSubActionNoEchoLatency in P1-P3, when load is in the interval from $0.3 byte/ns$ to $0.7 byte/ns$, is caused by the fact that the starved node is still able to transmit some packets. P1 will have to bypass packets to P2 and P3 (from P0) and P2 will have to bypass packets to P3 (from P0), whereas P3 does not have to bypass any packets from P0. As a results, the recovery stage is longer in P1 than in P2, and longer in P2 than in P3. When load exceeds $0.8 byte/ns$ we know that P0 no longer is transmitting, and it will therefore not affect the non-starved nodes. Consequently, P1-P3 experience the same average LocalSubActionNoEchoLatency when load is higher than $0.8 byte/ns$.

Observing graph 6.16b, we see that the results of the flow-controlled ring is different from the ring without flow control. For all nodes P0-P3, there is an upper and a lower bound for the average LocalSubActionNoEchoLatency. When load is low, all nodes experience the same average LocalSubActionNoEchoLatency, and when load is high, the nodes approach different maximum values. The results therefore show the asymptotic behavior we observed in graph 6.1c (section 6.2 related to uniform traffic pattern).

The results in graph 6.16b therefore indicate that the average LocalSubActionNoEchoLatency of each node in an SCI-ring with flow control and node-starvation, is bounded upward. It is also important to emphasize that the starved node has a **bounded** average LocalSubActionNoEchoLatency, as opposed to the seemingly unbounded average LocalSubActionNoEchoLatency in the ring without flow control.

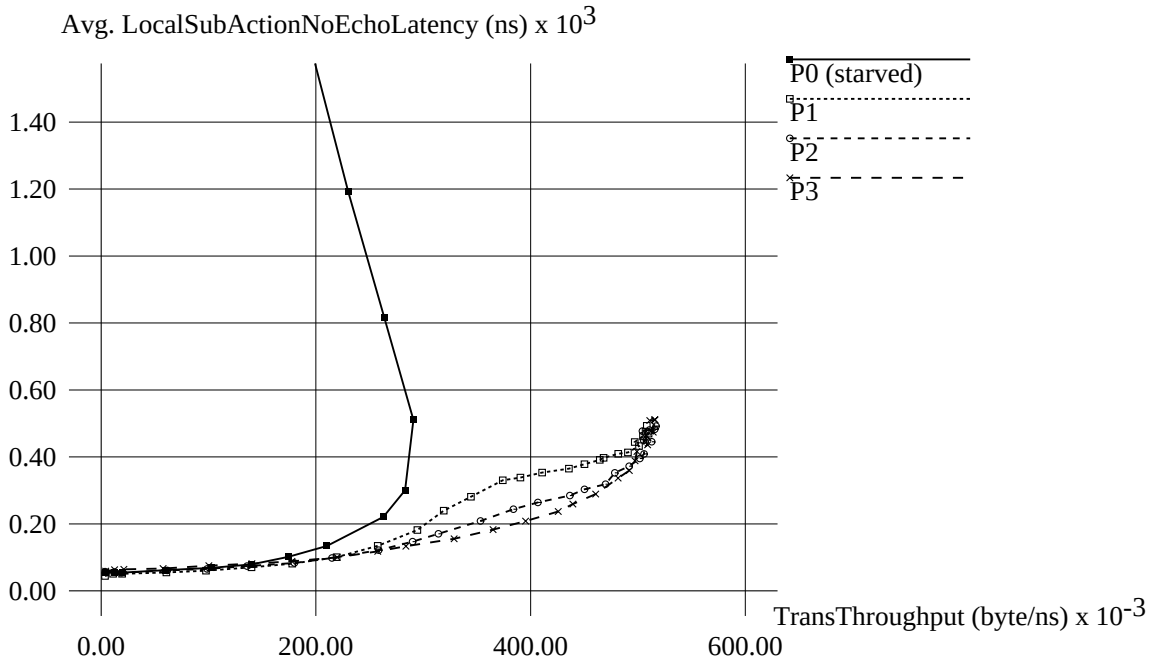
The results in graph 6.16b and the above conclusion is reasonable. When load is low, there are few packets circulating the ring and there are a lot of free space between packets. Consequently the bypass-queue seldom fills up and a packet inserted into the output-queue can be transmitted almost immediately. When load increases, there will be more packets circulating the ring, and there are less free space between packets. The bypass-queues begin to fill up and the output-queues as well, because there are more new packets generated than can be transmitted.

It is reasonable that the starved node experience a higher LocalSubActionNoEchoLatency than the non-starved nodes, despite that flow control is used, because its only way to empty the bypass-queue is to emit NoGo-idles between packets. The non-starved nodes receive packets and this will enable them to reduce the bypass-queue faster than the starved node.

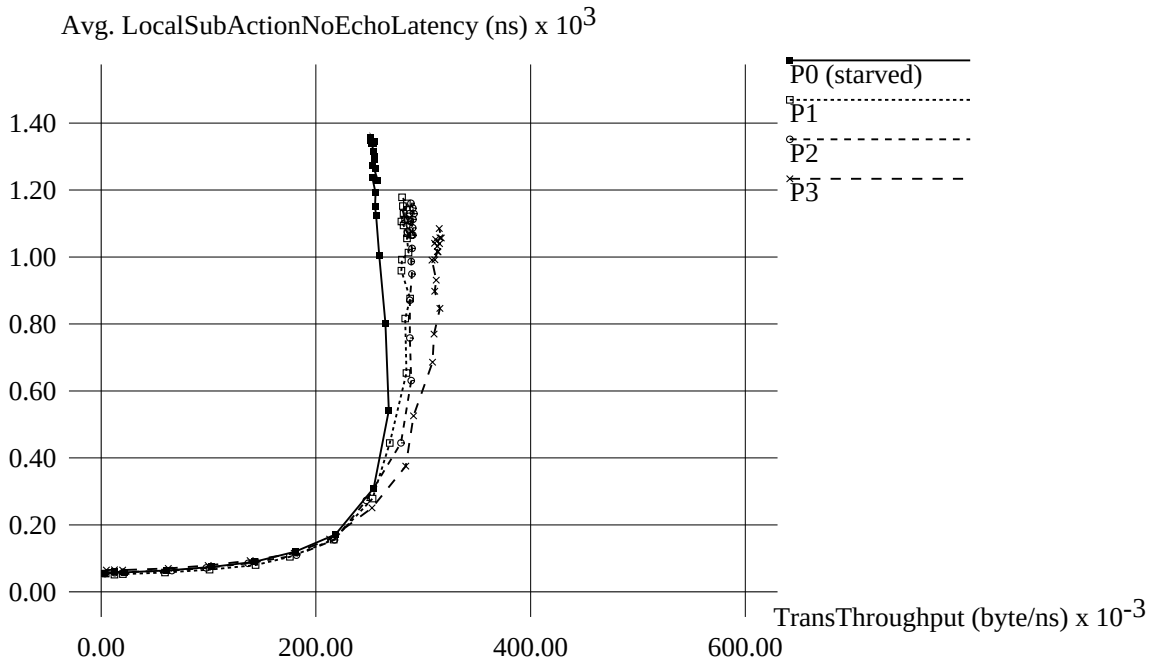
The difference in maximum average LocalSubActionNoEchoLatency among the nodes is caused by the difference in maximum TransThroughput. P0 has the smallest maximum TransThroughput of all the nodes, and it will take more time to transmit the same amount compared to P3. The latter node has the highest maximum TransThroughput of them all, and as a result, the smallest maximum average LocalSubActionNoEchoLatency.

The relationship between TransThroughput and average LocalSubActionNoEchoLatency, graph 6.17a-b

This figure contains two graphs, one related to SCI-rings without flow control (graph 6.12a) and the other related to SCI-rings with flow control (graph 6.12b). Each graph shows for each node P0-P3, the relationship between TransThroughput and average LocalSubActionNoEchoLatency. TransThroughput is specified in $byte/ns$ along the x-axis and LocalSubActionNoEchoLatency is specified in ns along the y-axis.



(a) No flow control



(b) Flow control

Figure 6.17: The relationship between throughput and latency (Node-starvation, 4 nodes, mixed packets).

Observing graph 6.17a we see that the performance of the four nodes is approximately equal when TransThroughput is low, less than $0.175 byte/ns$. When load increase, the performance of P0 diminishes, spending much time transmitting the same throughput as the other nodes. When load increase further, we observe that the performance of P0 grow even worse, the throughput decreases and the average latency increases. The performance of the non-starved node vary slightly, the performance of P3 is better than P2's, and the performance of P2 is better than P1's, but when load is very high their performance are almost identical.

This result is reasonable seen in the light of the results presented in 6.15a and 6.16a. When the load is low, the starved node experience the same TransThroughput and average LocalSubActionNoEchoLatency as the non-starved nodes, but when load increases, P0's TransThroughput will decrease and the LocalSubActionNoEchoLatency will increase, yielding a poor performance. The non-starved nodes have approximately the same TransThroughput and the same average LocalSubActionNoEchoLatency at higher loads, while at intermediate loads, P3 have a higher TransThroughput and a lower average LocalSubActionNoEchoLatency than P1 and P2, and therefore a better performance than P1 and P2. P2 have a higher TransThroughput and a lower average LocalSubActionNoEchoLatency than P1 at intermediate loads, and therefore has a better performance than P1.

Observing graph 6.17b, we will see another result compared to 6.17a. The performance of the nodes are almost identical when TransThroughput is less than $0.22 byte/ns$, and the average LocalSubActionNoEchoLatency is almost identical given the same TransThroughput. When TransThroughput exceeds $0.22 byte/ns$, the performance goes down in all nodes, and it takes much more time (higher average LocalSubActionNoEchoLatency) to transmit a little more throughput. The performance of P0 is slightly worse than the performance of the non-starved nodes.

The result in graph 6.17b is reasonable considering the results in 6.15b and 6.16b. It is also reasonable that the performance of P3 is higher than the other nodes because it has a higher maximum TransThroughput and smaller maximum average LocalSubActionNoEchoLatency than the other nodes.

Maximum value of LocalSubActionNoEchoLatency as a function of load, graph 6.18

This graph shows for node the starved node P0, the maximum value of LocalSubActionNoEchoLatency sampled during simulation, as a function of load in P0-P3. Load is specified in $byte/ns$ along the x-axis and LocalSubActionNoEchoLatency is specified in ns along the y-axis.

So far, adding flow control to a ring seems to have a promising effect on the starved node, and simulation result indicate that a minimum bandwidth can be guaranteed as well as an upper bound for the average LocalSubActionNoEchoLatency. It is also interesting to observe the maximum value of LocalSubActionNoEchoLatency of P0 sampled during simulation, and compare flow control to no flow control rings.

Observing graph 6.18, we will see that the maximum sample is identical in the flow case and the no-flow case when the load is less than $0.3 byte/ns$. When load exceeds $0.3 byte/ns$ the maximum sample in the no flow case increase rapidly, whereas the flow case seem to stabilize around $5000 ns$. The maximum value of LocalSubActionNoEchoLatency sampled during simulation of the no-flow ring is $246522 ns$, indicating a *very* long recovery stage.

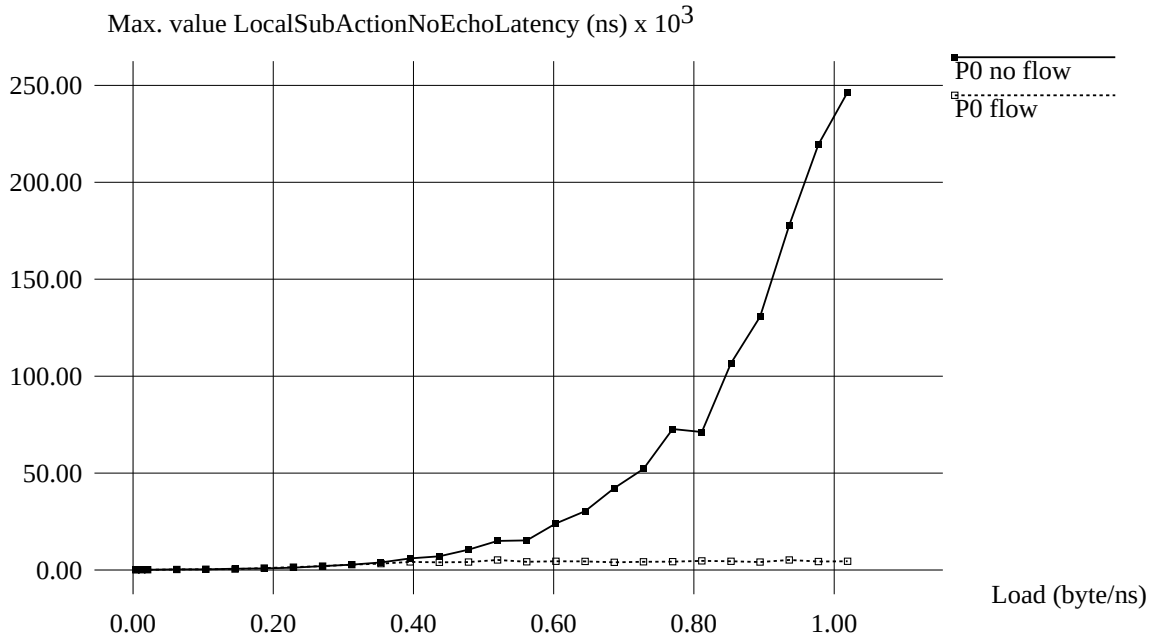


Figure 6.18: Worst case latency as a function of load for the starved node P0 (Node-starvation, 4 nodes, mixed packets).

The results in graph 6.18 therefore indicate that the worst case LocalSubActionNoEcho-Latency is bounded upward for the starved node when SCI flow control is used, whereas in the ring without flow control it will, most likely, be unbounded. Of course the results in graph 6.18 represent only *singular* values (the maximum sample), and could have been caused by mere chance. Nevertheless, the stable maximum value of LocalSubAction-NoEchoLatency in the flow controlled ring *and* the increase in the no-flow case during 15 simulations, give us reason to believe that an upper bound indeed exist when flow control is used.

6.4.2 Summary of results related to node-starvation

The following is a summary of the main results related to SCI-rings displaying a node-starvation load and traffic pattern, results which will help us decide on Issue 4 and Issue 6 (chapter 3):

- In an SCI-ring with a starved node and without flow control, the starved node is not affected when the load is low. When load increase, the starved node will be affected and will have a lower **TransThroughput** and a higher average **LocalSub-ActionNoEchoLatency** than the non-starved nodes. When load is very high, the starved node will be unable to transmit anything and the **LocalSubActionNoEcho-Latency**, both in the average case and the worst case, will be unbounded.

- In an SCI-ring with a starved node and flow control, the starved node receive a minimum amount of bandwidth, and has approximately the same **TransThroughput** as the non-starved node. Both the average and the worst case values of **Local-SubActionNoEchoLatency** for the starved node is reduced compared to the ring

without flow control. The maximum TransThroughput of the ring is reduced with approximately 15% compared to the no-flow case.

6.5 Summary

This chapter has presented and discussed results from the simulation of single SCI-rings, enabling us to decide on Issue 4 - Issue 7 related to the performance of SCI, as described in chapter 3.

The results have been presented in three main sections (6.2, 6.3 and 6.4) related to uniform, hot-sender and node-starvation load and traffic pattern respectively (defined in chapter 5), and a summary of these results are given at the end of each section (6.2.4, 6.3.2 and 6.4.2).

The summary of results related to uniform load and traffic pattern will help us decide on issue 4, Issue 5 and Issue 7, the summary of results related to hot-sender load and traffic pattern will help us decide on Issue 4, Issue 6 and Issue 7 and finally, the summary of results related to node-starvation load and traffic pattern will help us decide on Issue 4 and Issue 6. The reader should therefore refer to those latter sections. Final conclusions on Issue 4 - Issue 7 are given in chapter 9.

[This page has been intentionally left blank]

Chapter 7

Results from the simulation of an SCI multi-ring interconnect

This chapter presents and discusses results from the simulation of the 4-ring interconnect, consisting of 4 SCI-rings communicating through 4 switches. These results will help decide on Issue 8 in chapter 3.

The SCIsim simulator described in chapter 4 was used and its input was the parameters described in chapter 5.

Section 7.1 will briefly describe the parameter-values assumed during simulation and which measurements have been emphasized in section 7.2. The latter section will present the actual results and compare those to results from simulation of single SCI-rings. Section 7.3 will summarize the main results related to the multi-ring interconnect.

7.1 Parameters and measurements in multi-ring simulations

The following list describes the overall conditions assumed during simulation:

- **Topology:** 4 rings w/4 switches, refer to figure 5.1b.
- **Size:** 16 nodes.
- **Load and traffic pattern:** Uniform.
- **Transmitter-stage:** SCI flow control.
- **Send-packet size:** Mixed.

The load was increased from one simulation to another - starting with a value close to zero and going up to a level where the throughput and latency-measurements had stabilized. The length of one simulation was determined after some preliminary simulations (Refer to section 5.3) and it was found that simulation time equivalent to $1500000ns$ produced reliable results. The remaining parameter-values assumed during simulation, can be found in table 5.2 in section 5.1.2. The term **4-ring** will refer to the simulated interconnect.

In order to indicate the throughput of the 4-ring interconnect, this section will emphasize the **RawThroughput** and **NetThroughput** as defined in section 5.2. The two throughput-measurements include all send-packets received by an application process (The switches are not considered) and RawThroughput counts all bytes in a send-packet *minus*

the CRC-symbol, whereas NetThroughput counts the data-bytes only. Total and average figures will be presented - the former includes all nodes, while the latter equals the total throughput divided by the number of nodes.

In order to indicate the latency, this section will emphasize **RemoteSubAction-Latency** as defined in section 5.2. According to the definition, RemoteSubActionLatency equals the time from the packet is generated in the source-node until it is received by the target-node. In a multi-ring interconnect, the RemoteSubActionLatency may include the time a packet spends passing through several rings and switches.

7.2 4-ring interconnect

This section will present and discuss the results in the following order:

- Throughput as a function of load for the whole interconnect, comparison of 4-ring interconnect to single-ring. Graph 7.1.
- Latency as a function of load for the whole interconnect, comparison of 4-ring interconnect to single-ring. Graph 7.2a-b.
- The relationship between throughput and latency for the whole interconnect, comparison of 4-ring interconnect to single-ring. Graph 7.3a-b.

Total RawThroughput and total NetThroughput as a function of total load, graph 7.1

This graph shows total RawThroughput and total NetThroughput as a function of total load, in the 4-ring interconnect and a single-ring with 16 nodes. Total load and total throughput is specified in $byte/ns$ along the x-axis and the y-axis respectively.

If we first consider the 4-ring interconnect and its total RawThroughput and total NetThroughput we will observe the following:

- When load is in the interval from $0.0byte/ns$ to $1.6byte/ns$, the total RawThroughput and total NetThroughput increase approximately linearly when total load increase. Total RawThroughput is approximately equal to total load, whereas NetThroughput is less than total load.
- When total load is in the interval from $1.6byte/ns$ to $2.0byte/ns$, the RawThroughput and NetThroughput decrease when total load increase.
- When total load exceeds $2.0byte/ns$, the total RawThroughput and total NetThroughput approach a stable value when total load increase. The stable value of RawThroughput and NetThroughput is $0.95byte/ns$ and $0.60byte/ns$ respectively.
- In general, the NetThroughput is less than RawThroughput. For any given load, the NetThroughput is approximately 35% less than RawThroughput.

The results in graph 7.1 indicate some properties of the throughput in an interconnect configured as the 4-ring interconnect. When load is less than a certain limit L, the total RawThroughput and total NetThroughput will increase as total load increase, and approximately linearly. In particular the total RawThroughput equals the total load. When load

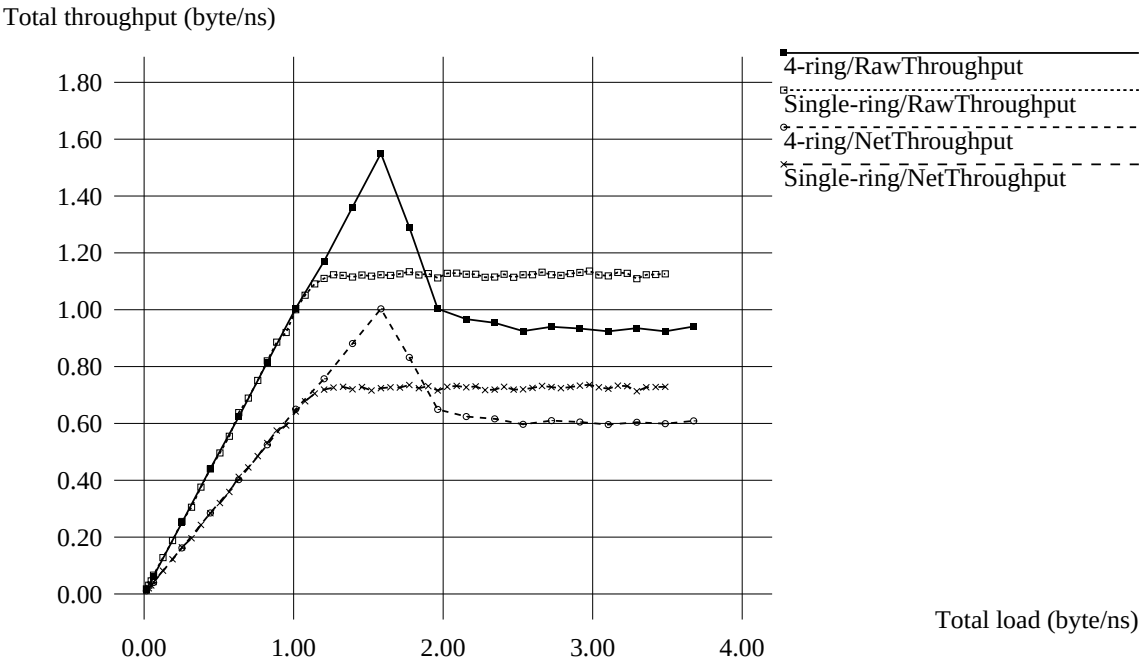


Figure 7.1: Throughput as a function of load (4-ring versus single-ring).

exceed L, the total RawThroughput and total NetThroughput will decrease as total load increase, and will level out and approach a stable value when total load gets high. This stable value is less than the maximum value experienced at lower loads. In other words, when load increase, the total RawThroughput and total NetThroughput will first increase, then decrease and finally stabilize.

The results in graph 7.1 and the above conclusion is reasonable assuming a 4-ring interconnect. The most significant event takes place when the switches begin to saturate. A 2×2**-switch**, the type of switch simulated in the SCIsim simulator, is saturated in one direction when the input-queue of the in-going node-interface *and* the output-queue of the outgoing node-interface is full. A switch is fully saturated when it is saturated in both directions.

A switch becomes saturated when total load exceeds a certain limit and packets are entering the in-going node-interface faster than packets are transmitted from the outgoing node-interface. The following will explain why it is reasonable that total throughput is decreasing in graph 7.1:

- When total load is less than a certain limit L, the switches will not saturate. All packets that are generated will be received either by a node-interface or by a switch. In the latter case the packets will be moved to another ring. Because packets are received at the same rate as they are generated, total RawThroughput will equal the total load.

- When total load exceed L, more packets will circulate each ring. Because traffic in each ring have increased, the switches are unable to transmit as much as they used to. In addition the number of packets passing through each switch have increased and

consequently the switches become saturated. When a switch is saturated, new packets which trying to gain access to the switch (in order to move to another ring) will be rejected and must be retransmitted. These rejected packets will come in addition to new packets and increase the traffic on each ring further. Rejected packets does not lead to reduced throughput in itself, but rejected packets will add to traffic in an already overloaded ring and make things worse.

Because traffic has increased in the ring, the switches are not able to transmit as much as they used to. When load was lower, the switches actually behaved as **hot senders** because a large part of the packets on each ring had to pass through the switches in order to reach their target-node. From section 6.3 we know that hot senders are able to transmit most when load is low in the remaining nodes and when load increase in the remaining nodes the hot sender will transmit less. The load and traffic pattern in the 4-ring interconnect is uniform and this imply that 12 out of 15 send-packets generated by an application process are destined for a node in another ring.

In other words, the switches are bottlenecks, but this alone does not cause the reduction in total throughput we observe in graph 7.1 when total load exceed $1.6 byte/ns$. What *is* causing the reduction in throughput is that the alleged bottlenecks become narrower when load increase.

The switches are, however, still able to transmit something, their minimum amount of bandwidth is guaranteed by the flow-control mechanism (according to section 6.3 and 6.4). Some of the packets transmitted by the switches are destined for nodes in the same ring and packets will be accepted by the target-node. When load is very high and the rings and switches are fully saturated, a stable level of throughput is reached. The routing of packets in the 4-ring interconnect will not lead to dead-lock situations (Refer to section 5.1.2 for further details), so packets in the switches will eventually leave and make progress.

It is also reasonable that total NetThroughput is less than total RawThroughput because the latter include both data and header of the send-packets while the former include only the data. In graph 7.1, the total NetThroughput is 35% less than total RawThroughput because 35% is spent on header. The following calculation of packet-size (P_{mean}) and header-size (P_{header}) shows this (note that the CRC-symbol has been omitted):

$$P_{mean} = 0.6 * 14 + 0.4 * 78 = 39.6 bytes$$
$$P_{header} = 14 bytes$$
$$P_{header}/P_{mean} = 14/39.6 \approx 35\%$$

If we then compare the 4-ring interconnect to the single-ring, we will see that total RawThroughput and NetThroughput is approximately equal when total load is less than $1.1 byte/ns$. When total load exceed $1.1 byte/ns$, we observe that the total RawThroughput and total NetThroughput of the single-ring will level out and approach a stable maximum value. In the 4-ring interconnect, the total RawThroughput and total NetThroughput will continue to increase as long as total load is less than $1.6 byte/ns$, upon which the total RawThroughput and total NetThroughput decrease, and approach a stable value. The stable value of total throughput in the 4-ring interconnect is approximately 17% less than the maximum total throughput of the single-ring.

The results in graph 7.1 therefore indicate that the total RawThroughput and NetThroughput is equal in the 4-ring interconnect and the 16node single-ring when load is

low. When load gets higher, the total throughput of the 4-ring interconnect is higher than in the single-ring, but only temporarily, because as load increase further, the throughput of the 4-ring interconnect will decrease and become less than in the single-ring.

Average RemoteSubActionLatency as a function of total load, graph 7.2

This graph shows average RemoteSubActionLatency as a function of total load, in a 4-ring interconnect and a single-ring with 16 nodes. Total load is specified in $byte/ns$ along the x-axis and average RemoteSubActionLatency is specified in ns along the y-axis. Graph 7.2b zooms in on graph 7.2a so that details are revealed.

If we first consider the 4-ring interconnect and its average RemoteSubActionLatency, we will observe the following:

- Average RemoteSubActionLatency increase when total load increase, and the rate of growth vary significantly.
- There is an upper and a lower bound of the average RemoteSubActionLatency. The average RemoteSubActionLatency approaches the lower bound of $170ns$ when total load approaches $0.0byte/ns$, and will approach the upper bound of $10500ns$ when total load exceeds $3.0byte/ns$.

The results in graph 7.2a-b indicate that in the 4-ring interconnect, the average RemoteSubActionLatency will increase when total load increase, and the average RemoteSubActionLatency will be bounded both upward and downward.

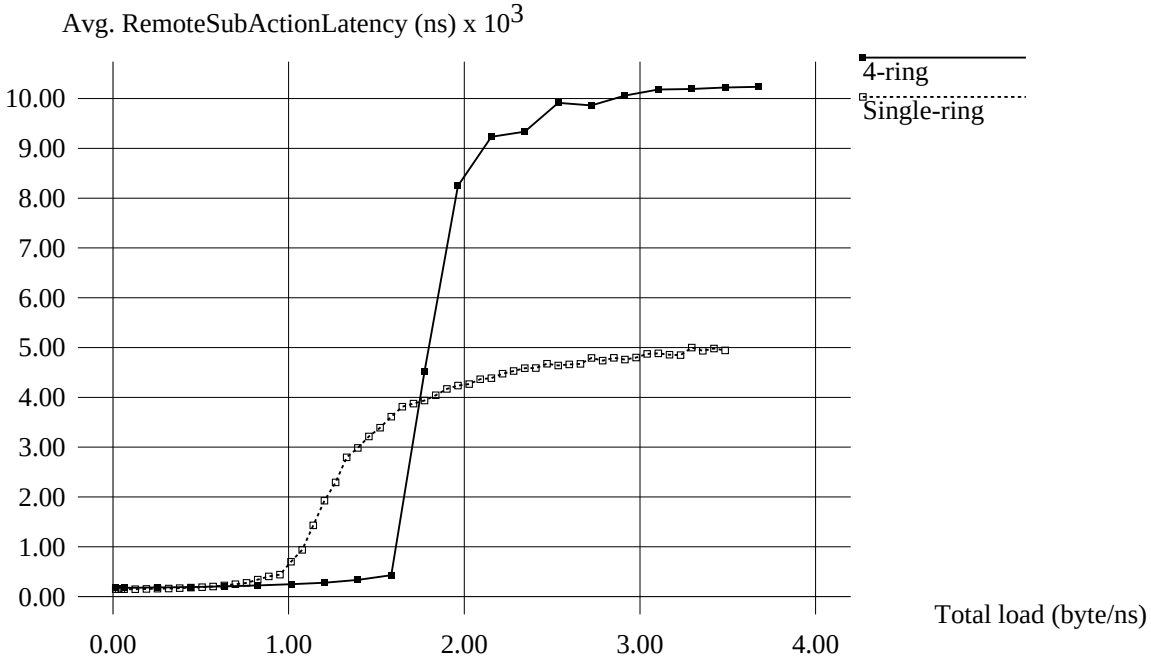
The results in graph 7.2a-b are reasonable. When load is less than $1.6byte/ns$, packets are transmitted at the same rate as new packets are generated and the output-queues does not fill up. From graph 7.1, we know that the total RawThroughput and total NetThroughput will increase linearly when load is less than $1.6byte/ns$.

When load exceeds $1.6byte/ns$, the switches begin to fill up because traffic in each ring is so high that switches are unable to transmit packets as fast as they are receiving new packets. As a result the output-queue will fill up and the latency will increase rapidly. From graph 7.1 we know that total throughput in fact will decrease when total load exceeds $1.6byte/ns$, so this will only make things worse.

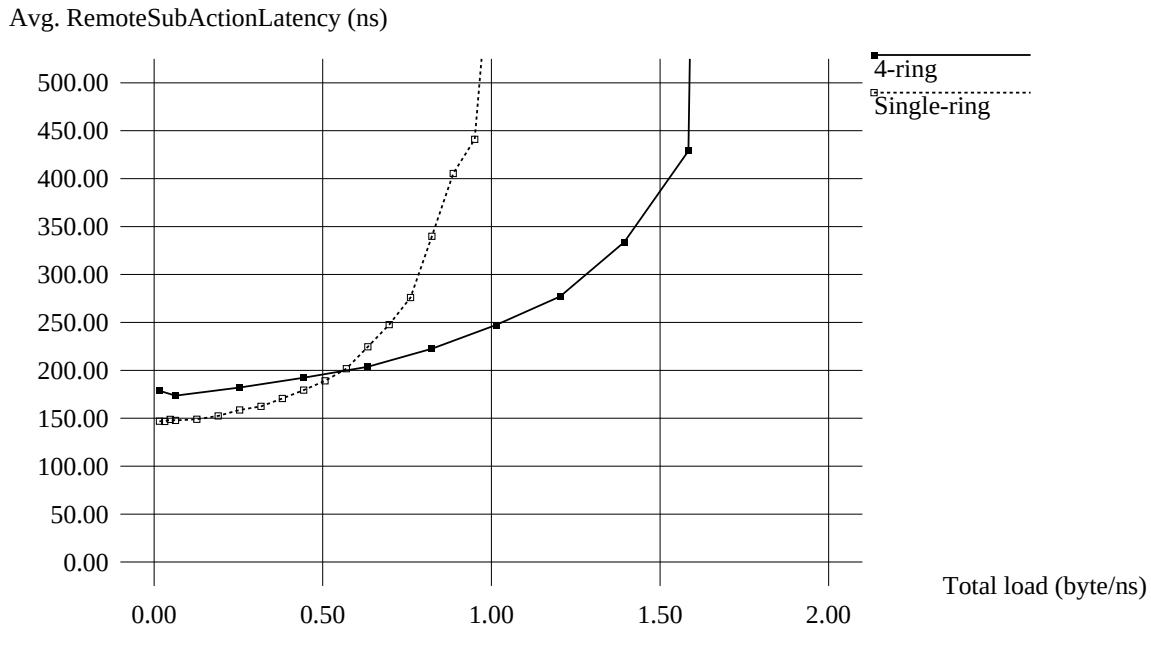
When total load is very high, exceeding $3.0byte/ns$, the rings and switches will be saturated, and the output-queues in the node will be full at all times. Therefore, the average RemoteSubActionLatency will approach a stable value. From graph 7.1, we know that total throughput will approach a stable value when total load exceeds $3.0byte/ns$.

If we then compare the 4-ring interconnect to the single-ring, we will observe the following:

- When total load is less than $0.55byte/ns$, the single-ring has a lower average RemoteSubActionLatency than the 4-ring interconnect.
- When total load lays in the interval from $0.55byte/ns$ to $1.75byte/ns$, the single-ring has a higher average RemoteSubActionLatency than the 4-ring interconnect.
- When total load exceeds $1.75byte/ns$, the single-ring has a lower average RemoteSubActionLatency than the 4-ring interconnect. In either case a maximum stable value is approached when total load gets very high, and in the 4-ring interconnect the maximum value is approximately twice the maximum of the single-ring.



(a)



(b) Zoom-in

Figure 7.2: Latency as a function of load (4-ring versus single-ring).

The results in graph 7.2 therefore indicate that the average RemoteSubActionLatency is smallest in the single-ring when load is less than a certain limit L_1. When total load exceed L_1, the situation is reversed and the average RemoteSubActionLatency of the single ring is higher than in the 4-ring interconnect. When load increase further and exceeds a second limit L_2, the situation is once again reversed and the average RemoteSubActionLatency of the single ring is less than in the 4-ring interconnect.

The results in graph 7.2 and the above conclusion is reasonable. When load is less than L_1 the fixed minimum delay in the 4-ring interconnect and the single-ring is significant. In the 4-ring interconnect a large part of the packets have to pass through one or more switches, and while the switches behave in a store-forward fashion, packets have to be fully received before they are passed on to the next ring. Therefore, the fixed minimum delay is higher in the 4-ring interconnect than in the single-ring, and the fixed minimum delay becomes significant when load is low.

When load exceeds L_1, more packets will circulate the single-ring and in the 4-ring interconnect. Packets in the single-ring will be delayed once they are on the ring because more packets are in front of it, while on the 4-ring interconnect the advantage of alternative paths becomes significant and the average RemoteSubActionLatency will be smaller in the 4-ring interconnect than in the single-ring.

When load exceed L_2, the 4-ring interconnect becomes saturated and the rings and the switches are saturated. Therefore, the delay in the switches have grown high because the input-queue of the in-going node-interface and the output-queue of the outgoing node-interface of each switch is full. In addition, the total throughput will decrease when total load exceeds $1.6 byte/ns$.

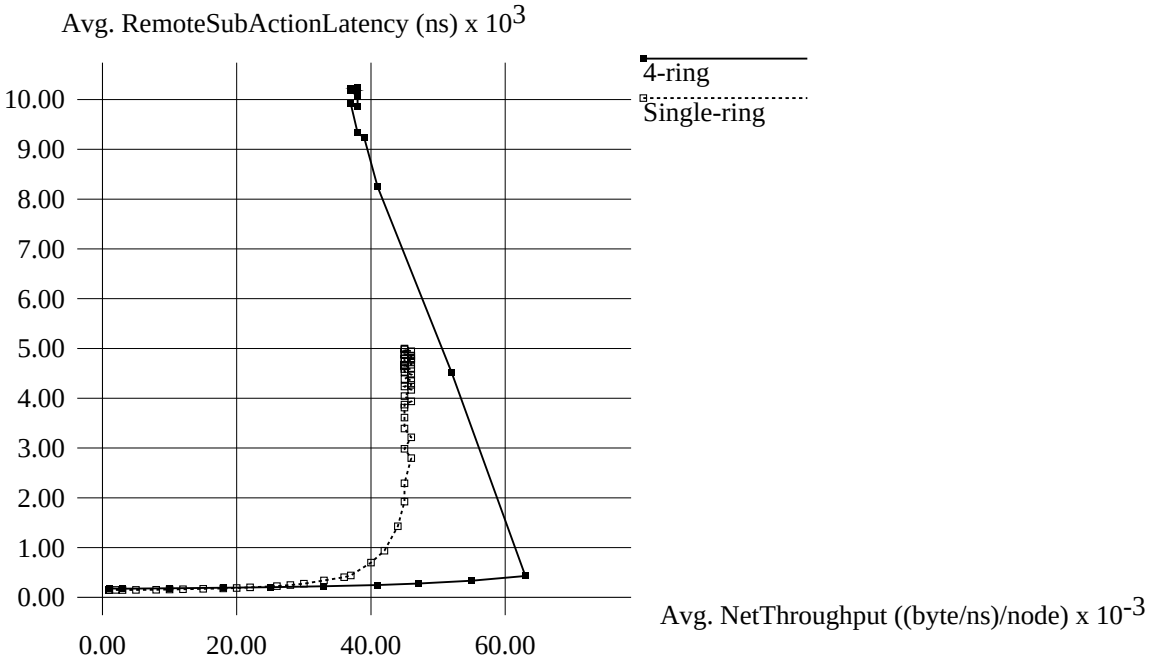
It is also reasonable that the maximum average RemoteSubActionLatency is higher in the 4-ring interconnect than in the single-ring because the total throughput is less in the 4-node interconnect than in the single-ring when load is high, and the saturated switches will only add to the delay on the interconnect.

The relationship between average NetThroughput and average RemoteSubActionLatency, graph 7.3a-b

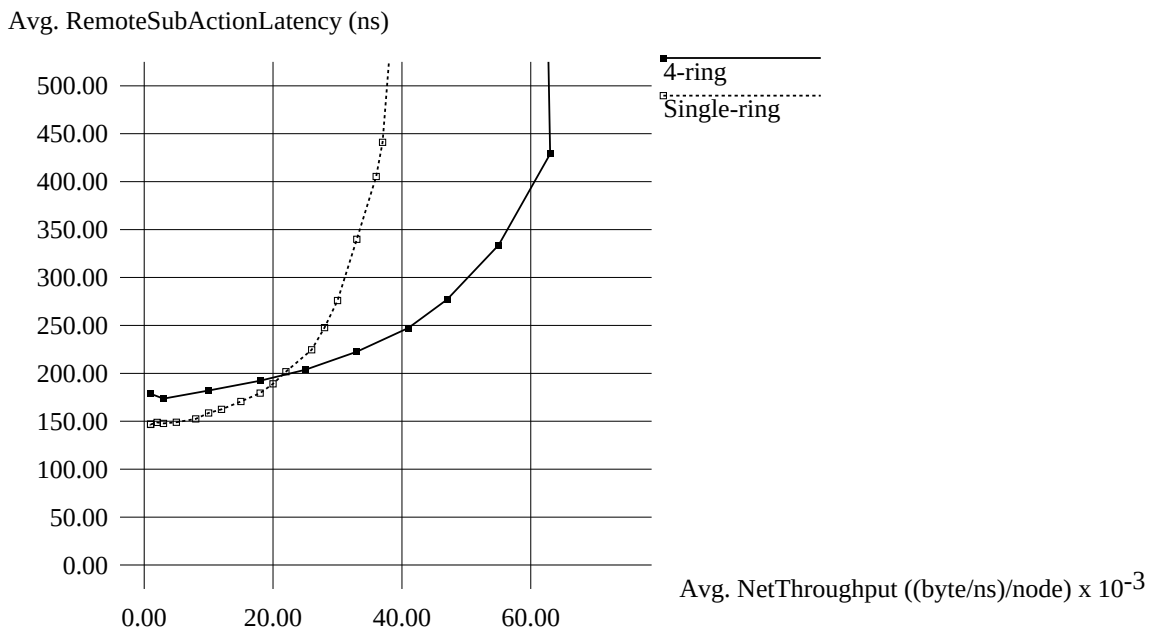
These graphs show the relationship between average NetThroughput and average RemoteSubActionLatency, in a 4-ring interconnect and a single-ring with 16 nodes. Average NetThroughput is specified in $(byte/ns)/node$ along the x-axis and the average RemoteSubActionLatency is specified in ns along the y-axis. While average NetThroughput is the total NetThroughput of the whole interconnect averaged over the nodes (16 in all) the graph show the performance of the average node, indicating how how much throughput it receive and how much time it takes. Graph 7.3b zooms in on graph 7.3a so that details are revealed.

If we observe the graphs 7.3a-b, we will see that the performance of the average node is better in the single ring than in the 4-ring interconnect when load is low. When load increase further, the situation is reversed and the performance of the average node is better in the 4-ring interconnect than in the single-ring. When load is very high, the performance of the 4-ring interconnect diminishes, and the average node in the single-ring perform better than the average node in the 4-ring interconnect.

The results in graph 7.3a-b is reasonable when we consider the results in graph 7.1 and 7.2a-b:



(a)



(b) Zoom-in on (a)

Figure 7.3: The relationship between throughput and latency (4-ring versus single-ring).

- When total load is less than $0.55 byte/ns$, the throughput is identical in the 4-ring interconnect and the single-ring, but the average RemoteSubActionLatency is lower in the single-ring. As a results, the performance of the average node is better in the single-ring than in the 4-ring interconnect in this load-interval.

- When total load lay in the interval from $0.55 byte/ns$ to $1.1 byte/ns$, the throughput is still identical in the 4-ring interconnect and the single-ring, but the average RemoteSubActionLatency is lower in the 4-ring interconnect than in the in single-ring. As a result, the performance of the average node is better in the 4-ring interconnect than in the single-ring in this load-interval.

- When total load lay in the interval from $1.1 byte/ns$ to $1.75 byte/ns$, the throughput is higher *and* the average RemoteSubActionLatency is lower in the 4-ring interconnect than in the single-ring. As a result, the performance of the average node is better in the 4-ring interconnect than in the single-ring in this load-interval.

- When total load exceed $1.75 byte/ns$, the throughput of the 4-ring interconnect will quickly get less than the throughput of the single-ring, and the average RemoteSubActionLatency will get higher also. As a result, the performance of the average node is better in the single-ring than in the 4-ring interconnect in this load-interval.

7.3 Summary

This chapter has presented and discussed simulation results from the simulation of a 4-ring interconnect, enabling us to decide on Issue 8 regarding the existence of a better way to connect 16 nodes than using a single SCI-ring (as described in chapter 3). The 4-ring interconnect consisted of 16 nodes equally distributed in 4 rings with 4 switches (figure 5.1b). The 4-ring interconnect has been compared to a single SCI-ring with 16 nodes and in both cases a uniform load and traffic pattern has been assumed. The following is a summary of the main results:

- The maximum total throughput observed in the 4-ring interconnect is approximately 38% higher than the maximum throughput observed in the single-ring.

- The total throughput in the 4-ring interconnect does not increase monotonically when load increase - it will first increase and after the maximum is reached, it will decrease and approach a stable value. The stable value of the 4-ring interconnect is 17% less than the stable value observed in the single-ring.

- The average RemoteSubActionLatency in the 4-ring interconnect increase when load increase, and are bounded upward and downward.

- The average RemoteSubActionLatency in the 4-ring interconnect is higher than in single-ring when load is low, but as load increase the situation is reversed and the average RemoteSubActionLatency in the 4-ring interconnect is less than in the single ring. When load is very high the maximum average RemoteSubActionLatency is approximately 100% higher in the 4-ring interconnect than in the single-ring.

A final conclusion on Issue 8 is given in chapter 9.

[This page has been intentionally left blank]

Chapter 8

Results from the simulation of SCI/RT

This chapter presents and discusses results from the simulation of SCI/RT, results which will help decide on Issue 9 in chapter 3. Results from the simulation of one of the modifications presented in [IEEE, 1992b] proposed in order to modify the SCI-protocol for real time purposes will be presented and discussed. As in chapter 6 and 7, the SCIsim simulator was used, and was given as input the parameters described in chapter 5.

Section 8.1 will briefly describe the parameter-values assumed during simulation and the measurements which have been emphasized in section 8.2. The latter section will present the actual results. Section 8.3 will summarize the main results related to SCI/RT.

8.1 Parameters and measurements in SCI/RT simulations

The following list describes the overall conditions assumed during simulation:

- **Topology:** Single ring.
- **Size:** 4 nodes.
- **Load and traffic pattern:** Uniform.
- **Priority distribution:** 4 priority-levels, uniformly distributed.
- **Transmitter-stage:** Packet preemption protocol.
- **Output-queue type:** Priority and preemptive queue.
- **Bypass-queue type:** Priority and preemptive queue.
- **Send-packet size:** Mixed.

The load was increased from one simulation to another - starting with a value close to zero and going up to a level where the throughput and latency measurements had stabilized. The length of one simulation was determined after some simulations (Refer to section 5.3) and it was found that simulation time equivalent to $1000000ns$ produced acceptable results. The remaining parameter-values assumed during simulation can be found in table 5.3 in section 5.1.3. The term **SCI/RT-ring** will refer to the simulated ring. Packets are

generated uniformly on four priority-levels **1-4**, and where 4 is the highest priority-level and 1 is the lowest. The terms **Pri1-Pri4** will also be used when referring to the various priority-levels. Because the priority distribution of new packets is uniform, the number of new packets generated on each priority-level are approximately equal.

In order to indicate the throughput of the SCI/RT-ring, this section will emphasize the **AckTransThroughput** as defined in section 5.2. This throughput measurement includes those send-packets transmitted by a node-interface which are acknowledged by the receiving node-interface. For each priority-level, total AckTransThroughput (the sum of AckTransThroughput of each node) will be presented.

In order to indicate the latency of the SCI/RT-ring, this section will emphasize **LocalSubActionLatency** as defined in section 5.2. This latency measurement equals the time from the send-packet is inserted into the output-queue of the transmitting node-interface until a corresponding DONE-echo is received (whereupon the send-packet is removed from the output-queue). For each priority-level, average LocalSubActionLatency will be presented.

8.2 SCI/RT results

This section will present and discuss the results in the following order:

- Throughput as a function of load for each priority-level, graph 8.1.
- Latency as a function of load for each priority-level, graph 8.2.a-b
- The relationship between throughput and latency for each priority-level, graph 8.3a-b.

These SCI/RT results will be discussed and evaluated according to the criteria proposed in section 3.2.3, where issues related to SCI/RT were presented.

Total AckTransThroughput as a function of total load, graph 8.1

This graph shows for each priority-level, 1-4, the total AckTransThroughput as a function of total load (sum of load on all priority-levels). Load and throughput is specified in $byte/ns$ along the x-axis and y-axis respectively. Note that total load equals the sum of load in each node, regardless the priority. While new packets are generated uniformly on 4 priority-levels, 25% of total load is on priority-level 1, 25% of total load is on priority-level 2 etc. When total load exceeds $35 byte/ns$ (not shown in graph 8.1), simulations indicate that the results in graph 8.1 for total load of $30 byte/ns$ and higher, is representative for even higher loads also.

Observing graph 8.1 we will see the following:

- When total load is less than $1.3 byte/ns$, the total AckTransThroughput in each case Pri1-Pri4, will increase when total load increase. Total AckTransThroughput is also approximately equal in the four cases. When total load is approximately $1.3 byte/ns$, the sum of total AckTransThroughput is:

$$\sum_{i=1}^{4} AckTransThrougput_{Pri_i} = 0.29 + 0.30 + 0.31 + 0.29 = 1.19 byte/ns$$

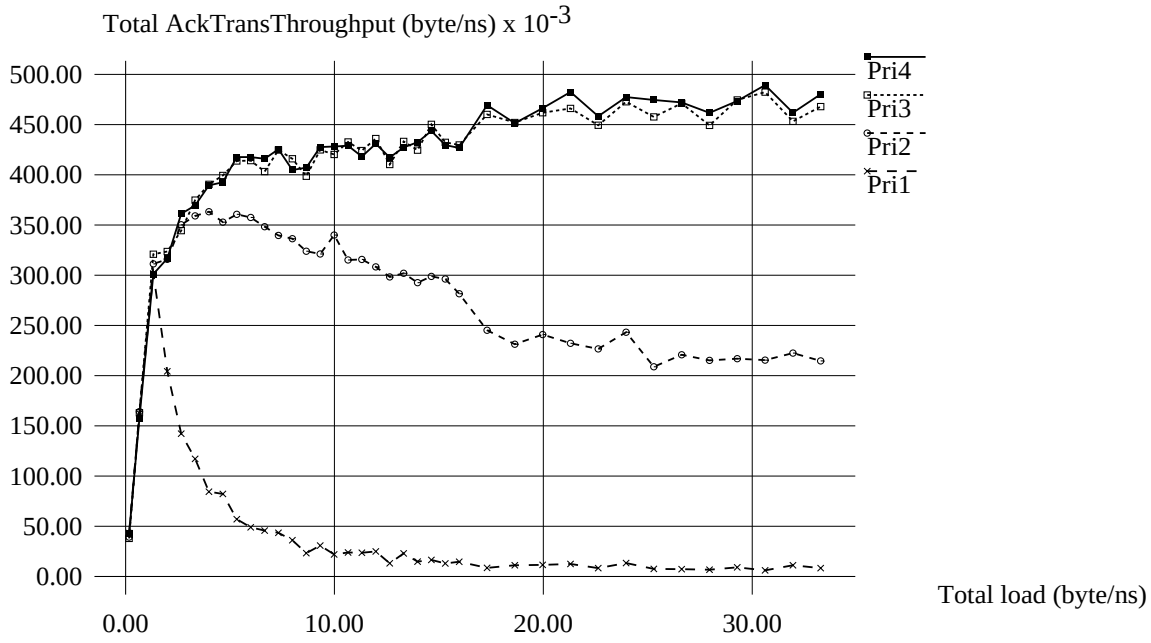


Figure 8.1: Throughput as a function of load (SCI/RT-ring).

- When total load exceeds $1.3byte/ns$, the total AckTransThroughput in each case differ significantly. The total AckTransThroughput is higher in the Pri3 and Pri4 case than in the Pri1 and Pri2 case. Total AckTransThroughput is also higher in the Pri2 case than in the Pri1 case. In the Pri1 case, total AckTransThroughput will decrease when total load exceeds $1.3byte/ns$, and will approach $0.0byte/ns$ when total load is very high. In the Pri2 case, the total AckTransThroughput will increase until total load exceeds approximately $5.0byte/ns$ and will decrease and approach a stable value of $0.21byte/ns$ when total load increases further. The total AckTransThroughput in the Pri3 and Pri4 case are approximately equal and will increase when total load increases, and will approach a stable value of approximately $0.47byte/ns$ when load is very high.

The results in graph 8.1 indicate some properties of an SCI/RT-ring in terms of throughput. When total load is less than $1.3byte/ns$, throughput (acknowledged by the receiving node) on the four priority-levels are approximately equal, and all packets, regardless of their priority, will reach their destination. This mean that the priority distributions of packets transmitted ***and*** acknowledged by the target node, equals the priority distribution of new packets when load is low.

When load is higher than $1.3byte/ns$, the throughput on the lowest priority-level, 1, is less than the throughput on higher priority-levels, and will approach $0.0byte/ns$. The throughput on the higher priority-levels, 2-4, will approach a stable value. This mean that the priority distribution of packets transmitted ***and*** acknowledged differ from the priority distribution of new packets with low priorities, 1 and 2, and resembles the priority distribution of new packets with high priority, 3 and 4. Nevertheless, the results in graph 8.1 is somewhat surprising because when total load exceeds $10byte/ns$, 25% of that is packets on priority-level 4 and these packets should fill the ring completely. 25% of $10byte/ns$ is $2.5byte/ns$ and this exceeds total throughput of the ring (refer to graph 6.1a) and should

eventually prevent lower priority packets from entering the ring.

It is reasonable when load is low, that the throughput on each priority-level is approximately equal assuming a uniform priority distribution of new packets. When load is low, the output-queues and bypass-queues rarely fill up and few packets, primarily lower priority packets, will be preempted. Consequently, all packets will reach their destination.

When load is high, the output-queues and the ring begins to fill up. Higher priority packets will have the right of way, and will more quickly reach their destination than the lower priority packets.

It is more difficult to explain why the throughput on priority-level 2 and 3 approach a stable non-zero value, rather than approach $0.0 byte/ns$, and that priority-level 4 does not consume all bandwidth. One possible explanation is that only retry and unsent packets in the output-queue are preempted and outstanding packets are unaffected, when a high priority packet try to gain access to a full output-queue. Lower priority packets will be preempted in the bypass-queue when load is high and because echo-packets in these simulations inherit the send-packet's priority, echo-packets will be passed by packets with a higher priority. In this way echo-packets will spend a long time in the ring before they are back at the node that transmitted the send-packet, and outstanding lower priority packets will stay longer in the output-queue than outstanding higher priority packets. In this way precious queue space is held up by low priority send-packets.

The results in graph 8.1 therefore indicate that priority output-queue and bypass-queue w/preemption, meet only a part of the requirements in 3.2.3. When load is substantially higher than the total throughput of the ring, it is reasonable to expect that only Pri4 packets should circulate the ring. Nevertheless, the results indicate that the throughput is higher for higher priorities than for lower priorities.

Average LocalSubActionLatency as a function of total load, graph 8.2a-b

These graphs show for each priority-level 1-4, the average LocalSubActionLatency as a function of total load. Load is specified in $byte/ns$ along the x-axis and latency is specified in ns along the y-axis. Graph (b) zooms in on (a) to reveal details otherwise unseen. When total load exceeds $35 byte/ns$ (not shown in graph 8.2a-b), simulations indicate that the results in graph 8.1, when total load exceeds $30 byte/ns$, are representative for higher loads also.

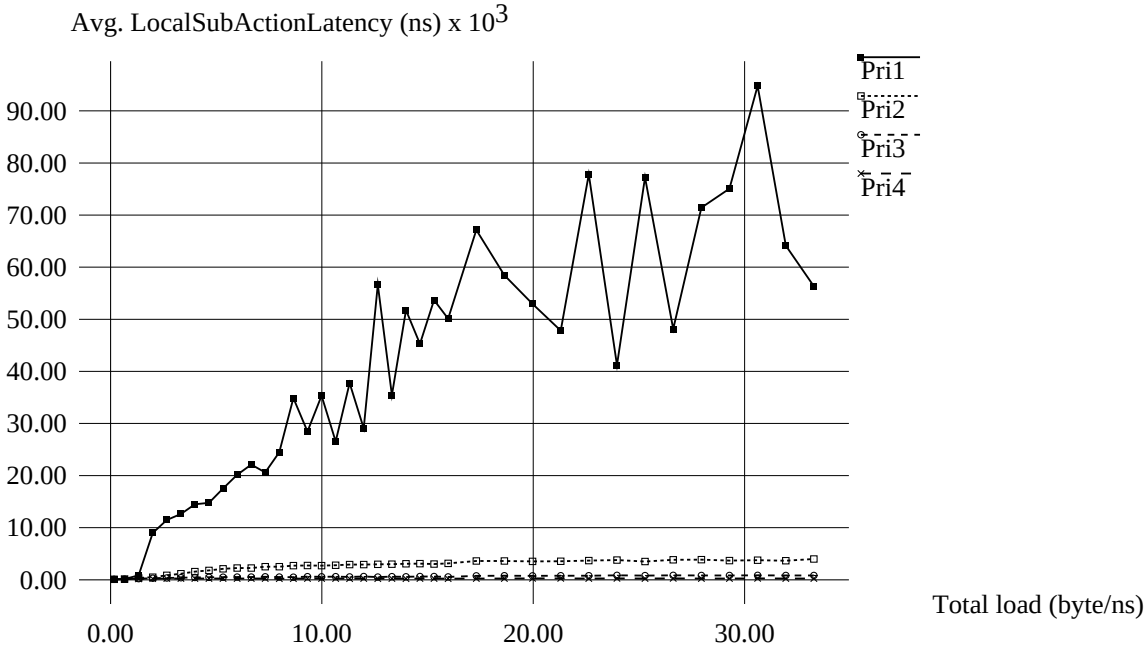
Observing graph 8.2a-b, we will see the following:

- When total load is less than $0.7 byte/ns$, the average LocalSubActionLatency in each of the four cases Pri1-Pri4, lay in the interval from $115 ns$ to $150 ns$. The highest priority has the lowest latency.

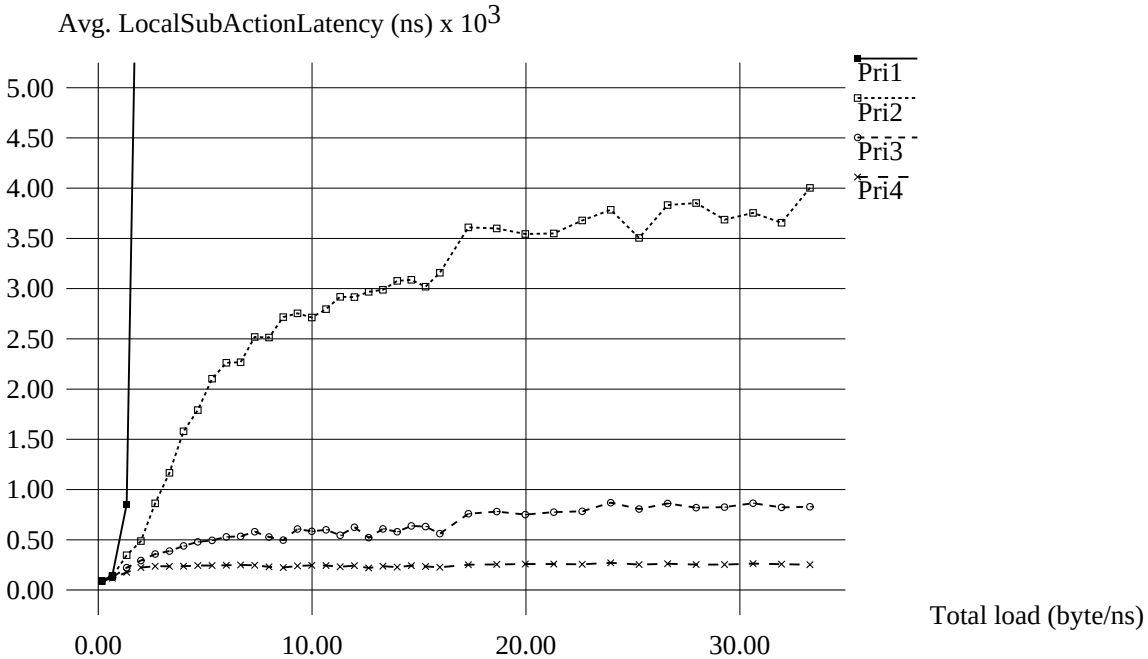
- When total load exceeds $0.7 byte/ns$, the average LocalSubActionLatency in each case differ significantly, and are higher for lower priorities than for higher priorities. Sorted in increasing order of latency, the priority-levels are: Pri4, Pri3, Pri2, Pri1.

  Average LocalSubActionLatency in the Pri1 case will increase when load increases, but when load exceeds $10 byte/ns$ the results are rather jumbled, but still indicate a significant higher latency than in the three cases Pri2-Pri4. The latter cases seem to approach a stable maximum value of $3700 ns$, $800 ns$ and $250 ns$ respectively.

The results in graph 8.2 indicate some properties of the SCI/RT-ring in terms of latency. When load is low, the average LocalSubActionLatency is approximately equal on the four



(a)



(b) Zoom in on (a)

Figure 8.2: Latency as a function of load (SCI/RT-ring).

priority-levels. This mean that packets, regardless of priority, spend approximately the same amount of time in the ring before they are acknowledged by the target node.

When load is high, the latency on a high priority-level is less than the latency on a lower priority-level. The latency is less on priority-level 4 than on any other priority-level and priority-level 1 has the highest latency of all. Even when load increases, the average latency of priority-level 4 is stable and will approach $250ns$. This meets the requirements in 3.2.3, where it was required that latency of higher priority packets should be less than the latency of lower priority packets, in general and when load is high.

The results in graph 8.2a-b are reasonable considering the results in graph 8.1. In the latter graphs we observed that the throughput was approximately equal on the different priority-levels when load was low, and therefore the latency will be approximately equal. When load is low, there is also few packets on the ring, so a low priority packet will rarely be passed by packets with a higher priority.

When load exceeds $1.3byte/ns$, we observed in graph 8.1 that throughput on priority-level 1 was less than the throughput on priority-level 2-4, and throughput on priority-level 2 was less than on priority-level 3-4. Because the priority of new packets are uniformly distributed, the latency of low priority packets will be higher than the latency of higher priority packets simply because less bandwidth is available to lower priority traffic.

We also observed in graph 8.1 that the total throughput of priority-level 3 and 4 were approximately equal, but the average LocalSubActionLatency is still approximately 50% - 70% less on priority-level 4 than on priority-level 3. This indicate that priority 4 packets pass lower priority packets once they are one the ring.

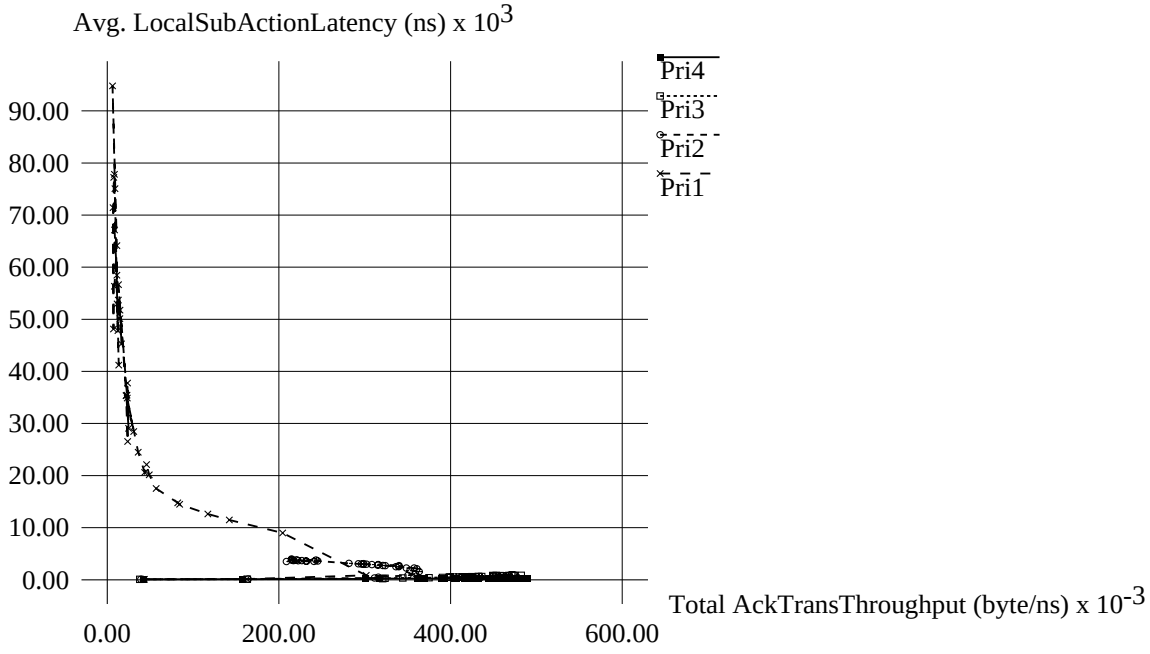
The relationship between total AckTransThroughput and average LocalSubActionLatency, graph 8.3a-b

These graphs show for each priority-level, 1-4, the relationship between total AckTransThroughput and average LocalSubActionLatency. Throughput is specified in $byte/ns$ along the x-axis and latency is specified in ns along the y-axis. The graph therefore indicate the performance of each priority-level, and associated with good performance is high throughput and low latency. Graph (b) zooms in on (a).

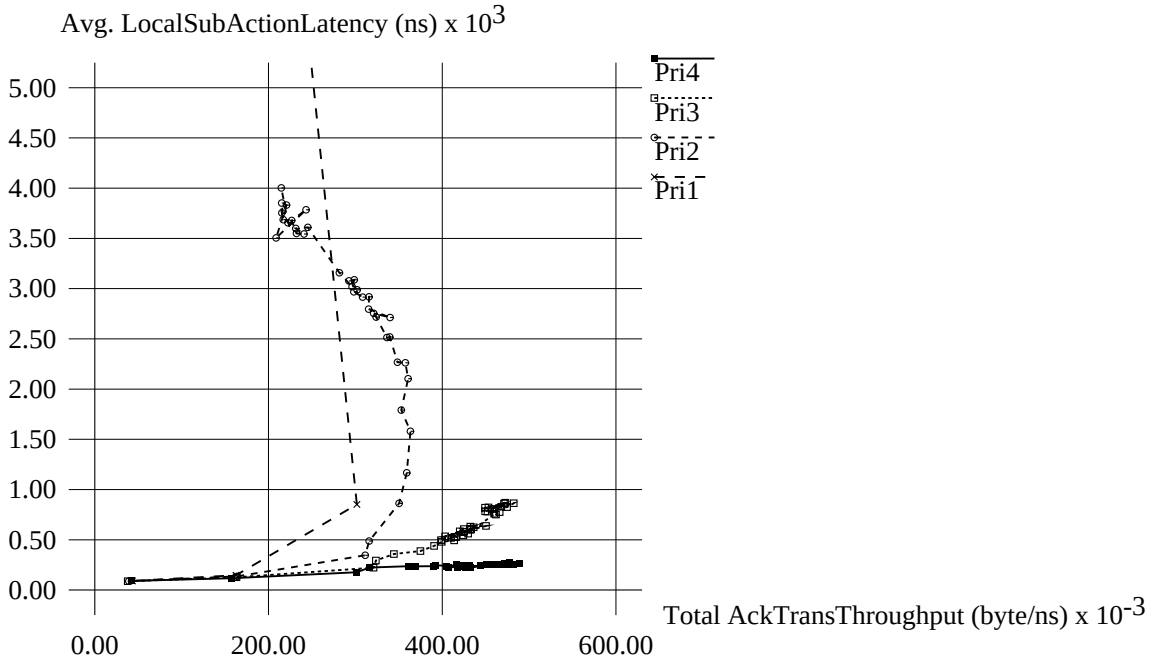
Observing graph 8.1a-b we see that the performance is better on priority-level 4 than on the lower priority-levels. Sorted in decreasing order of performance, the priority-levels are: 4, 3, 2, 1. In particular, the performance of priority-level 1 is very bad when load is high.

The results in graph 8.1a-b are reasonable considering the results in graph 8.1 and 8.2. In the two latter graphs, we observed that the AckTransThroughput is higher and the average LocalSubActionLatency is lower on priority-level 4 than on the other priority-levels. Consequently priority-level 4 has a better performance than priorities 1-3.

For priority-level 1 we observed in graph 8.1 and graph 8.2 that the AckTransThroughput decreased when total load exceeded $1.3byte/ns$ and that the average LocalSubActionLatency increased and become significantly higher compared to priority-level 2, 3 and 4. As a result the performance of priority-level 1 is worse than the performance on priority-level 2, 3 and 4.



(a)



(b)

Figure 8.3: The relationship between throughput and latency (SCI/RT-ring).

8.3 Summary

This chapter has presented and discussed results from the simulation of an SCI/RT-ring with 4 nodes, in this way enable us to decide on Issue 9 as described in chapter 3, regarding whether the modification consisting of preemptive priority output-queues and bypass-queues, controlled by the packet preemption protocol (chapter 2), meet the requirements in the definition of the SCI/RT project (chapter 2). The following is a summary of the main results related to SCI/RT:

- When load is low, the priority distribution of packets transmitted *and* acknowledged equals the priority distribution of new packets. When load is high the priority distribution of packets transmitted *and* acknowledged equals the priority distribution of new packets for the highest priorities only. However, the simulation results indicate that the highest priority-level does not consume all bandwidth when the load is very high (So high that the load on the highest priority-level exceed the maximum total throughput of the ring). Instead the throughput on the second and third highest priority-level approached a non-zero value.
- The average **LocalSubActionLatency** is approximately equal when load is low, but when load is high the latency of one priority differ from the other. The latency is higher for lower priority-levels than for higher priority-levels. Packets on priority-level 4 have a lower average latency than packets on priority-level 1-3.

A final conclusion on Issue 9 is given in chapter 9.

Chapter 9

Conclusion of the thesis

This chapter contains the conclusion of the thesis. The elements considered important here are to conclude on the thesis' original issues described in chapter 3, to indicate additional results and to indicate further work. Conclusions are based on results and observations presented in chapter 4, 6, 7 and 8.

Because the original issues of this thesis, described in chapter 3, are either related to the design and building process of the simulator or related to the performance of SCI and SCI/RT, this chapter contains two main sections.

9.1 Conclusion on issues related to the design and building process of the simulator

This section will present and discuss the conclusion on the issues related to the design and building process of the simulator. The conclusions are based on chapter 4, where the programming strategy and a part of the implementation of the SCIsim-simulator was presented.

Section 9.1.1 will conclude on the original issues presented in the beginning of the thesis (section 3.2.1). Section 9.1.2 will discuss additional results discovered when the simulator was designed and section 9.1.3 will present further work and an alternative approach to designing a simulator for the SCI-protocol.

9.1.1 Conclusion on the original issues

The issues in section 3.2.1 were expressed as questions and these will be repeated in the following discussion:

Issue 1: Is it possible to design a simulator for the SCI-protocol which is flexible and modifiable, so that future modifications to the SCI-protocol can be simulated without extensive re-design?

A simulator for the SCI-protocol has been designed and the design process was described in chapter 4. The simulator considers a subset of the SCI-protocol (packet transportation layer, refer to section 2.2.3) and some of the modifications proposed in relation to SCI/RT have been incorporated.

Chapter 4 argued that not only had the simulator to be modifiable and flexible, but also correct, parameterized, effective and enable performance analysis. In the

following it will be discussed whether the final SCIsim-simulator fulfills the criteria in chapter 4.

The implementation of the SCI/RT-modifications indicates that the SCIsim-simulator is modifiable and flexible. The work went fairly easy and the problem was understanding the intuitive idea rather than incorporating it into the simulator. Subclasses representing the SCI/RT-bypass-queue, SCI/RT-output-queue and SCI/RT-transmitter-stage were designed. Also the representation of the packet concept was changed half-way during the design process because the simulator had to be speeded up. Altering the representation of the packet concept did not affect the classes representing the output-queue and the input-queue. What may have reduced the flexibility and modifiability of the SCIsim-simulator is the fact that symbols are simulated by integers, and symbols often have to carry some kind of time-stamps during simulation. Consequently it may be difficult to add new latency measurements in the future.

The simulator is reasonably correct. Various simulations have been performed and some of them under assumptions similar to those in [Scott et.al., 1992]. Results in [Scott et.al., 1992] resemble those presented in chapter 6 (refer to section 6.2.4). The simulation results in chapter 6, 7 and 8 have also been discussed thoroughly and found reasonable.

The SCIsim-simulator is also parameterized meaning that each node, node-interface and link can be specified independently, in order to simulate various conditions. The performance of various SCI-interconnects can be analyzed because the simulator calculate various measurements during simulation. Some measurements do not correspond 100% to others, for example the RecThroughput does not correspond 100% to LocalSubActionLatency (which they should) because some send-packets may have been used in the calculation of the of RecThroughput and not yet in the calculation of LocalSubActionNoEchoLatency. However, the error will be smaller for longer simulations.

The efficiency of the simulator is acceptable and simulations which involve much dynamic allocation tend to be slower than simulations that involve less. Simulation time also increases when the interconnect grows in size.

If we assume that the modifications of the SCI-protocol affect the internal structure of a real-world entity (e.g. bypass-queue), it is reasonable to conclude that it is possible to design a flexible and modifiable simulator for SCI. The implementation of a class representing a real-world entity can be changed in an object-oriented program without affecting the rest of the program.

Issue 2: How successful is the object-oriented programming strategy when simulating SCI, when modifications of the SCI-protocol are simulated also?

An object-oriented programming strategy was used when the SCIsim-simulator was designed. The strategy was described in section 4.2 and the historical development and implementation was described in section 4.3.

The programming strategy was chosen because it was believed that it would help design a simulator meeting the criteria proposed in chapter 4. During the early stages in the design process the strategy was followed diligently, but when the efficiency proved to be lacking, symbols were represented by integers rather than class-objects.

```
class A;                        class B;
begin                           begin

  ref(B) rB;                      ref(A) rA;

  procedure P1; <..>              procedure P2; <..>

  .                               .
  .                               .
  rB.P2;                          rA.P1;
  .                               .
  .                               .

end;                            end;
```

Figure 9.1: Two mutually dependent classes.

Symbols had previously been represented by class objects, and using integers meant that an ad-hoc strategy had been employed.

In other respects the programming strategy helped design a simulator which were modifiable, flexible, correct, parameterized and which enabled performance analysis. The SCI/RT modifications were fairly easy to incorporate and was incorporated into the simulator by designing sub-classes for the SCI/RT-bypass-queue, SCI/RT-output-queue and SCI/RT-transmitter stage.

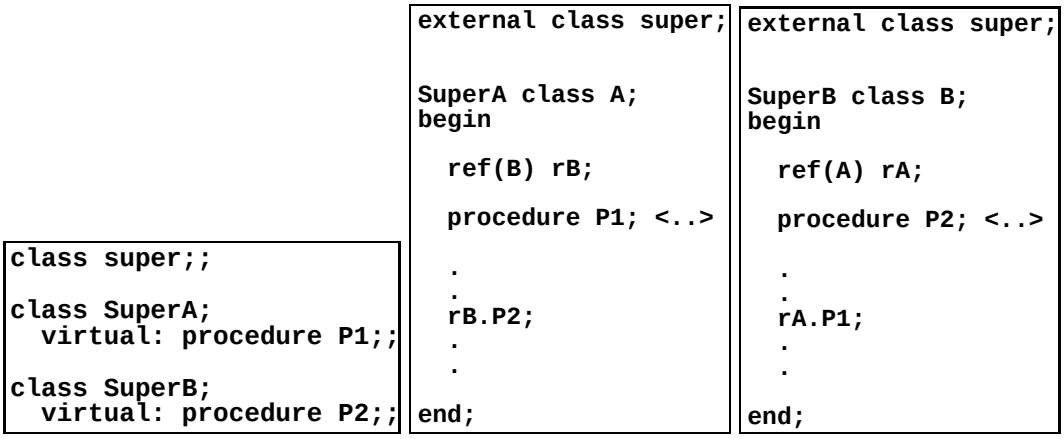
It is therefore reasonable to conclude that the programming strategy was fairly successful. All requirements, except the one concerning efficiency, were met. When it was clear that the simulator was too slow, symbols were represented by integers. This was an ad-hoc strategy which, at one point, violated the original object-oriented programming strategy.

Issue 3: How does the Simula programming language affect the design process of the SCI-simulator in general, and the object-oriented programming strategy in particular?

The SCIsim-simulator was written almost entirely in Simula. Procedures which enable bit-pattern manipulation were written in C. Part of the implementation was described in chapter 4.

Simula affected the design process of the SCIsim-simulator when the run-time system proved to be too slow. Because one of the requirements toward the simulator concerned efficiency, an ad-hoc strategy was employed and it was decided to represent symbols by integers. Nevertheless, Simula supported the programming strategy to a large extent and with its class and sub-class construct, real-world entities and concepts could be represented by classes. Variation within the same class of entities could be represented by various sub-classes.

When a large program is designed according to an object-oriented programming strategy, it is often desirable to have the program textually divided into smaller files, so that each file contains a single class or related classes. This will not only make the program more manageable when editing it, but will also emphasize the classes as separate, logical entities. A class in one file could be compiled separately and then the classes which depended on it could be recompiled without having to recompile the whole program. In general, classes in Simula can be compiled separately, but



```
class super;;

class SuperA;
  virtual: procedure P1;;

class SuperB;
  virtual: procedure P2;;
```

(a) File 1

```
external class super;


SuperA class A;
begin

  ref(B) rB;

  procedure P1; <..>

  .
  .
  rB.P2;
  .
  .

end;
```

(b) File 2

```
external class super;


SuperB class B;
begin

  ref(A) rA;

  procedure P2; <..>

  .
  .
  rA.P1;
  .
  .

end;
```

(c) File 3

Figure 9.2: A solution to the problem of mutually dependent classes.

when two classes are mutually dependent, they have to be compiled together and reside textually in the same file (figure 9.1). As a result, it is not always possible to textually divide a program into a set of files so that each file contains a single class which can be compiled separately.

When the SCIsim-simulator was designed, an attempt was made to divide the program into separate files which then could be compiled separately. This proved difficult because several classes were mutually dependent and had to reside textually in the same file. A part-wise textual partitioning was achieved by putting the classes into separate files and prior to a compilation merge the files into one. The C-preprocessor [Kernighan and Ritchie, 1988] was used for this purpose.

One way to divide textually the two classes in figure 9.1 is shown in figure 9.2. In the context of SCI-simulations the idea is as follows:

- Define a super-class with virtual procedures representing the various entities and concepts in a real-world SCI-interconnect. These super-classes could textually reside in one file and be compiled together.
- Define a sub-class each super-class for every entity-type in the SCI-interconnect. Each sub-class could textually reside in one file and be compiled separately.

To conclude on Issue 3, the Simula programming language affected the design process because of its lacking efficiency in certain situations (heavy use of dynamic allocation) and special measures had to be taken in order to speed up the program. In doing so, the object-oriented programming strategy was violated, by simulating symbols by integers rather than class-objects. This increased the speed considerable, but later, when SCI/RT modifications were incorporated, this also proved to have reduced the modifiability and flexibility, because it became more difficult to add latency measurements.

9.1.2 Other results

Designing the SCIsim-simulator was laborious, even though only a subset of the SCI-protocol was considered (the packet transportation layer). One has to understand the SCI-protocol thoroughly, which require considerable effort, and then has to decide how to represent it in the simulator. The SCI-protocol is complex, contains a lot of details and above all, describes only a protocol and not how to implement it. Some implementation issues are rather obvious, eg. that the bypass-queue should be represented by a FIFO-queue containing symbols, while other issues are far less obvious, eg. how to represent the ring bandwidth allocation protocol and input-queue allocation protocol (refer to section 2.2.3).

Designing the SCIsim-simulator also indicates that it benefits both the understanding of SCI and the process of representing it, to work on issues related to the SCI-protocol and the SCI-simulator simultaneously. To understand the SCI-protocol is of course mandatory when an SCI-simulator is designed, but the design process gives something in return. Designing a program creates an awareness for details and special cases which prove useful when complex parts of the SCI-protocol are approached.

As required, the SCIsim-simulator is parameterized and enable performance analysis. Each simulation can be specified in detail and the resulting simulation can be analyzed in detail also — the simulator gather a large amount of data during simulation. As soon as simulation-work began it became clear that this scheme was not very user-friendly, e.g. 30-40 specification-files had to be written in order to simulate an increase in load.

If a simulator is highly parameterized and generates extensive simulation-results, there should be a way to specify fixed parameters once and for all, and only those results which are interesting should be visible. The specification-files which were given to the SCIsim-simulator as input, were written according to a simple context-free grammar and one solution to the above problem - having to specify every parameter every time - could have been to design a program which generated specification-files according to this grammar. A fixed set of parameters could be specified once and for all, and only those parameters still unbounded had to be specified each time.

The SCIsim-simulator got rather big, and in hindsight it is clear that some details are superfluous. Nevertheless, a well-working simulator was needed because another important issue was to analyze the performance of SCI and SCI/RT, and this seemed to speak *for* prolonged effort when it was felt that enough time and effort was spent already.

9.1.3 Further work

If a new simulator was going to be designed at this stage, for the SCI-protocol and some of the SCI/RT modifications, two different approaches would be considered.

The first approach is identical to the object-oriented programming strategy described in section 4.2.2. Departing from or violating the strategy would not be accepted, because it is considered more important to ensure modifiability and flexibility than downright performance. If dynamic allocation proved to be massive and caused slow simulations, symbol-objects could be re-used rather than burdening the run-time system with constant allocation and deallocation. Also a consistent strategy to perform measurements would be developed and used from the very start.

The second approach would be to focus on the packet entity, the entity which gives raise to latency and throughput. This approach was proposed by Stein Gjessing at UiO during the early stages of the work (refer to section 4.2.5) but was abandoned because it

was not obvious how to ensure correctness. A substantial amount of the effort involved at the time was focused on understanding SCI, and this second approach seemed to take a thorough understanding of the SCI-protocol for granted. At *this* stage however, the strategy is more likely to succeed, because in the meantime, a better understanding of SCI has been acquired.

Either of the two above approaches has advantages and disadvantages. We have seen in this thesis, things that would indicate a modifiable and correct simulator if the first approach is used — the SCIsim-simulator is modifiable and correct (among other properties). On the other hand, re-using symbol-objects is not exactly in the spirit of object-oriented programming because allocation and deallocation of objects is the task of the run-time system and should be invisible to the programmer.

The second approach may lead to less code and a faster program because a lot of the actions performed in the SCIsim-simulator would be left out (they would not be needed). On the other hand, it is not obvious how to ensure correctness without a renewed analysis of the SCI-protocol, this time from the packet point-of-view.

Adding the cache coherence protocol on top of the simulator for the packet transportation layer, would make the SCI-simulator complete, and real programs could be the main source of input.

9.2 Conclusion on the issues related to the performance of SCI and SCI/RT

This section will present and discuss the conclusion on the issues related to the performance of SCI and SCI/RT. Conclusions are based on chapter 6, 7 and 8, where simulation results related to single SCI-rings, a multi-ring interconnect and SCI/RT-rings were presented.

Section 9.2.1 will conclude on the original issues related to the performance of SCI and SCI/RT (section 3.2.2 and 3.2.3). Section 9.2.2 will discuss additional results discovered during simulation. Section 9.2.3 will discuss further work.

9.2.1 Conclusion on the original issues

The issues in section 3.2.2 and 3.2.3 were expressed as questions, and these will be repeated in the following discussion:

Issue 4: What affects the performance of single SCI-rings, with less than 16 nodes?

Single SCI-rings have been simulated under various conditions, including ring size, load and traffic pattern, flow control and packet size. The results from these simulations were presented in chapter 6.

The total throughput of a uniform SCI-ring will increase when load increases, and will approach a stable value. This means that the total throughput of an SCI-ring remain stable under transient overload (Refer to section 6.2.4).

The average latency of a uniform SCI-ring is bounded upward and downward, and the upper bound remain stable even when load is high. This means that the maximum average latency is stable during transient overload (Refer to section 6.2.4). The maximum average LocalSubActionLatency and LocalSubActionNoEchoLatency depends on the maximum output-queue size.

The SCI flow control mechanism will, regardless whether the load and traffic pattern is uniform, hot sender or node starvation, reduce the maximum total throughput and increase the average latency of the ring, compared to an identical ring without flow control. How much the throughput is reduced and latency is increased depends on the size of send-packets transmitted in the ring. When either 16byte packets or 80byte packets are transmitted, the maximum total throughput is reduced with approximately 15% and maximum average latency is increased with approximately 28% (Refer to section 6.2.4, 6.3.2 and 6.4.2).

An SCI-ring transmitting small send-packets will have a lower average latency compared to an SCI-ring transmitting larger send-packets, when the two rings are identical and the load is low. The situation is reversed when load is high, and the SCI-ring transmitting large send-packets will have a lower latency than the SCI-ring transmitting smaller send-packets. Large send-packets also mean a higher maximum total throughput and a lower average latency than smaller send-packets (Refer to section 6.2.4).

Increasing the number of nodes in a uniform SCI-ring will not increase the total throughput, but will instead increase the average latency in general, and the maximum average latency in particular.

To conclude on Issue 4: The load, the size of send-packets, the type of flow-control mechanism and the ring-size affects the performance of an SCI-ring. A non-uniform load and traffic pattern will affect the performance of each node individually in an SCI-ring (more of that in relation to Issue 6).

Issue 5: Is the SCI-ring scalable, when the number of nodes are less then 16?

SCI-rings of size 4 and 16 have been simulated, with or without flow control and assuming various sizes of the send-packets. The main results were presented in section 6.2.3.

We observed that total throughput was approximately equal (differing no more than 5% in favor of the smaller ring), and that the latency was generally higher in the bigger ring. The performance of the average node is also higher in the smaller ring than in the bigger ring, transmitting more with a lower latency (Refer to section 6.2.4).

To conclude, the SCI-ring is not scalable in terms of throughput and latency. If the SCI-ring was scalable, the throughput of the ring would increase when the number of nodes increased. When we observe that the throughput does not increase, but instead observe that the latency increases, we must conclude that the SCI-ring is not scalable. This conclusion apply to the ring structure and the packet transportation layer, and a 16-node SCI-ring may still be a better choice than a 4-node SCI-ring. If for example, the processors and memory chips did not have the speed to fully utilize a 4-node ring, more processors and memory chips could utilize a 16-node ring fully (or an even bigger ring).

Issue 6: Does the flow control mechanism specified in the SCI-protocol ensure fairness among the nodes?

SCI-rings of size 4 and 16 have been simulated, with or without flow control and which display non-uniform load and traffic patterns, referred to as hot-sender and

node-starvation (Refer to section 5.1.2 for further details). Section 6.3 and section 6.4 presented results from these simulations.

Related to hot sender we observed in the ring without flow control, that the downstream neighbours of the hot sender were affected depending on their distance from the hot node. The closest (downstream) node was more affected than the other nodes and had a higher latency and slightly lower throughput. Adding flow control eliminated the difference in throughput and latency among the non-hot nodes, both average and worst case (Refer to section 6.3.2).

Related to node starvation we observed in the ring without flow control, that the starved node was not affected when load was low, but was severely affected when load was higher, and its the transmitted throughput approached zero and the latency was unbounded. Adding flow control helped the starved node, and it was able to transmit something, though slightly less than the non-starved nodes, and both the average and worst case latency was bounded (Refer to section 6.4.2).

The results from simulating non-uniform load and traffic patterns therefore give good reason to conclude that the SCI flow control mechanism ensures fairness among the nodes. When flow control is added to the ring, a prize is paid in terms of reduced maximum total throughput and increased average latency.

Issue 7: Are the results achieved using the simulator developed in this thesis comparable to results in [Scott et.al., 1992]?

Some of the simulations were performed under conditions which were made as close as possible to those conditions assumed in [Scott et.al., 1992]. A direct comparison to the results in [Scott et.al., 1992] was carried out in the end of section 6.2.2 and 6.2.3.

There we observed that the results achieved using the SCIsim-simulator resembled the results in [Scott et.al., 1992] and either case indicate the same properties of the SCI-ring towards packet size, flow control and ring size (Issue 4). The quantitative differences in direct comparison were sometimes small, ranging from 3% to 9%, and sometime bigger. The biggest difference were in the results related to flow control (Refer to 6.2.4).

To conclude, the results in [Scott et.al., 1992] resembled those presented in chapter 6, and both indicate the same properties of the SCI-ring towards flow control, packet size and ring size. When results were compared directly, the quantitative differences varied.

Issue 8: Is there a better alternative than using a single SCI-ring interconnect if we were going to connect 16 nodes?

A 4-ring interconnect with 16 nodes and 4 switches were simulated and compared to a 16 node SCI-ring. Except for the difference in interconnect structure the conditions assumed during simulation were identical, among them a uniform load and traffic pattern. Chapter 7 presented the results from the comparison.

We observed that the total throughput of the 4-ring interconnect was not stable when total load increased, and actually decreased after it had passed a temporary peak. The maximum total throughput was nevertheless higher in the 4-ring interconnect than in

the single-ring, but the stable total throughput was lower in the 4-ring interconnect than in the single-ring when load was very high.

Up to the point when the 4-ring interconnect became saturated, the latency was lower in the 4-ring interconnect than in the single-ring (except when load was very low, where the latency was lower in the single-ring than in the 4-ring due to the store-forward switch).

The results therefore give no clear indication of whether there is a better way to connect 16 nodes than using the single-ring. The throughput and latency was better in the 4-ring interconnect than in the single ring, but only until the 4-ring interconnect got saturated. From then on the throughput decreased and latency increased, i.e. the 4-ring interconnect was not stable during transient overload. Still, the results from the 4-ring simulations give reason to believe that there is a better way to connect 16 nodes than using a single ring, and perhaps more switches could increase the performance of the 4-ring interconnect.

Issue 9: Does the packet preemption protocol in combination with priority output-queue and preemptive bypass-queue meet the requirements stated in the formal definition of the SCI/RT project (refer to 2.3)?

A ring with 4 nodes has been simulated and results were presented in chapter 8. The nodes had priority-preemptive output and bypass-queue. In section 3.2.3 it was proposed certain criteria with which the SCI/RT results should be evaluated, because the criteria in the SCI/RT draft [IEEE, 1992b] was considered too elusive.

In the SCI/RT results, we observed that the modifications met only a part of the criteria. The throughput was higher for higher priorities than for lower priorities, and the latency was less for higher priorities than for lower priorities, but when load was very high it was reasonable to expect that the highest priorities consumed all bandwidth. Instead the the throughput for the 3 highest priorities (out of 4 priority-levels) approached a stable non-zero value.

To conclude on Issue 9: The modification consisting of priority-preemptive output and bypass-queue does not meet the entire set of the criteria proposed in 3.2.3, but the results are still promising. The modification ensures that higher priorities get more of the bandwidth than the lower priorities and that the latency is lower for the higher priorities than for lower priorities.

9.2.2 Other results

The 4-ring interconnect is not stable under transient overload. Simulations indicate that the switches are causing the situation, because they are unable to transmit more when load increase. Instead the switches transmit less. The switches are bottlenecks which become smaller when load increase and nodes in each ring transmit more.

Because a switch is transmitting packets on behalf of several nodes, it will behave almost as hot sender, and as we observed in section 6.3, a hot sender will transmit less when load (in the remaining nodes) increase.

Incorporating SCI/RT into the SCI-simulator was one of the biggest challenges in the work related to the thesis, not only because of the implementation problem encountered, but also because it required an overview and understanding of the various modifications proposed, and above all to decide on which of the proposals to investigate. Different people

had different ideas on how to modify the SCI-protocol (they still do because SCI/RT is still an open issue), some proposed modifications in relation to Rate Monotonic Scheduling (RMS) and other propose a token-based scheme. These were, and still are, the two main approaches to SCI/RT. A difference between the two approaches seems to be that in the RMS-approach packets are preempted and bandwidth inevitably wasted, while in the token based scheme the idea is to negotiate for bandwidth before packets are transmitted.

It is not possible to recommend one approach over the other based on the results in this thesis, only modifications related to RMS have been investigated, but the results are still promising on behalf of the RMS-approach. Furthermore, the process of understanding SCI/RT and deciding which of the proposals to investigate, indicates that modifying the SCI-protocol for real-time purposes is difficult, even when the modifications are based on mathematical theories, like RMS.

9.2.3 Further work

In this thesis uniform and non-uniform load and traffic patterns have been simulated. Worst case and best case have not been simulated but would be interesting to determine additional properties of the SCI-interconnect.

It would also be interesting to investigate different configurations of nodes whose load and traffic pattern are given. If there are groups of nodes which communicate almost only within the group, these groups should perhaps be assigned a ring each. In this way nodes are partitioned into smaller groups of nodes, and each group be assigned one ring each. Switches could be used between the rings to enable the occasional communication between groups.

Interconnects which are larger than those considered in this thesis, may display properties not visible in smaller rings, so simulating large interconnects consisting of hundreds of nodes would be interesting. If the cache coherence layer was implemented on top of the existing simulator, real programs could be used as input to the simulator.

Related to SCI/RT simulations, several issues arose after the results had been investigated. The size of the bypass-queue is expected to affect the performance of the ring and large bypass-queues may lead to a higher latency, but less preemption. Also the various strategies regarding the priority of echo-packets have not been investigated, except that one of these strategies is used during simulation (an echo-packet inherits the priority of the corresponding send-packet). Several strategies exist and echo-packets can be assigned highest possible priority, lowest possible priority, inherit the priority of the corresponding send-packet or inherit the priority of the blocked send-packet in the output-queue in the case of an echo-packet generated in a bypass-queue by a preemption.

The size of the output-queue and the number of outstanding packets is also expected to affect the performance of the SCI/RT ring, and the size of the output-queue and the bypass-queue should perhaps relate in some way. If the bypass-queue is too big compared to the output-queue, a lot of echo-packets could reside in the bypass-queue and could be passed by higher priority send-packets, instead of getting back to the node and free space there. In this way outstanding low priority send-packets may hold up valuable space in the output-queue and eventually prevent higher priority packets from entering the output-queue, and as a result reduce the throughput for higher priorities.

The preemption priority output-queue and bypass-queue assumed in the SCI/RT simulations should also be regarded as theoretical devices, because the time to perform a preemption is not considered. An unlimited number of send-packets can be preempted

within a clock-cycle, which is not possible in a hardware implementation. It would be very interesting to investigate how the preemption-latency affected the various priority-levels.

9.3 Summary

This chapter has presented and discussed the conclusion of the thesis. It has concluded on the issues presented in chapter 3, Issue 1 - Issue 9, and has presented additional results and discussed further work, related to the design process of the SCIsim-simulator and the performance of SCI and SCI/RT. Consequently, this chapter summarizes the main results of the thesis and rather than summarize the summary here, the reader should refer to section 9.1 and 9.2.

[This page has been intentionally left blank]

Appendix A

Proposals on underlying models of the transmitter-stage

This appendix contains three proposals on how to describe the physical model of the transmitter-stage in a node-interface. These proposals have been used when the SCIsim simulator was designed.

In a node interface (Figure 2.6) the transmitter-stage must decide, at each instant of time (each clock cycle), whether to transmit from the output-queue or the bypass-queue. The transmitter-stage in an SCI-ring with flow control must behave differently from a transmitter-stage in an SCI-ring without flow control (if our intention was to investigate the difference in performance). Each of the following three cases, which have been investigated in chapter 6, 7 and 8, are handled by a different type of transmitter-stage. In figure A.1-A.3 the transmitter-stages are shown as finite state machines:

SCI flow control: In an SCI-ring with flow control, the ring bandwidth allocation protocol (refer to section 2.2.3) are used.

When all nodes in an SCI-ring implements the finite state machine in figure A.1, the SCI-ring bandwidth allocation protocol is obeyed and fairness ensured.

No flow control: In a ring without flow control a node will transmit from the output-queue only when the bypass-queue is empty.

When all nodes in an SCI-ring implements the finite state machine in figure A.2, the SCI-ring will not employ any form of flow control.

Packet preemption protocol: In an SCI/RT-ring where the output-queues and priority bypass-queues are preemptive priority queues, the nodes must choose the queue with the highest priority. However this is not always possible - if the output-queue has a higher priority than the bypass-queue and the bypass-queue is full, the node has to transmit from the bypass-queue or preempt the bypass-queue. Again there are various alternatives on how and when to preempt, and in this thesis the packet preemption protocol, described in section 2.3.3, is considered.

When all nodes in an SCI/RT-ring implements the finite state machine in figure A.3, the packet preemption protocol is obeyed.

As mentioned above, the transmitter-stages are shown as finite state-machines, and each transmitter-stage behave according to this state-machine. The states are shown as

ABBREVIATION	MEANING
MUX	The multiplexer
RecStage	The receiver-stage
RecStage.lg	The value in the lg-bitfield (Refer to 2.2.3) of the last symbol from the receiver-stage
OQ.GetSymbol	Get the next symbol in the first packet in the output-queue
BQ.GetSymbol	Get the next symbol in the bypass-queue
BQ.PutSymbol(X)	Put symbol X into the bypass-queue
BQ.Preempt(X)	Preempt packets in the bypass-queue until X bytes of free space is created
SaveIdle	The saveidle-buffer
New NoGo-idle	Create a new No-Go idle-symbol

Table A.1: Abbreviations used in the statements

ABBREVIATION	MEANING
MUX == PacketizedSymbol	The last symbol passing the multiplexer was a packet symbol
MUX == NoGo-idle	The last symbol passing the multiplexer was a No-Go idle-symbol
MUX == Go-idle	The last symbol passing the multiplexer was a Go idle-symbol
EOP	The last symbol passing the multiplexer was the last symbol in a packet
RecStage == PacketizedSymbol	The last symbol from the receiver-stage was a packet symbol
RecStage == Idle	The last symbol from the receiver-stage was an idle-symbol
BQ	There exist at least one packet in the bypass-queue
BQ.SuffFreeSpace(X)	There exist at least X number of free bytes in the bypass-queue
BQ.SuffDelSpace(X)	There can be created (by preempting packets) at least X number of free bytes in the bypass-queue
OQ	There exist at least one packet in the output-queue
OQ.Pri >= BQ.Pri	There exist a packet in the output-queue with a higher or equal priority than the highest priority in the bypass-queue
OQ.first.size+2	Size of the highest priority packet in the output-queue plus size of the CRC-symbol (2 bytes)

Table A.2: Abbreviations used in the predicates

ovals (eg. the **IDLE** state in figure A.1) and a state-transition is shown as a an arrow from one state to another (eg. from **IDLE** to **BLOCKED**). A transition between two states takes place when the predicate associated with the transition (shown in bold-face, e.g. **not OQ**) is true, and during the transition the statements (shown in parenthesis, e.g. `[ MUX := RecStage ]`) are executed. To simplify the figures, the two output-queues are treated as one queue.

The statements are written in pseudo-code and contain some abbreviations which are defined in table A.1. The abbreviation used in the predicates are defined in table A.2.

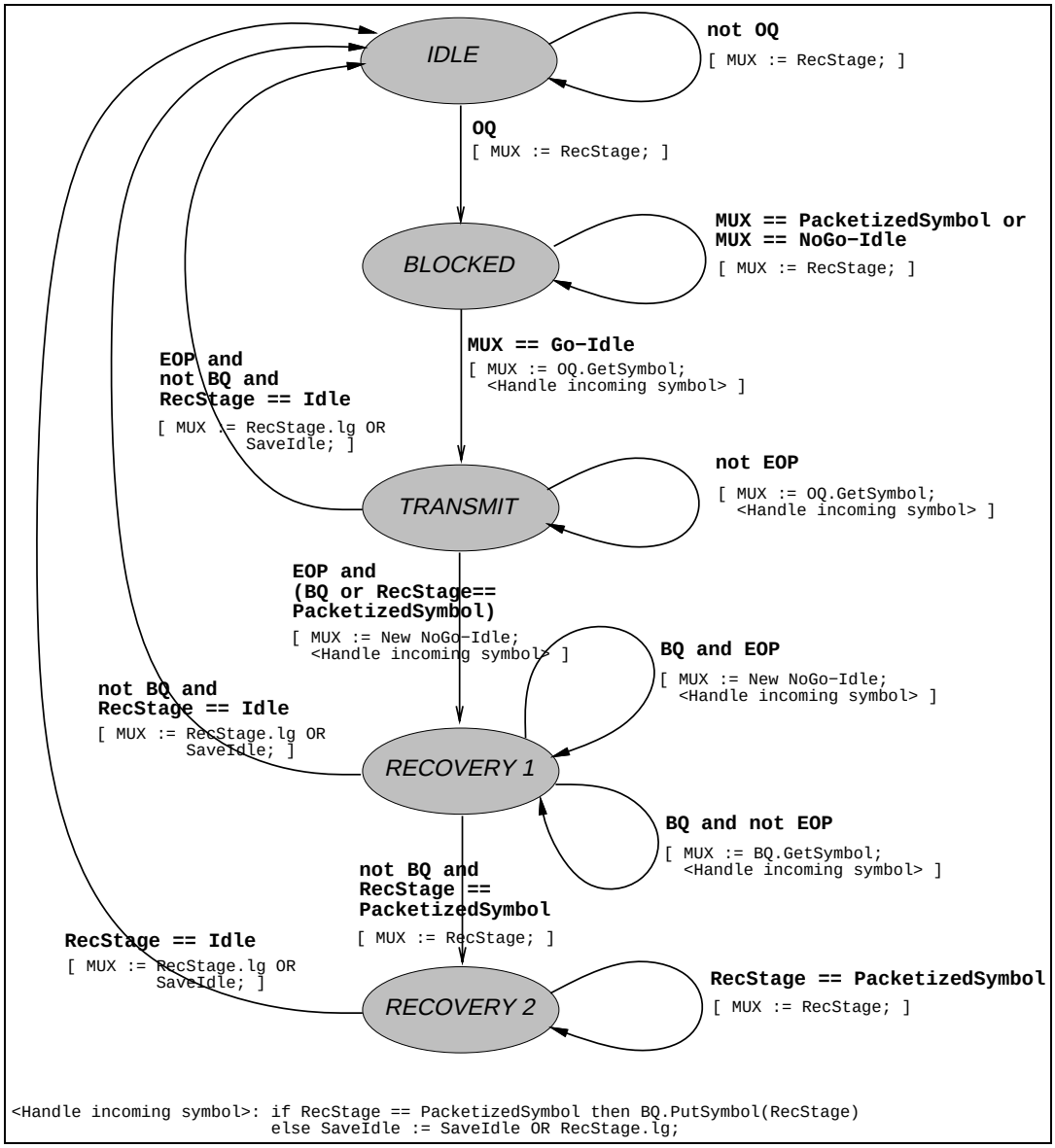


Figure A.1: A transmitter-stage according to SCI flow control

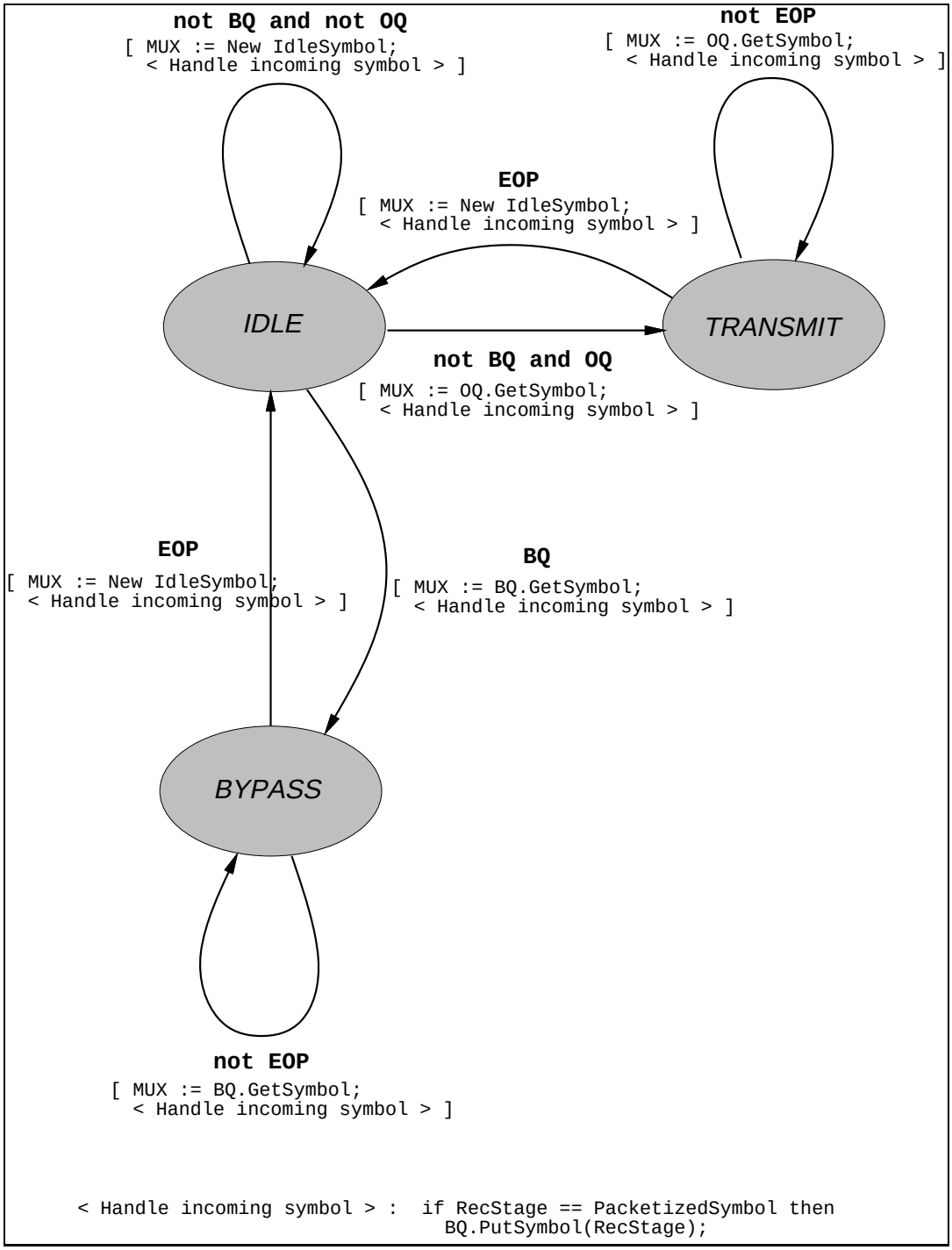


Figure A.2: A transmitter-stage according to no flow control

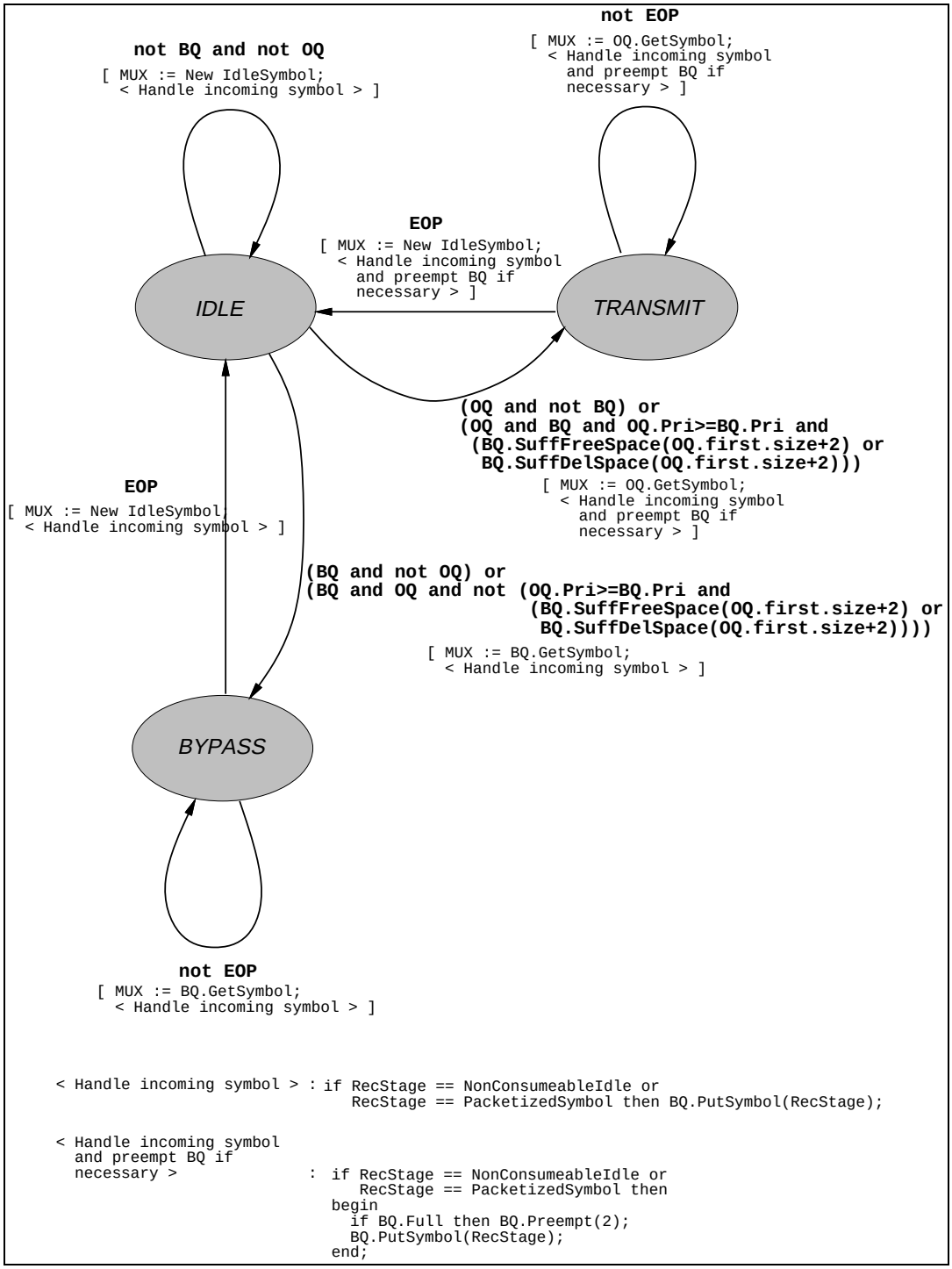


Figure A.3: A transmitter-stage according to packet preemption protocol

Bibliography

[Bhattacharyya and Johnson, 1977] Gouri K.Bhattacharyya, Richard A.Johnson

«Statistical Concepts and Methods»

New York: John Wiley & sons, Inc., 1977

[Birtwistle et.al., 1982] Graham M.Birtwistle, Ole-Johan Dahl, Bjørn Myhrhaug, Kristen Nygaard.

«SIMULA BEGIN»

Lund, Sweden: Studentlitteratur. Goch, Germany: Bratt Institut für Neues Lernen. Bromley, UK: Chartwell-Bratt Ltd., 1982

[Bogaerts and Wu, 1995] Andre Bogaerts, Bin Wu.

«The SCILab Cookbook»

CERN 1211, Geneva-23, Switzerland, July 1995

Available via anonymous ftp from: sunsci.cern.ch - simulation/DOC/SCILab.ps

[Bothner and Hulaas, 1991] John Weding Bothner, Trond Ivar Hulaas.

«Various interconnects for SCI-based systems»

University of Oslo, Department of Informatics, P.O.Box 1080 Blindern, N-0316 Oslo 3, Norway, 1991

Available via anonymous ftp from: ifi.uio.no - pub/sci/Topologies_Paris.PS

[Bothner and Hulaas, 1993] John Weding Bothner, Trond Ivar Hulaas.

«Topologies for SCI-based systems with up to a few hundred nodes», Cand. Scient. thesis.

University of Oslo, Department of Informatics, P.O.Box 1080 Blindern, N-0316 Oslo 3, Norway, 1993

Available via anonymous ftp from: ifi.uio.no - pub/sci/Topologies_Thesis.PS

[Censier and Feautrier, 1978] Lucien M.Censier, Paul Feautrier.

«A New Solution to Coherence Problems in Multicache Systems»

IEEE Transactions on Computers, Vol.27, No.12, December 1978, p.1112-1118

[Chaiken et.al., 1990] David Chaiken, Craig Fields, Kiyoshi Kurihara, Anant Agarwal.

«Directory-Based Cache Coherence in Large-Scale Multiprocessors»

IEEE Computer, June 1990, p.49-58

[Dahl et.al., 1982] Ole-Johan Dahl, Bjørn Myhrhaug, Kristen Nygaard.

«SIMULA 67 Common Base Language»

Report no. 725 (S 22), Norwegian Computing Center, revised November 1982.

[Flynn, 1972] Michael J.Flynn.

«Some Computer Organizations and Their Effectiveness»

IEEE Transactions on Computers, Vol.C-21, Iss.9, September 1972, p.948-960

[Gjessing et.al., 1990a] Stein Gjessing, Stein Krogdahl, Ellen Munthe-Kaas.

«A top Down Approach to the Formal Specification of SCI Cache Coherence», Technical Report 146.

Department of Informatics, University of Oslo, P.O.Box 1080 Blindern, N-0316 Oslo 3, Norway, August 1990

Available via anonymous ftp from: ifi.uio.no - pub/sci/tech-rep-146.PS

[Gjessing et.al., 1990b] Stein Gjessing, Stein Krogdahl, Ellen Munthe-Kaas.

«Formal Specification and Verification of SCI Cache Coherence», Technical Report 142.

Department of Informatics, University of Oslo, P.O.Box 1080 Blindern, N-0316 Oslo 3, Norway, August 1990

Available via anonymous ftp from: ifi.uio.no - pub/sci/tech-rep-142.PS

[Gjessing and Munthe-Kaas, 1991] Stein Gjessing, Ellen Munthe-Kaas.

«Formal Specification of Cache Coherence in a Shared Memory Multiprocessor», Technical Report 158.

Department of Informatics, University of Oslo, P.O.Box 1080 Blindern, N-0316 Oslo 3, Norway, 1991

Available via anonymous ftp from: ifi.uio.no - pub/sci/tech-rep-158.PS

[Goodman, 1983] James R.Goodman.

«Using cache memory to reduce processor-memory traffic»

Proceedings Tenth Annual Symposium on Computer Architecture, Stockholm, Sweden, (ACM), 1983, p.124-131

[Gustavson and Li, 1995] David B.Gustavson, Qiang Li.

«Local-Area MultiProcessor: the Scalable Coherent Interface»

SCIzzL, Santa Clara University, Department of Computer Engineering, Santa Clara, California 95053, 1995

[Hennessy and Patterson, 1990] John L.Hennessy, David A.Patterson.

«Computer Architecture, A Quantitative Approach»

San Mateo, California: Morgan Kaufmann Publishers, Inc., 1990

[Hexsel and Topham, 1994] Roberto A.Hexsel, Nigel P.Topham

«The Performance of SCI Memory Hierarchies»

Technical Report CSR-30-94, Department of Computer Science, Edinburgh University, February 1994

[IEEE, 1992a] Institute of Electrical and Electronics Engineers.

«SCI - Scalable Coherent Interface», IEEE Std. 1596-1992.

Institute of Electrical and Electronics Engineers, Inc., 345 East 47th Street, New York, NY 10017, USA

[IEEE, 1992b] Institute of Electrical and Electronics Engineers.

«SCI/RT - Scalable Coherent Interface For Real Time Applications», draft D0.13 for IEEE p1596.6.

Institute of Electrical and Electronics Engineers, Inc., 345 East 47th Street, New York, NY 10017, USA

[Jain, 1991] Raj Jain.

«The art of computer systems performance analysis: techniques for experimental design, measurements, simulation, and modeling»

New York: John Wiley & sons, Inc., 1991

[Kernighan and Ritchie, 1988] Brian W.Kernighan, Dennis M.Ritchie.

«The C Programming Language»

2nd ed., Englewood Cliffs, New Jersey: Prentice Hall, 1988

[Kirkerud, 1989] Bjørn Kirkerud.

«Object-oriented programming with SIMULA»

Wokingham, England: Addison-Wesley Publishing Company, 1989

[Kogge, 1981] Peter M.Kogge.

«The Architecture of Pipelined Computers»

New York NY: McGraw-Hill Book Company, 1981

[Lin and Tarng, 1991] Tein-Hsiang Lin, W.Tarng.

«Scheduling periodic and aperiodic tasks in hard real-time computing systems»

Performance Evaluation Review, Vol.19, Iss.1, May 1991, p.31-38

[Liu and Layland, 1973] C.L.Liu, James W.Layland.

«Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environement»

Journal of the ACM, Vol.20, No.1, January 1973, p.46-61

[Picker et.al., 1994] Dan Picker, Ronald D.Fellman, Paul M.Chau.

«An Extension to the SCI Flow Control Protocol for Increased Network Efficiency»

Department of Electrical and Computer Engineering, University of California, San Diego, La Jolla, CA.92093-0407

[Picker and Fellman, 1994] Dan Picker, Ronald D.Fellman.

«An SCI Simulator with Traffic Flow Animation

Department of Electrical and Computer Engineering, University of California San Diego, La Jolla, CA.92093-0407

Available via anonymous ftp from: arad.ucsd.edu - pub/Papers/SCISimulator.ps.Z

[Roth, 94] Luchi Roth.

A discussion of Proposed SCI Enhancements for Military Appls

AMPAC, Inc., Warminster, PA 18974, USA.

Email: roth@NADC.NADC.NAVY.MIL

Distributed January 16. 1994 on mailing list sci_announce@hplsci.hpl.hp.com.

[Scott, 1995] Tim Scott.

«Simple train protocol for sci/rt»

NAVSURFWARCENDIV, Code 6041, 300 Highway 361 Crane, IN 47522-5001, USA.

Email: tscott@avoca.nwscc.sea06.navy.mil

Distributed March 6. 1995 on mailing list sci_rt@sunrise.sci.edu.

[Scott et.al., 1992] Steven L.Scott, James R.Goodman, Mary K.Vernon.

«Performance of the SCI Ring»

19th Annual International Symposium on Computer Architecture

Computer Architecture News, Vol.20, Iss.2, May 1992, p.403-414

[Stroustrup, 1991] Bjarne Stroustrup.

«The C++ Programming Language»

2nd ed., Reading MA:Addision-Wesley, 1991

[Tang, 1976] C.K.Tang.

«Cache system design in the tightly coupled multiprocessor system»

AFIPS Conference Proceedings, Vol.45, National Computer Conference, 1976, p.749-753.

[Wegner, 1990] Peter Wegner.

«Object-oriented programming»

Encyclopedia of Computer Science

3rd ed., London: Chapman & Hall, 1990, p.959-962

[Wilkes, 1965] M.V.Wilkes.

«Slave Memories and Dynamic Storage Allocation»

IEEE transactions on electronic computers, Vol.14, Iss.2, April 1965, p.270-271

[Zalewski, 1993] Janusz Zalewski.

« What every engineer needs to know on rate monotonic scheduling. A tutorial»

Department of Computer Science, Southwest Texas State University, San Marcos, TX 78666-4616.

Email: jz01@academia.swt.edu

Note: The above internet-addresses are correct at the time of writing, but because internet-resources tends to be unstable, their correctness in the future cannot be guaranteed. The internet-addresses should therefore be regarded as a supplement to the formal reference.