UNIVERSITY OF OSLO
Department of informatics

Fault-tolerant routing in SCI networks

Master thesis
60 credits

Håkon Kvale Stensland

August 1st 2006

Table of contents

Figure list

Abbreviations

Abbreviation	Interpretation
ACK	Acknowledgement
API	Application Program Interface
ASI	Advanced Switching Interconnect
ATT	Address Translation Table
B-link	Backside-link
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
DDR	Double Data Rate
DMA	Direct Memory Access
DSM	Distributed Shared Memory
ECC	Error Correction Code
FDDI	Fiber Distributed Data Interconnect
FIFO	First Inn First Out
GENIF	Generic Interface
GIO	Global Input / Output
GPL	General Public License
HA	High Availability
HPC	High Performance Computing
I/O	Input / Output
IB	InfiniBand
IEEE	Institute of Electrical and Electronics Engineers
IGP	Interior Gateway Protocol
IOCTL	I/O Control
IRM	Interconnect Resource Manager
KO	Key Offset
LC	Link Controller
LVDS	Low Voltage Differential Signaling
MAN	Metro Area Network
MMU	Memory Management Unit
MPI	Message Passing Interface
NUMA	Non-Uniform Memory Access
OSPF	Open Shortest Path First
PAL	Physical Abstraction Layer
PCB	Printed Circuit Board
PCI	Peripheral Component Interconnect
PCIe	Peripheral Component Interconnect Express
PCI-X	Peripheral Component Interconnect Extended
PIO	Programmed Input / Output
PSB	PCI to SCI bridge
QoS	Quality of Service

RAID	Redundant Array of Independent Disks
RMA	Remote Memory Access
RPR	Resilient Packet Ring
RTG	ReadyToGo Timer
SAN	System Area Network
SCI	Scalable Coherent Interface
SISCI	Software Infrastructure for SCI
SMP	Symmetric Multiprocessors
TCP/IP	Transmission Control Protocol/Internet Protocol
UID	Unique ID

Preface

This thesis was written during the period January 2005 to July 31st 2006. My supervisors have been Olav Lysne and Tor Skeie at Simula Research Laboratory and Hugo Kohmann at Dolphin Interconnect Solutions.

First of all, I would like to thank my supervisor Olav Lysne for his excellent guidance, various thoughts and good comments during my thesis work. I also want to thank my other supervisors, Tor Skeie and Hugo Kohmann for valuable feedback during the thesis.

The work on this master thesis has been done at Dolphin Interconnect Solutions at Skullerud in Oslo, and at Simula Research Laboratory at Fornebu. Thanks to the students and employees both at Simula and Dolphin for such a nice workplace environment. I would also like to show my gratitude to the students at 'Fagutvalget ved Informatikk' at the Department of informatics for a good social environment during my studies.

I also want to say thank you to my family, friends and especially to my girlfriend Marianne, for supporting me.

Finally a special thanks to Roy Nordstrøm at Dolphin Interconnect Solutions for help with Dolphin's SCI driver and for valuable comments and feedback during the work with this thesis.

Oslo, July 31st 2006
Håkon Kvale Stensland

Summary

Fault-tolerant routing has been a hot topic in the academic community for quite some time now, and several different approaches have been suggested. In the interconnect industry however, fault-tolerant routing has not been implemented to the same extent. In this thesis we have adapted and implemented a local fault-tolerant routing approach in SCI interconnect technology produced by Dolphin Interconnect Solutions. The existing technology used in SCI is based in a static reconfiguration approach, where the traffic is disabled, while the new routing is calculated by a central front-end and distributed out to the nodes.

Our algorithm builds upon the principle of enabling the nodes to make routing decisions from the information that is available to them locally, and having the rest of the nodes in the cluster to be prepared for this unexpected traffic. The algorithm has been tested on real hardware, and we have shown that it can handle several levels of traffic in the network. The test has also proven that our method gives the same performance both before and after the error occurs if the packets have the same conditions, such as competing traffic and link length. Our routing algorithm is currently integrated as a part of Dolphin Interconnect Solutions driver in the last official release.

1 Introduction

1.1 Background

For interconnects, fault-tolerant routing have always been a hot topic in the academic community. A fault-tolerant routing is an important requirement for several applications, for example real-time systems like databases and telecommunication system. These systems have very high demands on communication reliability, and they can often not tolerate long periods of communication downtime. Redundant hardware is often the solution for applications that need high availability, thus increasing the cost and complexity of the network.

SCI is a interconnect technology standardized in 1992. SCI offers a distributed shared memory between computers, and the most common topologies today is a 2D and 3D torus topology. The first and only commercial implementation was done in 1993 by the Norwegian company Dolphin Interconnect Solutions, and they have continued to evolve both the hardware and the software. The current version of Dolphin's SCI implementation uses a static reconfiguration approach controlled by a central front-end computer over Ethernet for handling problems like bad cables between the nodes, faulty hardware and software lockups in the network. The reconfiguration process is also a slow process, and it needs to halt the network for the entire reconfiguration time.

A fault-tolerant routing algorithm that handles error locally had never been implemented on SCI hardware before.

1.2 Problem statement

In this thesis, we are going to try to implement a fault-tolerant routing algorithm for SCI networks. The implementation will be done on the current generation of Dolphin Interconnect SCI cards, and since we use existing hardware, any problems we might run

into must be solved in software. The integration into Dolphin's SCI driver will be done in cooperation with software engineers at Dolphin Interconnect Solutions.

Our approach calls for having the nodes to make routing decisions locally, and setting up the routing tables in all the nodes speculative.

The second goal is to develop a tool to measure for how long time the communication is down between the nodes. The fault-tolerant routing algorithm will be tested under different scenarios on real hardware. We want to see how much the bandwidth and latency is affected when a fault is introduced in the network, and how much they are affected when the load in the network is increased.

1.3 Structure

This thesis is divided into 7 chapters. Chapter 2 contains an introduction to the SCI standard and important concepts in the SCI standard. The chapter continues with an overview of Dolphin's implementation of SCI. We examine how the SCI hardware and software is built up, and how default routing is done. This chapter concludes with a quick look on other competing interconnect technologies. Chapter 3 takes a look at different aspects of existent fault-tolerance mechanisms in Dolphin's SCI hardware, before we take a quick look at our approach. Some related work is presented in chapter 4. Chapter 5 describes the implementation, and different aspects of the driver that needed modification to support our new routing algorithm, and the chapter also takes a look on the test application developed to measure communication downtime. Chapter 6 is the evaluation. In this part we present the results from the different benchmarks we used to test our solution. We also have a discussion, where we compare our new solution to other approaches to fault-tolerance. In chapter 7 we conclude the thesis, and take a look at some suggestions for further work.

2 Interconnects

An interconnect is a fast network built especially to connect computers in a multiprocessing area network for applications that require high bandwidth, low latency and reliable service.

In this part we will take a look at the SCI technology, with a brief historical look, some concepts in the technology and look at an application. Next we will take a look on Dolphin's SCI hardware, and how routing is done. The chapter will be wrapped up with a quick look at other interconnect technologies.

2.1 SCI (Scalable Coherent Interface)

SCI [1] has its origin back in the late 1980s. The initial effort was to define a high performance computer bus, called "Superbus" that was supposed to support a significant degree of multiprocessing. It was soon discovered that this technology was not able to meet these requirements, since a bus is a centralized resource. Faster processors would also worsen the serial bottleneck and bus signaling was already reaching the theoretical limits (speed of light). Consequently the bus-oriented approach was abandoned. Focus was put to develop a distributed solution that would overcome shared-resources and signaling problems, while retaining the goal of defining an interconnect with services known from central busses.

The specification was finally approved in 1992, and it described both hardware and protocols that provide that the processors with a shared view of the memory. SCI also describes how to read, write, lock memory, and transmit messages and interrupts. Hardware protocols also described how to keep processor caches coherent. In the SCI interconnect the memory system and the associated protocols are fully distributed and scalable. The SCI network was going to be based upon point-to-point links, and implement a fully hardware based distributed shared memory.

SCI was designed with several goals in mind. The primary goal, as of any interconnect network, is high performance by having a high sustained throughput, low latency and a low CPU overhead during communication. The second goal is scalability in many respects. Some of them are: Scalability of aggregate bandwidth as the numbers of nodes increase, scalability of interconnect distance, scalability of the memory subsystem, and in particular cache coherence. Another important goal is to have no immediate limits for addressing nodes.

Coherent memory system was also a goal in SCI. In modern CPUs, efficient usage of cache is important. To support large SMP[1] systems with NUMA[2] characteristics, where remote memory access is supposed to be as inexpensive as local memory access, the caches must be kept coherent in hardware.

SCI also had the goal to give good economical scalability, by using the same components in high-end and low-end systems. To achieve this, the SCI description was to have a standard interface, which allows multiple devices from multiple vendors to be attached and cooperate in the same network. SCI was going to serve as an open distributed bus between CPU, memory, I/O controllers and other devices or bridges.

2.1.1 Concepts in the SCI standard

Many of the goals that were set when work on SCI started has been accomplished and some have not. We will now look at some important concepts, and I will focus on concepts important for SCI as an interconnect between computers.

Point-to-Point link: SCI networks are built from unidirectional point-to-point links between the nodes. This means that SCI does not have the one-at-a-time problem limitation of buses. Aggregate bandwidth in an SCI cluster will also increase linearly as more nodes are added. The protocol has the possibility to be implemented both with parallel links for short distances and serial links for longer distances. The unidirectional

[1] Symmetric Multiprocessors
[2] Non-Uniform Memory Access

point-to-point design also makes signaling easy compared to shared buses. These parameters help to make SCI a scalable network.

Nodes and topology: By design, SCI is able to connect a large number of nodes. The specification suggests up to 64k nodes. A node can be a complete computer, processor, memory modules, I/O controllers, or a bridge to another medium. In all implementations of SCI so far, nodes are complete machines. The basic topology in SCI is a ring topology with unidirectional point-to-point links. This ring topology is often referred to as a ringlet. A switch based topology is also available, and it can include computers connected directly to the switch, rings connected to the switch, or a combination of these. (Figure 1) Another possibility is rings connected in multiple dimensions, also called a 2D or 3D torus topology (Figure 1). The multi-dimension torus topologies are considered the most efficient design in large systems with over 16 nodes, both because of a lower hardware cost, and an increased number for communication links, giving more aggregate bandwidth and more redundant paths in the network.

Transactions and packets: All transactions in SCI are split. They consist of a request, and a response, and the nodes communicating are often referred to as a requester and a responder. Each packet that is sent by a requester and recived succesfully by the responder generates an echo packet (acknowledgement) that is returned by the reciver. This echo tells the requester if the packet is accepted by the reciver, or rejected. If the packet is accepted the requester will remove it from the output queue, and if the packet is discarded a retransmission is initalized. This retry mechanim prevents SCI from being able to guarantee in-order delivery.

The packet types in SCI is corresponding with the transaction phases. We have four basic types of SCI packets: request send, request echo, response send and response echo. An SCI send packet is a contiguous sequence of 16-bit symbols. A header is typically 14 bytes, and the tail, which contain the checksum, is 2 bytes. A typical SCI header can be seen in (Figure 10). In the header we have fields for both target and source address, flow

control, and sequence number. The packet can carry 0 to 256 bytes of data. SCI echo packets are small compared with send packets. An echo packet is typically 8 bytes.

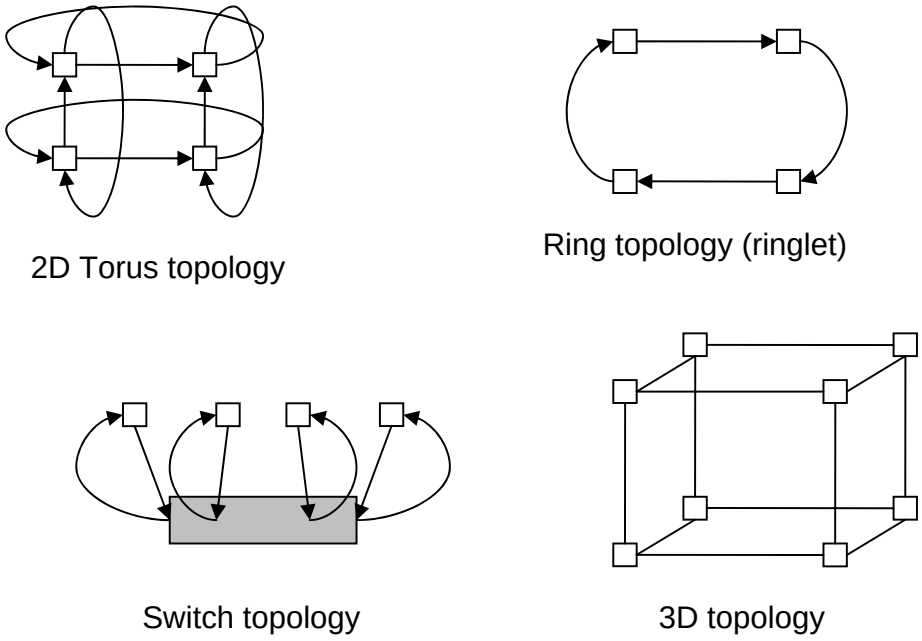

Figure 1: SCI topologies

Reliability in hardware: In order to achive high-speed transmission, SCI uses error detection in hardware. This detection is based upon a 16-bit CRC[3] mechanism in the tail of each SCI packet. If the reciving node detects a checksum-error, the packet will be dropped. The sender side will detect this, because of a missing response back on the dropped packet. The sender detects lost packets with a timer. If a response on the packet is not recived before the time-out, the sending node automaticly assumes that the packet is lost. Since these responses are sent on a per-packet basis, we can not guarantee in-order delivery in SCI. Not being able to guarantee in-order delivery is a factor that has to be conciderd when implementing software on SCI. In SCI implementations this is solved with memory barrier operations to enforce a memory access order.

[3] Cyclic Redundancy Check

Addressing and remote memmory access: SCI uses a 64-bit addressing scheme. The 16 most significant bits is used as a node ID, and the remaining 48-bit is used for adressing internaly within the node. The addressing scheme in SCI uses a global 64-bit address space, in order to have a physically addressed, distributed memory system. The way the memory is distributed is transparrant to software and processors. The distrubuted memory can also be addressed from user-level, without involving the operating system.

Layered specification: The SCI specification is structured into a three layer stack. We have a physical layer, a logical layer, and an optional cache coherence layer. Cache coherence is specified in the SCI spesification, but it is optional to implement.

In the specification, three different physical link models are defined. The first is a 16-bit parallell electrical link operating at 1 GByte/s over short distances (meters), the second is a serial electrical link operating at 1 GBit/s over intermediate distances (tens of meters) and third a serial optical link that operate at 1 GBit/s over long distances (kilometers). Today, it is only the parallel electrical interface that is being used.

2.1.2 Applications of SCI

During standardization, SCI was thought to fulfill several demands. Some of these demands were: To work as a system area network (SAN), to work like a memory interconnect for cache coherent systems, and to operate as an I/O subsystem interconnect. All implementations of SCI so far have been focused towards a high speed interconnect for system area networks. My focus in this section will therefore be SCI as a system area network for clusters.

Networks of normal workstations or PCs are used to be able to offer cost-efficient parallel processing. SCI can offer reliability and high-performance in such clusters. In this application the SCI adapter is connected to an I/O bus (i.e. PCI or PCI Express) and works on the same principle as an Ethernet adapter. The main difference is that SCI offers hardware based physical distributed shared memory (DSM). Together with SCI

driver software, SCI hardware offers the network characteristics of a NUMA parallel machine. Nodes can create a shared memory segment in the physical memory. This is then converted to SCI address space. The other nodes in the network can use this distributed address space in their own I/O space. The SCI card uses a local ATT[4] table to maintain the mapping between local I/O addresses and global SCI addresses. To maintain the mapping between virtual addresses and physical addresses, the machine uses an MMU[5], usually embedded in the memory controller on the chipset. The process of mapping memory on a local node to do a remote memory access is shown in Figure 2.

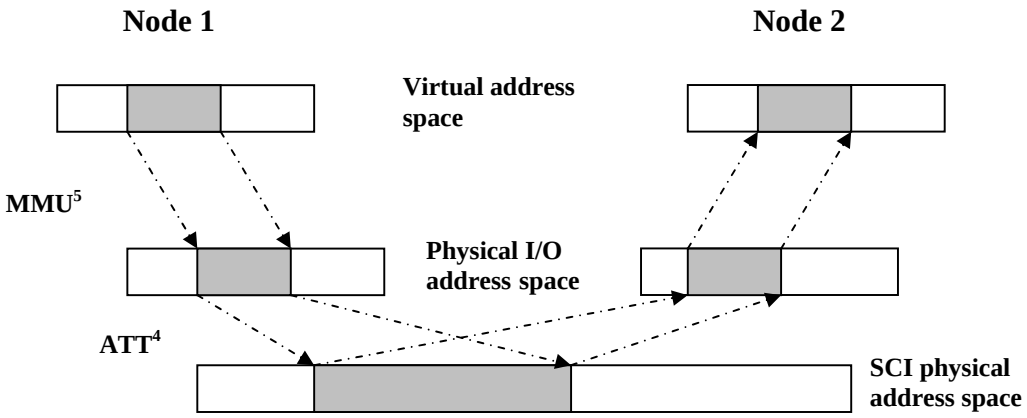

Figure 2: Address space for SCI

When the memory mapping has been set up, communication between nodes can be accomplished by processes at user level with normal CPU load and save operations into the segments mapped for remote memory. Since the SCI card translates these memory accesses into SCI transactions, intervention from the operating system and a complex protocol stack will not be needed. This helps to keep the latency down, not only for kernel-level applications, but also for user-level processes.

[4] Address Translation Tables
[5] Memory Management Unit

Two modes of data transfers are available. The first method, called PIO[6], is a mode, where the CPU actively reads data (load) from memory locations, and writes it (store) to the memory area designated for remote memory access. The advantage here is that this can be done in user-mode, and the latency is low. The disadvantage is that this method uses CPU resources. Therefore this method is only profitable with small data sizes.

The second transfer mode is called DMA[7] mode. This mode requires a dedicated hardware, called a DMA engine in the SCI controller. This controller copies data into and out of the nodes memory. The advantage with this approach is that the CPU is relieved, but the startup cost is higher, since the driver has to set up the DMA engine. This mode is often the preferred mode to use when transferring large data sizes.

2.1.3 Dolphin SCI hardware

Dolphin Interconnect Solutions is as of this date the only manufacturer of SCI hardware. The first SCI adapters became available in 1993. The first generation of SCI cards was based upon the Dolphin Link Controller LC-1. Later this evolved into the Link Controller 2, and now, the Link Controller 3. Dolphin's SCI cards have been built to connect computers in clusters, and software has been developed for this purpose. The interface between the adapter and the I/O bus on the computer has also evolved. First generation SCI products featured an SBus interface, which is an interface previously used by Sun Microsystems in SPARC-systems. Newer generations of SCI cards uses a PCI-SCI bridge. This was done to connect SCI to an open and more widely used bus. The PCI interface has also evolved from a 33 MHz, 32-bit interface to the 66 MHz, 64-bit interface used today. The last generation products also feature a PCI – PCI Express bridge, which enables the SCI cards to connect to the newest generation of I/O interfaces. New products featuring a native PCI Express design is also in development. In this section I will investigate the current generation of PCI SCI cards based upon Dolphins Link Controller 3 (LC3) (Figure 3).

[6] Programmed Input/Output
[7] Direct Memory Access

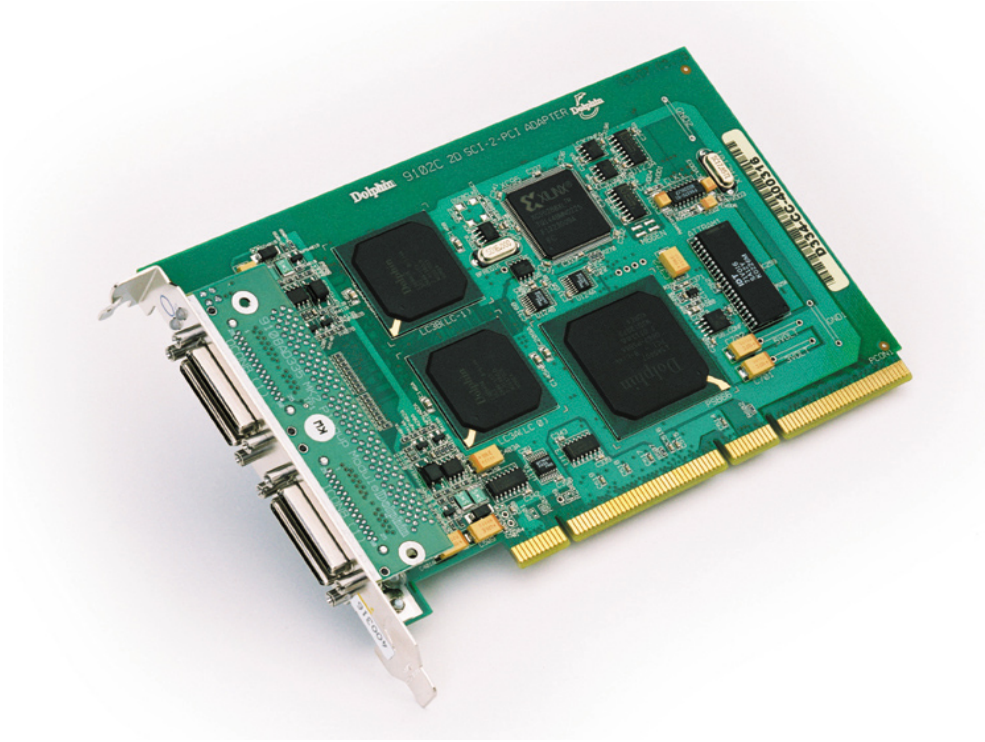

Figure 3: Dolphin D334 2D PCI Adapter

Dolphin's SCI cards are connected to the computer on the I/O bus, thus it allows the mapping of memory access between the I/O busses on many computers. This is often referred to as RMA[8]. Cache coherence is not implemented, and SCI is only used as a high bandwidth, low latency connection between computers.

The SCI cards are basically divided into two parts: a *Link Controller*[9] *(LC)* which transports packets over SCI and a *PCI to SCI Bridge*[10] *(PSB)* which has the responsibility for address mapping, DMA, interrupts, etc.

[8] Remote Memory Access
[9] LC: Link Controller
[10] PSB: PCI to SCI Bridge

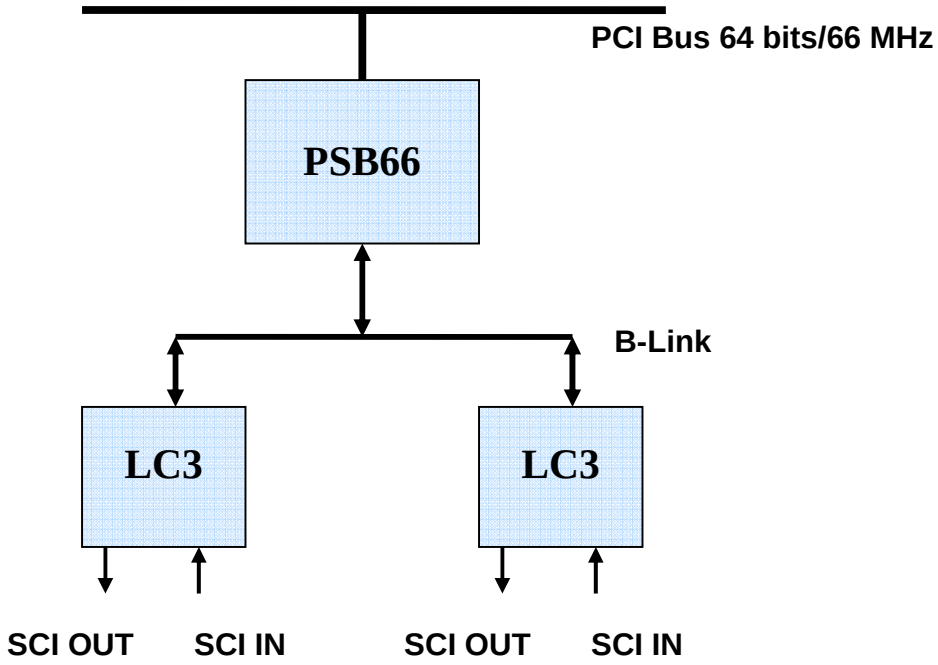

Figure 4: Architecture on Dolphin D334 2D SCI Card

The PSB and LC are as shown in Figure 4 connected with a back-end interface called a *B-link*. The B-link is a packet based split-transaction bus with simple control. SCI packets are encapsulated to utilize this bus. The B-link can have up to 8 link controllers attached. In SCI cards, the B-link is implemented as a 64-bit bus, operating at 80 MHz. This gives a theoretical bandwidth of 640 Mbytes/second.

2.1.3.1 Dolphin Interconnect, PSB66

The PSB66 chip is the bridge between PCI and B-link [2]. It is also responsible for building packets with headers and checksum, mapping memory, controlling the ATTs, set up DMA transfers and generate interrupts to the driver.

Three modes of operation are available on the SCI cards. When the card is in *slave mode* it translates PCI transactions into SCI packets. The PSB uses by default the maximum payload size of 128 bytes, while the PCI bus uses no fixed transfer size In current hostbridges, normal size is 128 bytes. Due to this fact, the performance on SCI is very sensible to parameters used on the PCI bus. In *master mode* the card transfers SCI

transactions into PCI. The last mode is *DMA mode*. In this mode the PSB is set-up to offload the CPU with doing the read and write operations to and from the memory. In DMA mode, the card will use both master and slave mode.

Addressing is an important issue when bridging from PCI to SCI. PCI has a 32-bit or 64-bit address space, and that does not match the 64-bit SCI address space. Since it is a mismatch, we need to use memory mapping. In SCI this mechanism is called an Address Translation Table (ATT). We have a total of 16k ATT entries in the PSB, and it is the driver that is responsible for the initialization of the table. The table contains the mapping of the memory, and the full SCI address, with a 16-bit node ID, and 48-bit address offset. In Dolphins implementation, only the 8 most significant bits of the node ID is actually used for addressing nodes. This gives a maximum limit of 256 nodes in a Dolphin cluster. This can be considered a serious limitation, but it has not been changed since Dolphins first implementation of SCI in 1993. The primary reason for this is that the cost of adding more routing memory to the link controllers was very high in the beginning of the 1990s. 256 nodes were considered more than enough for any cluster. The remaining 8 bits in the node ID field is used for internal addressing on the card, telling if the packet is going to the PSB, the link controller or the *hardware mailbox*.

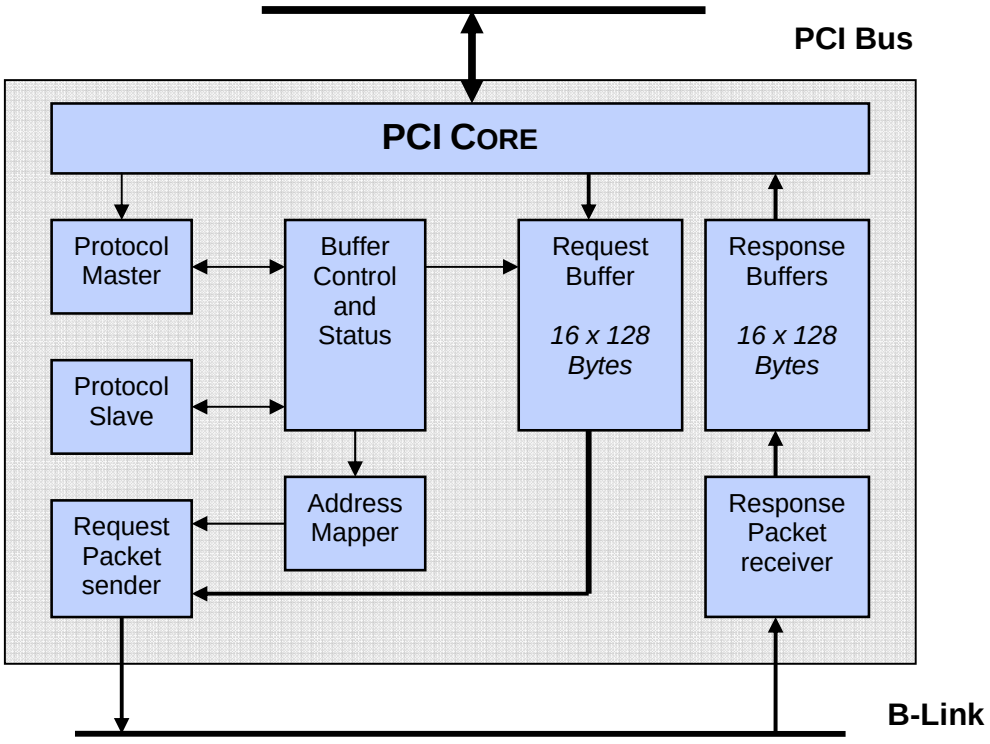

Figure 5: Block diagram of Dolphin PSB66 PCI-SCI bridge

In Figure 5, we can see a block diagram of the primary features in the last generation PSB, PSB66. We will now take a quick look at the steps that are taken when a SCI packet is sent.

1. The PSB receives data from the PCI bus. This normally happens in 64 or 128 bytes bursts. The data are encapsulated with header information. This is done in the "PCI Core" part of the chip
2. A 16-bit CRC[3]-checksum is added to the packet, by the "Buffer Control and Status" part of the chip.
3. An Access to the Address Translation Table (ATT) is done by the "Address Mapper", to generate a 64-bit SCI destination address.
4. One of the PSB buffers (streams) is selected for the SCI transfer. This stream consists of a 128-bytes data buffer. And for sending packets we use the "Request Buffer"

5. When the SCI packet leaves the PSB, a timer is started, and maintained by the "Buffer Control and Status" part of the chip. We have one timer available per stream.

If the corresponding response packet is not received within the timeout, an error interrupt is generated for the driver, and the driver reports this back to the application. If the correct response packet is received within the time-out value, the SCI protocol can guarantee that the packet has successfully been received at the destination node. The latest generation of PSB, PSB66 can handle 16 read and 16 write accesses simultaneously.

Another feature developed by Dolphin in the SCI cards is called a hardware mailbox. This is basically SCI packets with a special tag that are handled by the card in a special way. When a mailbox packet arrives, an interrupt is set by the hardware. The SCI address indicates if the packet is a mailbox packet. The advantage of the hardware mailboxes is that it enables the driver on different cards to be able to communicate with each other without accessing the remote memory buffer through the ATT tables.

2.1.3.2 Dolphin Interconnect, LC3

The link controller is the bridge between B-link and SCI, and it controls the SCI link interface. The newest generation link controller is called Link Controller 3 [3]. This is Dolphins third generation link controller. The link controller features two unidirectional SCI links, each with a theoretical capability of 667 megabytes per second. Hot-plugging of SCI cables is supported.

The new Link Controller 3 arrived in 2001, and was a major improvement over the last generation Link Controller 2. The packet size was increased to 128 bytes, from 64 bytes, and the frequency on the link was increased from 100 MHz to 166 MHz. These two improvements yielded a major increase in bandwidth. SCI also uses DDR[11] technology.

[11] Double Data Rate

This means that the signal is clocked both on the top and bottom edge of the signal. The SCI cable (Figure 6) has 18 LVDS[12] [4] signals. This increases the operating reliability, and reduces the vulnerability to noise.

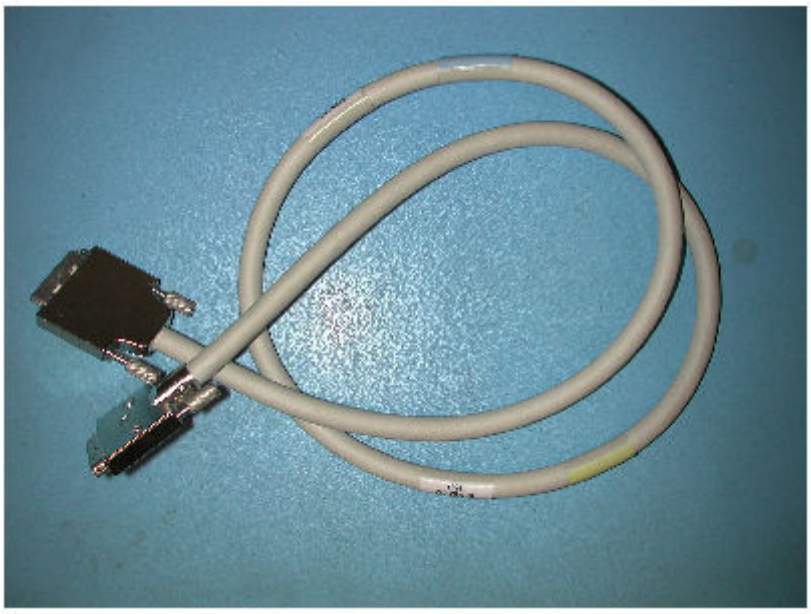

Figure 6: SCI Cable

The link controller also has the responsibility of routing. The routing tables are initialized by the driver when it is loaded. Routing in Dolphin's SCI products will be described in section 2.1.3.5. A link controller holds two routing tables. Both routing tables have 256 entries, one for each valid node ID.

The first table is called a *Link Routing Table.* The link controller looks up the packets destination address. If a bit is set in the field of the destination address, the link controller will take the packet off the SCI ring and place it in the Receive Queue (Figure 7), before sending it out on the B-link. This routing operation (SCI - B-link) takes approximately 70 nanoseconds. If the packet has no bit set in the Link Routing Table, the link controller will put it in a FIFO queue (Figure 7), and send it out on the SCI link again. This bypass operation (SCI - SCI) takes approximately 50 nanoseconds.

[12] Low Voltage Differential Signaling

The second routing table is called a *B-link table*. This table is built up in the same way as the Link Routing Table with 256 entries. If a bit is set for the destination address the link controller will take the packet out from the B-Link, and put it in the Send Queue (Figure 7), before sending it out in the SCI link.

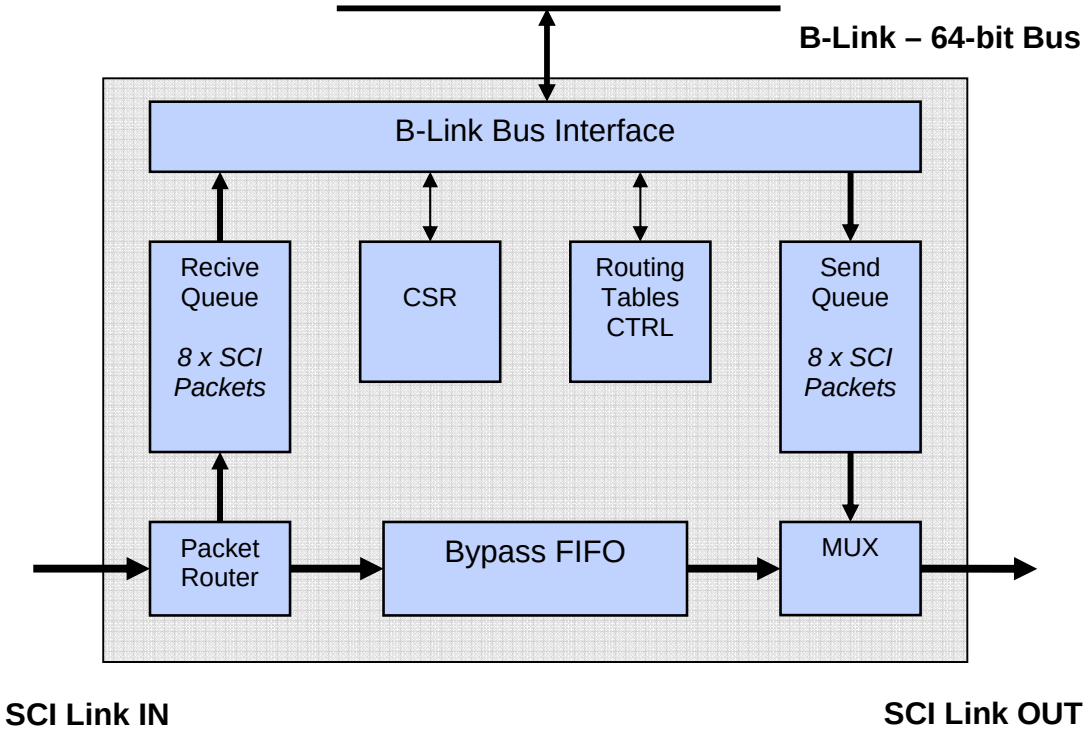

Figure 7: Block diagram of Dolphin LC3 Link Controller

The link controllers are also responsible for a basic flow control mechanism. If a remote node is busy it will send an ECHO_BUSY signal back to the local link controller. The local link controller will then back off for an extra cycle, before retransmitting. The Link controllers are also responsible for the hardware initialization on the ring. The hardware initialization is used to make sure that the cables in the ring are connected properly and elects a scrubber node on the ring. The scrubber node and flow control are described in the Fault-Tolerance chapter.

Another feature on the LC3 chips is a Bx-bar. Bx-bar is a crossbar switch used in the latest generation of Dolphin SCI switches. The crossbar enables the link controllers in a switch to have a dedicated connection to all other controllers in the switch.

The link controllers also supports all the other topologies proposed in the SCI standard [1]. The newest topology in SCI is a 3D torus topology featuring three link controllers on a single SCI card. 3D topology is used in several clusters with more than 16 nodes. The link controllers use parallel copper cables and signaling as specified in SCI sub-specification IEEE 1596.8 [5]. A parallel optical transceiver was also developed, allowing SCI to span over distances up to 150 meter, compared with the maximum 10 meters with normal copper cables. The optical solution was never produced in a volume, because of high cost, and problems with heat development. The proposed serial communication cables and communication links in the specification has not been used in any Dolphin products.

2.1.3.4 The drivers for Dolphin's SCI adapters

To allow easy access to SCI, Dolphin provides driver and programming interfaces on various levels, and for multiple operating systems. The driver is built-up as one unified component, with support for all cards and switches in the two last generations of Dolphin SCI products. The Dolphin driver is available for the following operating systems:

- Linux 2.0, 2.2, 2.4 and 2.6
- Microsoft Windows NT/2000/XP/2003
- Solaris 2.5.1, 2, 6, 7, 8, 9
- VxWorks 5.4
- Lynx 3.0.1

The driver is available for the following processor architectures:

- Intel x86 (IA32)
- x86-64 (AMD64)
- Intel IA64 (Itanium)
- Sun SPARC

- PowerPC

We are now going to take a more detailed look on how the driver is built up. The driver includes low-level components that set up the hardware and handles routing. And also higher level API's[13] like SISCI [6] and SCI SOCKET [7].

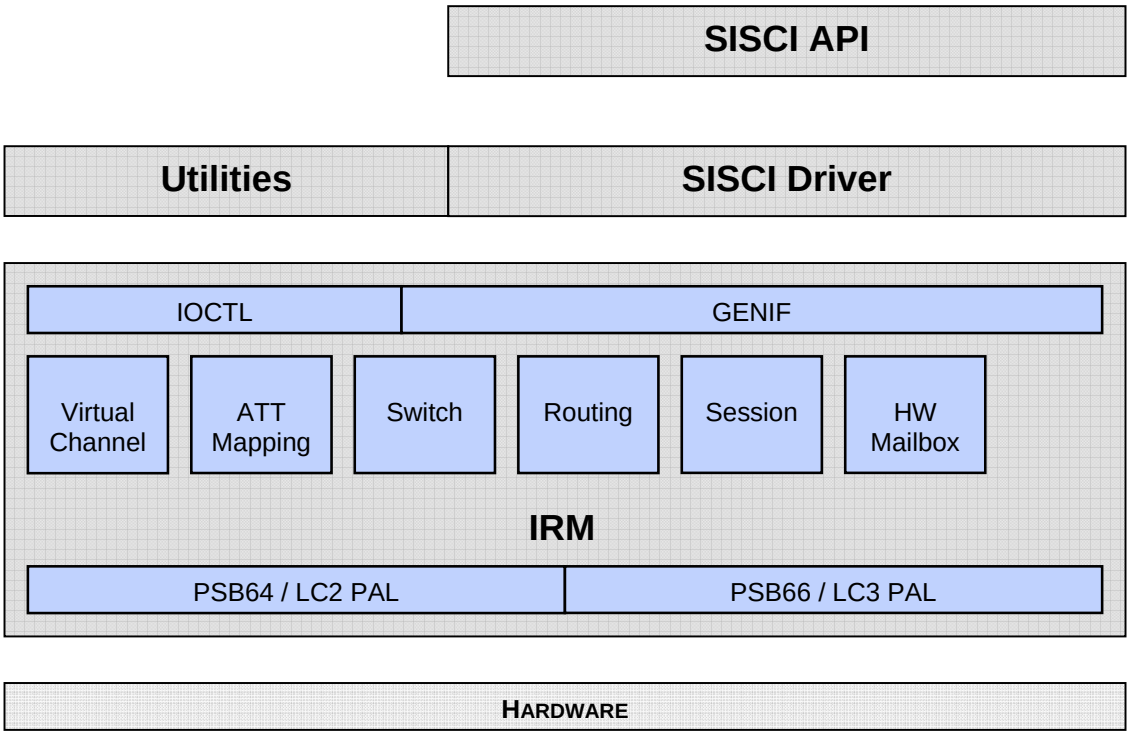

Figure 8: Dolphins driver stack

Interconnect Resource Manager (IRM): The IRM is the part of the driver that interfaces with the hardware. The low-level part of the IRM is called the physical abstraction layer (PAL-layer). In the PAL-layer we have specific code for different versions of SCI chips. In the upper part of the IRM, we find the interface for upper layer functions. This interface is called generic interface (GENIF), and the other interface is called I/O control (IOCTL). The IOCTL interface can be used by higher level applications and utilities, and among other things set a user defined routing, change topologies etc.

[13] Application Program Interface

The IRM is also built up of modules that are common for all the latest generations of Dolphin's SCI hardware. The primary role of the IRM driver is to manage hardware resources for clustering in the local node. When the system boots up and the driver is loaded, the IRM also sets up the SCI card for operation. IRM also controls and manages the ATT tables in the system. The adapters DMA engine is also managed from the IRM, and the driver sets up the DMA request from the software. The session mechanisms, including heartbeat-alive checks and endpoint to endpoint control, are also handled by the IRM. Error recovery and notification is also handled by the IRM.

On the top of the IRM, we find the GENIF. This is a kernel level programming interface, and it can do memory segment allocation, connection and mapping. DMA engine control can also be done through GENIF. It also controls remote interrupt triggering.

Above the IRM we find several utilities, some of them are *scidiag*, *sciconfig* and *scitool*. These programs use the IOCTL interface to fetch information about the adapter, and it can also configure parameters such as node ID, topology type, and other hardware parameters. In this layer we also find the Software Infrastructure for SCI (SISCI) driver. This is the driver for the SISCI API, and it interfaces directly with the GENIF part of the IRM.

The SISCI API was originally defined in the ESPRIT project [6]. The SISCI API enables easy access for user-level applications to clustering. All the tools developed for measuring data in this assignment have been developed on the SISCI API.

Dolphin has also developed a custom implementation of Berkley Sockets that enables socket applications to run over SCI hardware instead of Ethernet and TCP/IP. This socket implementation also interfaces against the GENIF layer in the IRM.

2.1.3.5 Routing in Dolphin SCI cards

Routing on Dolphin SCI cards are handled by the local node, or by a cluster manager, running on a front-end. In this section we will take a quick look at how default routing works in Dolphins SCI cards.

The routing solution on Dolphins cards today is called Graph Routing. This is a new algorithm, developed at the end of 2005. The old routing method will not be discussed in this assignment. Developers at Dolphin describe the method as an ad-hoc method, with a lot of special cases, and hard to service.

On a single ring, routing is not required in the same extent, since every node take the packet with their address in the destination field of the ring. On switches, every output port gets a range of 16 node IDs. Every port can have a single node, or a ring of maximum 16 nodes connected. The switch will forward the packets based upon which node ID range the destination address is in.

The routing for SCI's topologies of either 2D or a 3D mesh is based upon Dijkstra's shortest-path-first algorithm [8]. The routing is done in three steps, first build up structure, then run Dijkstra's shortest-path-first, and finally generate tables. We will now look at these three steps in detail for a 2D mesh.

1. *Build the structure*: In the first step, we have to tell the routing tool about our topology. The default size of the cluster on a 2D cluster is 16 nodes in X-dimension, and 16 nodes in Y-dimension. (16x16). If we have a smaller cluster, the runtime can be reduced by specifying the size i.e. 4x4. The nodes and information about them will now be placed in an array. We also have an array that stores information about which links the nodes are connected with. This array can be updated with broken links and dead nodes. The cost of the link is also stored in this list.

   The link cost is currently set to 1 for a pass-through over the FIFO queue on the link controller, and 10 for a jump to another link controller over the B-link. The cost

reflects the time these operations take on the hardware. For a pass-through the FIFO uses 50 nanoseconds, and a jump over the B-link takes about 300 nanoseconds. This weighting will make sure that the packet will move to the correct place in the current dimension before moving to the next. The algorithm is also designed so that the routing always does the X-dimension first, before jumping to the Y-dimension, and finally to Z-dimension if 3D topology is used.

2. *Dijkstra and shortest-path-first*: After the structure is built, we start with the first node, and traverse through the graph. Every node that we already have seen, we mark. This is done to avoid cycles (dead-locks) in the routing. The shortest-path-first process is done for all-to-all, and we put all the valid paths in a priority list. Next step is to look in the priority list, and pick out the paths with the lowest cost, and make sure that all nodes have a path to all other nodes. Finally we end up with a double array with start and destination node. This array also contains a linked list with all the paths in the network.

3. *Build tables*: In the last step we will look through the array with the shortest paths from all nodes to all destinations. If the packet is just going to be sent through the link we put a 0 in the link table for that node ID, and if we are going to send it out on another link controller, we set the bit in the link table for that node to 1, and we also set the bit for this node in the B-link table for the correct link controller to 1.

This algorithm is executed on all the nodes when they are initialized. It is only executed once on the local nodes. Since a local node does not have information about dead nodes and broken links, we are not able to do rerouting. The optional network manager (described in the chapter 3.2) which runs on a front-end also uses this algorithm, but updates the tables with broken links and dead nodes.

The complexity for Dijkstra's algorithm is O (n^2). Dolphins SCI implementation has a maximum limit of 256 nodes and with the calculation power available in present day CPUs, this will not be an issue.

2.2 Other interconnect technologies

In the interconnect market, there are several competing interconnect technologies to SCI. We will now take a quick look at how some of them work.

2.2.1 InfiniBand

InfiniBand [9] emerged after the merging of two competing designs, Future I/O (Compaq, IBM and HP) and Next Generation I/O (Intel, Microsoft and Sun Microsystems). The merging was complete in 1999, and the companies formed the InfiniBand Trade Association.

The original idea with InfiniBand was that it was supposed to become a System Area Network, which would connect CPUs and provide a high speed I/O, instead of using technologies like PCI and Fiber Channel. PCI was already starting to show its limitations. During the design process the vision was that all I/O and CPU's in a cluster were supposed to connect to a single switched InfiniBand fabric. So far InfiniBand has only been connected to the standard I/O busses on computers like PCI-X and PCI Express.

As of today, the use of InfiniBand has been limited. It is mostly being used for computer clustering applications, and some effort has been taken to adapt InfiniBand as a standard interconnect between low-cost machines. InfiniBand uses a unidirectional serial connection, and point-to-point connections. All InfiniBand nodes are connected to InfiniBand switches. The InfiniBand protocol uses IPv6 headers, which support an efficient connection between InfiniBand architecture and traditional Internet and intranet infrastructures.

The InfiniBand specification classifies the adapters in two categories. Host Channel Adapters (HCA) and Target Channel Adapters (TCA). HCA adapters are present in servers and desktop machines and provide the interface used to integrate InfiniBand with

the operating system. TCA adapters are present in I/O devices such as RAID[14] controllers. Each adapter can have one or more ports, allowing multiple connections to a switch, or a connection to multiple switches. By having multiple paths, InfiniBand can give both increased bandwidth and increased reliability in the case of a link failure.

In InfiniBand, the switches just forward packets between two of their ports. A collection of end-nodes connected to each other through one or more switches is called a subnet. Each subnet must have one subnet manager. The subnet manger is responsible for configuration and management of the whole subnet. Routers, like switches, simply just forward packets between their ports. The difference however is that routers are used to interconnect two or more subnets. Within a subnet, each port on the switch is assigned a unique identifier by the manager. This identifier is called a Local ID (LID). In addition, each port is also assigned a globally unique identifier, called a GID. Routers make use of the GID while routing packets across domains. While switches make use of the LID for routing packets from the source to the destination.

Another feature in the InfiniBand Architecture, which is not available in the current shard bus I/O's, is the ability to partition the ports within the fabric. This is useful for partitioning storage across one or more servers, both for management and security reasons.

The InfiniBand Architecture primarily describes how the adapters, switches and routers are built. Therefore we have several competing software stacks that can not necessarily communicate with each other. Some of these stacks are: OpenIB, Silverstorm, Voltarie and Cisco.

The market share of InfiniBand is increasing, and per 06/2006 they are present in 36 of the top500 clusters [10].

[14] Redundant Array of Independent Disks

2.2.2 Quadrics QsNet

Quadrics is a company that formed in 1996, and specializes in producing hardware and software for clustering computers into massively parallel systems. Quadrics produces the QsNet [11] interconnect technology. Their first design was the Elan2 chip, used with Sun UltraSPARC systems. The next design was the Elan3, which used a PCI interface, and was aimed at the DEC Alpha architecture. The last generation of products is the Elan4 chip.

The Elan4 chip uses a PCI-X interface to the host computer. Elan4 also supports RMA, a 64-bit virtual address space, system calls, and inter-process communication. To speed up memory access, the card has a dedicated memory bank on-board. To do transfers to and from the host machine we have the possibility of two parallel DMA transfers. This is done to maintain full PCI-X read bandwidth. Elan4 uses an embedded 64-bit RISC processor to assist message passing, and offload the main CPU. This embedded RISC CPU is also user programmable, giving the Elan4 card characteristics of a network processor card.

The Elan4 networks are constructed using an 8 way switch component. These switches are arranged in a radix 4 fat tree network, with each having 4 links “down” and up to 4 links “up” to higher stages in the network [11]. This fat tree topology is used because of good scaling, and several routes between the nodes. This topology is defined as a logarithmic network topology (Figure 9). One of these 8 way switch components build up one “port” in a QsNetII switch (Elan4), and these switches are available in 16 ports and 128 port editions.

On the physical link, Elan4 uses 10 LVDS pairs. Copper cables support link lengths up to 13 meter, for longer distances, a fiber link can be used. Since they use two bidirectional links they can carry a theoretical 1.3 gigabytes per second each direction.

Elan4 is today used in several of the top500 clusters, and per 06/2006 they are present in 14 clusters on the top500 list [10].

2.2.3 Myrinet

Myrinet [12] is a high-speed local area network designed by Myricom. Myricom started in 1994 with its first Myrinet implementation, as an alternative to Ethernet to connect nodes in a cluster. Myrinet was designed to have much less communication overhead than Ethernet, thus providing better throughput and less latency.

Myrinet uses two fiber links, one upstream, and one downstream. They are connected on a single connector. Myrinet uses point-point links, and the network is built up by one or more switches. The last generation of Myrinet products feature 10 gigabit per second transfers in both directions, and they are interoperable with 10 Gigabit Ethernet on the physical layer with cables, connectors, distances and signaling. Myrinet is also like Quadrics Elan4 based upon an on-board embedded processor to offload work from the host CPU, but the interface on the Myrinet cards is more limited and can not be programmed to the same extent. The cards are connected to the host computer with a PCI-X interface, or a PCI Express interface.

Myrinet switches are 8-256 ports, and in the current generation the 8 and 16 ports switches are full crossbars. Large Myrinet networks use a similar topology like Quadrics Elan4 (QsNetII). This topology is called a Clos network topology (Figure 9). This topology very scalable, and gives a degree of fault tolerance, because the end-point switches is connected to one or more spine switches.

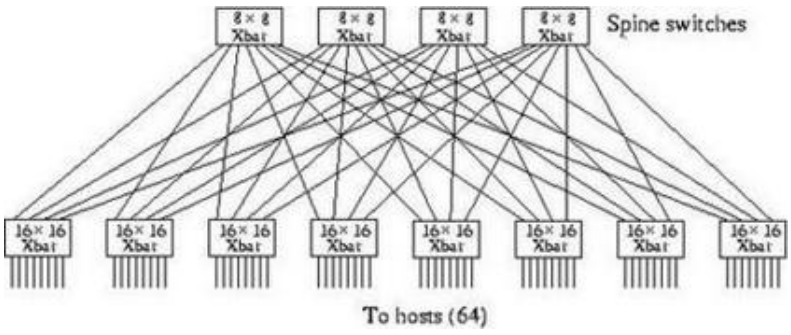

Figure 9: Example of a logarithmic network topology (Clos network topology)

A Clos network is an example of a logarithmic network with the maximum bisectional bandwidth of the endpoints.

Myrinet is also popular on the top500 list of clusters, event though their share have decreased. Per 06/2006 they are in 87 clusters on the top500 list [10].

2.2.4 Advanced Switching Interconnect (ASI)

Advanced Switching interconnect [13] is a switching-interconnect technology, that try to combine features of existing proprietary fabrics, like InfiniBand with industry standard technology. The design and development group is called Advanced Switching Interconnect Special Interest Grout (ASI-SIG), and they are an extension of the PCI Express technology. The only element added from PCI Express, except from the ASI header, is a sequence number, and a CRC^3 checksum.

The Advanced Switching technology overlays the physical and link layers of the existing PCI Express technology. The specification call for an ASI transaction layer on top of PCI Express, to provide source-based path routing, compared with memory mapped PCI Express routing. This allows ASI to use more flexible topologies such as star, dual-star and mesh. ASI has the same possibility, like PCI Express, to achieve higher bandwidth by combining multiple lanes in a single link. The ASI links operate identically to PCI Express, using a serial bus, where one lane has a theoretical bandwidth of 2.5 gigabytes per second.

Protocol Tunneling is also an important concept. Advanced Switching encapsulates data packets, and attaches a header that routes it through the fabric regardless of format. This header is supposed to have a Protocol Information (PI) field for the destination node to determine the packets format. This also enables us to connect computers with PCI Express packets through ASI.

Advanced Switching uses a routing methodology called *Path Routing*, where the source of the packet provides all information needed by the switches to route the packets to the destination. Path routing is another name for source-based routing. This reduces the complexity of the switches, since they only have to switch packets based on headers, not have advanced logics for routing.

No products have yet been made on ASI. Even though PCI Express is in most computers sold today. The PCI Express bus is primarily being used for graphics adapters.

2.2.5 Gigabit Ethernet

Gigabit Ethernet [14] is standard in most new computers today, and it is currently being used in around 50 % of the top500 clusters [10]. Gigabit Ethernet is based upon the IEEE 802.3 specification [14]. It is a point-to-point network, and it primarily uses switches to connect.

Gigabit Ethernet is both available in copper cables (CAT6 cables) and fiber optics. The next evolution on the Ethernet family is 10 Gigabit Ethernet, which is specified under the IEEE 802.3ae standard [14], and rely so far only fiber optics.

Ethernet is cheap compared to dedicated interconnects like InfiniBand, Myrinet, Quadrics and SCI, but it lacks several important features like the possibility for shared distributed memory. The IP protocol stack normally used by Ethernet is also a heavy protocol stack, and in order to be efficient, work has to be offloaded from the CPU.

A lot of clustering software is already available for Ethernet, as well as several distributed file systems and other MPI[15] interfaces.

[15] Message Passing Interface

3 Fault-tolerance

Fault-tolerance is a very important aspect in several cluster applications, and in particular databases and other real-time applications. In a cluster we can get several types of errors. The two most common errors are loss of a node due to software problems, and loss of communication on a link due to hardware problems.

In SCI networks today, fault tolerance to routing problems is not a supported feature by default; this can be done with an optional software-based application. No automatic low-level driver mechanisms that will try to restore the connection between the nodes are available. We will now look at hardware and software based mechanisms to protect SCI networks from potential errors.

In the next section we will look at mechanisms in Dolphin's current generation of SCI hardware. We will start with hardware based mechanisms, and look at mechanisms in the lower parts of the driver, and finally look at a software approach.

3.1 Hardware-based mechanisms for fault-tolerance in SCI

The most common error seen in the current generation of Dolphin SCI hardware is transfer error that can happen because of bad connectors, problematic cables, and other non SCI related hardware problems. In Dolphin's latest generation of SCI components we rely upon a hardware based mechanism to prevent errors during data transfers. The sending node generates a 16-bit checksum and adds it to the tail of every packet. The receiver checks this checksum when the packet arrives. If an error is detected, the packet will be discarded, and an attention interrupt will be generated in the local driver. The sender will not recive an ACK confirming the packet and it will send an attention interrupt.

Another feature in the hardware, to protect the network from accumulated packets with a destination unknown to the nodes on the link is a *scrubber*. The scrubber node on the ring receives a packet addressed for itself, with the scrubber bit set it will be automatically

discarded, and an attention interrupt will be sent to the driver. The reason for doing this is that if packets without a valid destination are not handled they, will accumulate on the ring and finally choke up the system.

The scrubber role has to be elected, and this is controlled by a hardware negotiation protocol. This negotiation protocol is based upon a standard ring based election protocol [15]. When an SCI ring becomes operational a hardware initialization protocol is triggered, and the scrubber is elected in this mode. Every link controller checks the link frequency, and sends an initialization signal to the next node downstream. They also send out a unique hardware identification number, called UID. This hardware ID consists of a 16-bit field for the card type (model number), an 8-bit field for node ID, and a 12 bit field for the cards serial number. This UID must be unique if the initialization process is going to complete successfully. When a node receives this UID, it will check if it's higher than its own UID, and if so it will send the received UID to the next node downstream, if not it will replace it with its own. The highest UID will win this negotiation, and become the scrubber in the SCI ring. The scrubber will set a bit in the SCI header on every packet that passes by in the ring.

On the B-link between the PSB controller and the link controllers we have a parity check. Before the packet is sent out on the B-link it is put into a B-link frame. If there are any parity problems with this frame, the packet will be discarded, and an interrupt will be set in the driver. These types of errors usually only occur when there is an issue with the hardware. The B-link also has a feature similar to the scrubber on the rings. If a B-link frame is not taken by any link controller, or the PSB, the frame will be discarded, and a B-link timeout interrupt generated.

The two remaining features that we are now going to describe is used for flow control, but overall in the cluster they are two important features for preventing packets and responses accumulating, and choking the communication.

The hardware has a remote throttling mechanism built in. This feature allows a value to be set in the link controller (75 % by default). If the send and recive buffer is filled above this value, a bit will be set in the ACK-packets that are being sent back. This bit will make the sending node wait for a couple of cycles before sending the next packet. We also have a feature like this, but locally implemented in the PSB controller. If the PSB tries to reach the link controller, and the link controller is busy, it will get a signal back, and a back-off algorithm is initialized.

A mechanism to stop flooding of packets on an overloaded ring is also built in. Every link controller from Dolphin's last generation has buffers for 8 incoming, and 8 outgoing packets. If this buffer is full on a node, it will generate a reply to the sending node with a *busy bit* set in the header (Figure 10) before it discards the packet.

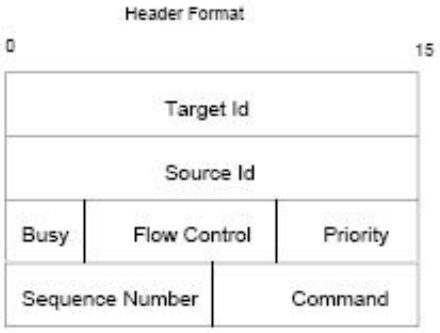

Figure 10: SCI header format

3.2 Software-based mechanisms for fault-tolerance in SCI

On the low-level driver side in the IRM, a heartbeat mechanism is used to tell the receiver node that it is alive. When two computers in the cluster want to communicate, a session is enabled between the nodes, and a heartbeat is by default sent out 20 times per second. It is possible to tune this parameter in the driver. The heartbeat is just a simple counter, which is incrementing itself every time before it is sent. A watchdog in the driver has the responsibility to check the heartbeats. If three heartbeats are missed, the driver will assume that there is a problem. If a potential problem is detected, the local node has the possibility to initiate a remote read on the node it communicates with. This

remote read operation will read a scratch register with information about the status of the node, and it will only be done if we can not recive heartbeats. If a session between two communicating nodes is lost, the driver will try to re-establish the connection.

Dolphin's SCI implementation has the possibility to use software in the higher level of the driver to handle errors that require rerouting in the cluster (Figure 11). In fault-tolerance, this approach can be classified as a static reconfiguration approach. The software is optional, and is built with a traditional server-client approach. The software is available for both Linux and Windows, and it uses communication over Ethernet and TCP/IP to reach the nodes in the cluster. This application contains a network manager that is running on the cluster nodes, and a network administrator running on a front-end. Information about the cluster, for instance adapters present, and their node ID is configured on the front-end.

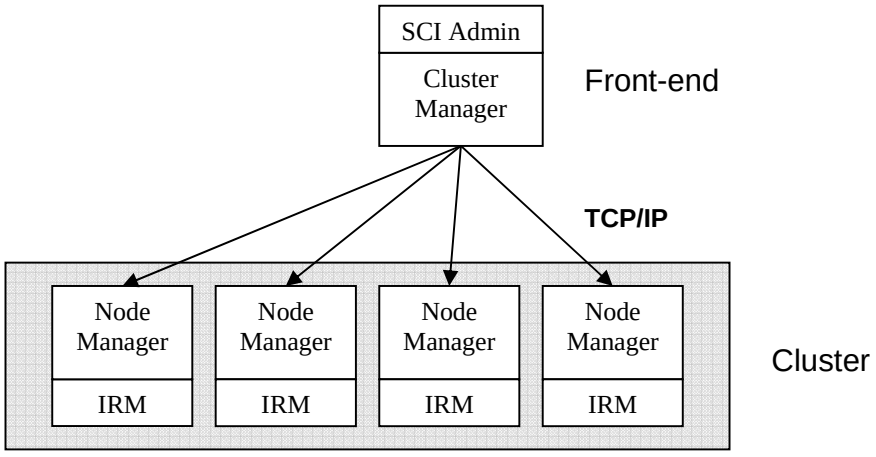

Figure 11: An overview of the network manager topology

The front-end application will poll all nodes in the cluster over Ethernet and TCP/IP every 60 seconds by default. (Figure 12) The nodes reply the front end with their cable status, adapter status and link information. If a problem is detected, the network administrator will disable the links affected by the problems. When this is done, the cluster is set in a special mode to prepare a new routing. This mode sets the cluster in a fatal-mode, and disables all the communication between the nodes. This fatal mode is

used to stop all communication and prevent packets and responses from being sent while nodes in the cluster are being configured. The administrator then generates a new set of routing tables based on the broken node/link lists it received from the nodes. The new routing tables are distributed to all nodes in the cluster. The SCI cards in the cluster will do a reset, and initiate the new routing when it is received, and fatal-mode will be exited, sessions between nodes reestablished, and communication can resume to normal.

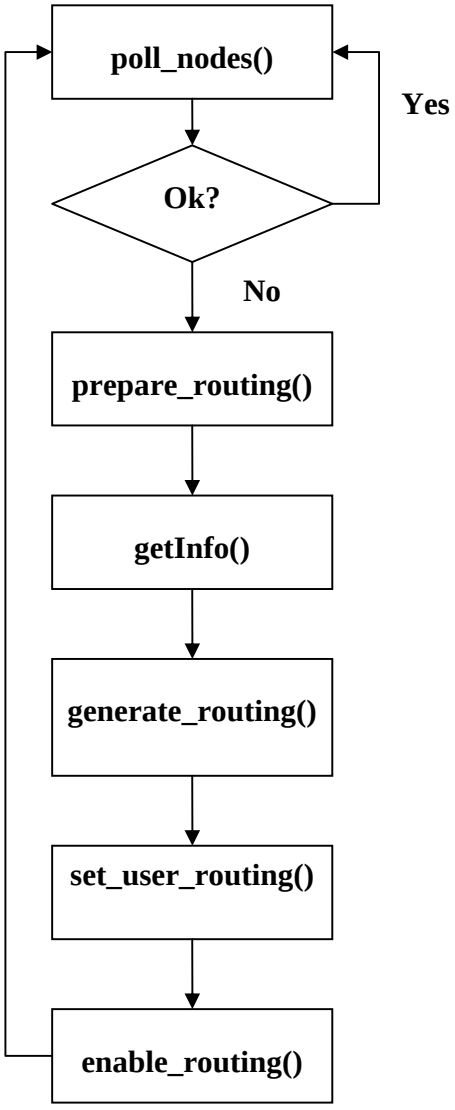

Figure 12: Flowchart for network manager

1. *poll_nodes():* This function polls all the nodes in a given period of time. The poll is done over TCP/IP. Poll_Nodes will compare the status it receives with the last status to see if there is any change.
2. *prepare_routing():* This function sets the cluster in fatal-mode, which disables communication and cleans up any unwanted interrupts and removes all the packets in transit.
3. *getInfo():* The getInfo function collects information from the nodes in the cluster, about cable status to detect dead links, and it makes sure that all the nodes is present. If a node does not reply after three attempts, we will assume it is dead. Information is collected over TCP/IP.
4. *generate_routing():* This function generates a new set of routing-tables based on the new information from the getInfo function. This routing-table is generated with the standard routing algorithm. Tables are being sent out to the nodes with TCP/IP.
5. *set_user_routing():* This function sets the new routing on the nodes, and prepares the hardware.
6. *enable_routing():* The enable_routing function enables the new routing, and communication is started. We now return back to poll_Nodes to check the new status in the cluster.

The calculation of routing tables normally takes around 5 seconds in a small 4 node cluster. In a worst-case scenario; when an error occurs right after a poll, it might take over 60 seconds until a new configuration is applied, depending on how often the driver is configured to do a poll. The network manager can be configured to poll more frequently, but this will cause more traffic over Ethernet. The Ethernet connection is also a weak point in this approach. If the front-end is unable to get reply from a node after two attempts, the manager will assume that the node is dead.

These issues with the network manager, makes the existing solution not suitable for real-time applications, primarily because of the time it takes before the front-end is notified

and a solution is applied. If a problem occurs in a cluster that does not use the centralized front-end with the network manager, communication will be disabled until a manual reset is done.

3.3 Fault-tolerance with redundant hardware

The SCI driver has built-in support for fault-tolerance with redundant hardware. This enables the user to have two or more SCI cards (Figure 13), and use them as virtual adapters. Two modes are available. Striping mode combines both adapters to increase the available bandwidth for applications. In this mode, PCI or PCI-X bandwidth will be the bottleneck. The last mode is mirroring mode. This allows the data to be duplicated for better reliability.

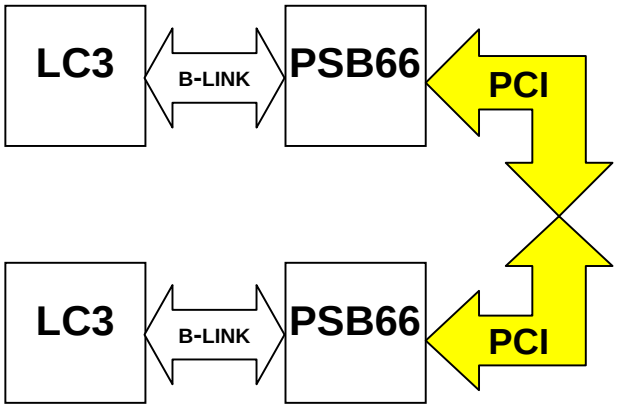

Figure 13: Virtual adapters for redundancy in SCI

There are several issues with the redundant hardware approach. The main issue is that it will not protect the applications if a node dies, or freezes and we need a rerouting in the system. Both these cases will bring down all the rings the dead or frozen node is connected to. Therefore redundant links will only give protection if one of the links is disabled. Another drawback with all redundant hardware is the increased cost, and the fact that we have hardware that is unused or unutilized most of the time.

3.4 Different types of errors

We have four categories of common errors in an SCI network that trigger the need for a fault-tolerant solution to routing. Without support for fault tolerance these errors can be fatal for the cluster.

Cable problems: Dolphin's current generation of SCI hardware uses a 50 pin high density connector, and several problems can occur. One error can be a bad connection at one or several of the pins. To try to prevent these errors, some of the pins on the SCI connector are shorter than the pins responsible for data-transfer. These pins are used by the hardware to detect if a cable is partly out. A cable partly out can in some cases interrupt transfers and corrupt packets. The hardware will detect these errors with the CRC checksum and the packets will be discarded. This will generate an attention interrupt, and increment a counter in the local driver. On the remote node the packet transfer will time out, and it is up to the application if it will try to resend. This problem will not cause a rerouting in the cluster. If several pins loose connection and the hardware detect a synchronization problem on the link, the problem will be considered fatal, and the link will be disabled by hardware. This will result in the entire ring being disabled. This error will cause a rerouting. Anther occurring problem is that some old SCI-cables might have a weak point between the connector and the cable. These intermediate failures can be both hard to detect, and hard to debug.

Dead node: A dead node is defined by a power loss on the node. This is usually caused by a faulty power supply. A dead node will cause a synchronization error on all the links connected through this node. The other nodes on the ring detect this error at once, when the clock synchronization signal in the SCI cables is lost. This has to cause a rerouting for all the nodes affected.

Failure on other hardware: Other components in the cluster can also fail, even though electronics in modern computers is normally very reliable. If a failure does happen, it can often be traced back to mechanical components like hard-drives, cooling-fans and power-supplies. This will often lead to an immediate stop, or a software lockup. Errors on the

SCI hardware are very rare, but they can also occur. This is often due to bad soldering, or old connectors. All these errors will require a rerouting in the cluster.

Software lockup: This error often occurs when software on a node triggers a fatal error in the driver or the operating system kernel. This will halt the system, but the SCI card might still be alive, and sending out the clock synchronization signal, but is unable to execute exceptions caused by the driver. It is in some cases possible for the card, if power is available, to route packets, due to the routing tables still being loaded into the link controllers when the driver was initialized. To detect these types of errors the heartbeat mechanisms between the two communicating nodes is used. If one of the nodes does not receive the session heartbeats, it will cause the driver to assume that the node is dead. The session to it will be disabled, and a rerouting will be needed.

3.5 An overview of our approach to fault-tolerance in SCI

As mentioned before, SCI had limited support for fault-tolerance to problems that require rerouting. We wanted to implement a better and faster solution to this issue. Instead of using the slow interconnect manager, we implement a fault-tolerant routing. The project started with several constraints. The main constraint is that SCI has been shipping for years, and it is not possible to change the hardware. The last revision of the LC3 link controller (B-link to SCI bridge) has been shipping since year 2000. The SCI standard was defined in 1989. Another, but smaller constraint was that we had to integrate our new routing mechanisms into an existing software-stack, without breaking compatibility for the layers and API's above in the stack. The driver also needed change. If an error occurred the local node would just disable communication and sessions for the application, thus we had to implement support for making decisions about local routing.

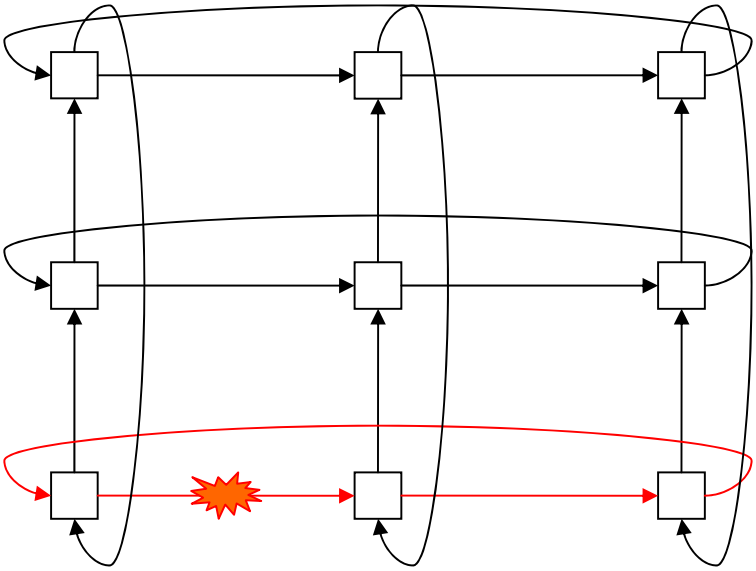

Figure 14: 9 nodes in a 2D SCI network with a cable out

In Figure 14 we can see an SCI network with an X-dimension cable out. As described in the section about the SCI technology, one link down on a ring will bring down communication on the whole ring. All the nodes connected to this ring will detect this immediately, because a built-in hardware mechanism detects the loss of a synchronization signal on the cable.

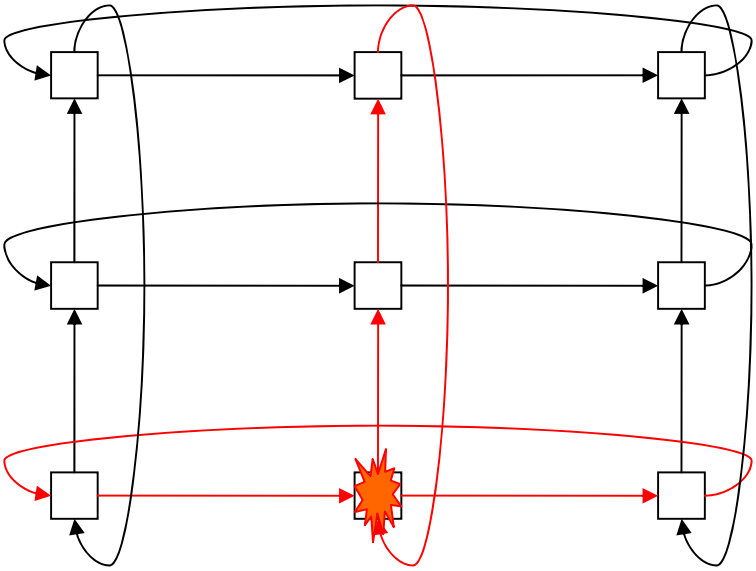

Figure 15: 9 nodes in a 2D SCI network with dead node

In Figure 15 we have a case where a node is dead. This will cause all the rings connected to this node to be disabled. The other nodes affected will detect this with the same mechanism used to detect a disconnected cable.

The idea to our approach came after a presentation at Simula Research Laboratory about FRoots, a Fault Tolerant and Topology-Flexible Routing Technique [16]. In this approach, the network and routing tables are set up for failover from start with redundant paths. This approach is for point to point networks. Dolphins SCI technology is based upon rings, either one ring, or several rings in a 2D or 3D torus topology, so a lot of adaption would have to be done.

After investigating the SCI specifications [1] and the specifications on Dolphin Interconnect Solutions LC3 link controllers [3], we soon discovered that the hardware used small routing tables with only room for 256 addresses (node ID is 8-bit). This limitation is due to the fact that the first SCI implementation by Dolphin is from 1992, and in 1992, over 256 nodes in a cluster was unthinkable. SCI was originally based upon a topology with switches, and early implementations by Dolphin were also based on the

switch-approach. Later on, when first 2D, and then the 3D torus topology was introduced the need for a lager address space emerged. The link controller is also currently not able to change the destination address after a SCI packet is sent, thus a SCI packet can only have one destination address.

With these limitations in mind it was decided to try implementing a solution that locally calculated new routing tables based on the operational links on this node. As described in the SCI chapter. Dolphin's current generation of SCI hardware uses two routing tables per link controller. One table, called the link table is used to tell the node if it should take the packet and send it out in the B-link, or just put it in the pass-through FIFO queue and back out on the ring. The second table is referred to as the B-link table. It decides if the link-controller shall fetch the packet from the B-link.

This solution would not be instant, since it takes some time to calculate the new tables, clear all interrupts generated when the error occurred, and reinitialize the link controllers with the new tables. Since the other nodes in the cluster might not be aware of any problem, and just rout the packets back into the problem, we also had to configure the standard tables on all the nodes to handle packets it would not see in normal operation. We will refer to this as speculative routing.

The main idea is that if a node or a ring is dead, the first node that knows about problem would just forward the packet another step along the ring in the same dimension. Then the next node downstream that receives the packet will have to be set up to handle this packet as the previous node would have done in a normal case. The default routing in SCI is already made to route packets to the right place in the X-dimension, before sending it to the right place Y-dimension, and finally to the right place Z-dimension if a 3D topology is used.

To just send a packet through the link controller with the FIFO buffer costs 50 nanoseconds. In comparison, a jump to another dimension, through the B-link and out on another link controller costs 300 nanoseconds. Our approach will in many cases generate

at least one extra dimension jump for the packet, so we would expect to see a slight increase in latency. We also expect to see a slight decrease in bandwidth, since more resources will share the same path. Another parameter we are measuring is how long it takes after a problem occurs before a solution is in place that fixes it.

To implement support for fault-tolerant routing on the existing generation of SCI hardware from Dolphin Interconnect Solution, several elements had to be changed and added to the low-level part of the driver, also referred to as the IRM. These changes will be described in details in chapter 5.

4 Related work

A lot of work has been done on this topic, but most of the research has been done on point-to-point networks such as Myrinet [12], InfiniBand [9] and Ethernet [14]. In ring based interconnect networks like SCI, there has not been much activity in this field lately. The reason for this is that the SCI technology is a quite old technology, with little academic activity.

In this section, we are first going to take a glance at different approaches to fault-tolerance, with an example in each field, and finally take a look at a fault-tolerant approach for a ring based topology.

Fault-tolerant routing can be divided into three groups:

- *Reconfiguration* is the first group, and it includes both static and dynamic reconfiguration. In reconfiguration, we detect the error, calculate a new routing, and enable it.
- *Source-based*: The client detects the problem and retransmits the packet. This method is often described as a non-local method, since the error is detected remotely, and the local client gets new information on how to send the packet.
- *Switch-based* is the last group. Here we only use information available locally to do the rerouting, without notifying the source.

We are now going to take a more detailed look at these three groups of fault-tolerant routing algorithms, with some examples.

4.1 Reconfiguration

Reconfiguration is the first group of fault-tolerant networks. This group includes static and dynamic reconfiguration. Reconfiguration involves three steps. The first step is to detect a failure in the network. The second step is to calculate a new routing, without the faulty component. This is often done with a graph-search through the network. The third step is to enable the new routing strategy. The graph search is done with the knowledge

of the whole topology, and offer maximum flexibility. A packet can always be delivered if a physical path exists between two nodes. The drawback of reconfiguration is that the routing overhead is significant. The process that traverses the graphs often requires a lot of CPU resources.

In networks that use static reconfiguration, the whole network must be halted before we do a reconfiguration. This is up to now the most common procedure in the industry, and this is also the procedure used in the existing Dolphin SCI products. Dolphin's network manager discussed in more detail in chapter 3.2 halts all communication in the network while the reconfiguration process is running. The static reconfiguration process is, like in the network manager, often a global process.

Dynamic reconfiguration is relying on allowing communication in the network, while the reconfiguration process is running. This can be challenging, due to the possibly of deadlocks in the network. An example of a protocol that uses dynamic reconfiguration is OSPF[16] (Open Shortest Path First) [17] in Internet. OSPF uses the link-state routing principle [18] and is an open protocol. The protocol is based upon Dijkstras shortest path first algorithm [8]. OSPF is used between routers in the internet, and is often referred to as an IGP[17] (Interior Gateway Protocol). OSFP does not have any issue with deadlocks. This is because in Internet, packets can be dropped, delivery is not guaranteed.

For real-time applications in interconnects, dependability is very important. The systems must be able to operate and reconfigure, while maintaining the QoS[18] needs for the application. The process of introducing new routing-functions after an error has occurred in the net may introduce dependencies between network resources. In the paper "A Methodology for Developing Dynamic Network Reconfiguration Processes" [19], a dynamic reconfiguration process is described. The goal of this approach is to propose a simple method to do dynamic and deadlock-free reconfiguration. This approach is based upon the following methodology. The reconfiguration process can be derived from a

[16] Open Shortest Path first
[17] Interior Gateway Protocol
[18] Quality of Service

sequence of confirming pairs of routing functions. The atomic process of transitioning between them is called Adding Phase, or Removing Phase. The reconfiguration is done between compatible or coexisting routing functions. If the coexisting routing functions are not compatible, a service should be in place to halt injection of packets from the problematic region of the network

This function does not add any requirements on the type of routing function, virtual channels in the net, or network topology.

Double Scheme [20] is another method for dynamic reconfiguration. Double Schemes avoid reconfiguration by spatially separating resources. This is done by doubling the number of resources used by the routing algorithm, by using two distinct sets of virtual channels. One set uses the old routing function, and the other set uses the new routing function. This is done to escape deadlocks. All packets will be drained from the old function before switching to the new function.

4.2 Source-based

Source-based rerouting is based upon the principle that the clients detect the problem and resend the message. These approaches are often referred to as non-local approaches, since the error is detected remotely, and the local client gets the new information to send the packet.

Advanced Switching Interconnect [13] is an example of a interconnect network that allows the uses of a source-based mechanism. ASI uses a path routing scheme, where the routing information is embedded into the packet header. If an error occurs in the network, the information about the error and the new topology has to be sent to all the clients, and the lost packets have to be resent.

This approach is discussed in “A Routing Methodology for Achieving Fault Tolerance in

Direct Networks" [21]. The goal of this paper is to introduce a fault-tolerant routing algorithm for regular interconnection networks. The method is based upon the selection of an intermediate node for each pair of source and destination. The method is adaptively routing packets to this intermediate node, and from this intermediate node, the packet is adaptively sent to their destination. This method requires support for virtual channels in the network. The packets will travel through minimal paths along both sub paths. It has been tested that this method works on large networks on the size equal to the BlueGene/L supercomputer [22]. Evaluation showed that the proposed method is 5-fault tolerant for 64 nodes.

4.3 Switch-based

In the switch-based approach, the source does not notice anything about the problem, and the switch does the job of handling the problem. This method is often referred to as a local method where the local node or switch takes the necessary decisions about how to overcome the error. Our fault-tolerant routing method for SCI networks has some of these capacities, since the local node that detects the problem sends the packet in another direction, and the next receiving node is prepared to handle this error. To the best of our knowledge this is the first implementation of switch-based fault-tolerance in SCI networks.

An example of a switch-based method for fault-tolerance is FRoots [16]. The goal for FRoots is to have the network available and connected, while using redundant paths. FRoots is the successor of MRoots [23]. MRoots uses virtual channels and a number of Up* / Down* graphs with the roots spread to reduce bottlenecks. The FRoots algorithm is based upon a load balancing version of Up* / Down* routing [24], and it uses a small number of virtual channels to achieve the redundant properties. When FRoots detects a fault it uses the redundant paths to overcome it, and dynamic global reconfiguration to prepare for future faults and changes. This is done with an approach called Double Scheme, described in the reconfiguration section. The presence of reconfiguration method makes FRoots a hybrid method.

4.4 A fault-tolerant ring-based topology

Some related work has been done on IEEE 802.17 Resilient Packet Ring [25, 26]. Resilient Packet Ring was standardized by IEEE in 2004. In this protocol we use several point-to-point links between stations. The connections between the stations are bidirectional, and this allows resilience. RPR is a new MAC-layer technology that may span into MAN[19]s and WAN[20]s. RPR can easily bridge to Ethernet [14]. This bridging is easy because RPR uses the same 48-bit MAC-address format like the other 802 family protocols. However RPR also have the possibility to encapsulate Ethernet frames with an RPR header. RPR and SCI use the same basic access method to the ring. Both uses buffer insertion mode, where you have buffers incoming and outgoing traffic to the local station, and a buffer for pass-through traffic. In RPR this access method is further developed with the availability for packet priority, with multiple queues.

RPR has several modes for resilience. When a station recognizes that a neighbor is dead it sends out topology information that indicates a broken ring. Packets are being sent in the other direction. It is also a possibility for a station to wrap the ring if one of its neighbors is dead. Instead of discarding the packets it is possible send the packets back in the other direction. This can only be done if the packets have this option set in the header.

Another ring based network topology that is similar with RPR is FDDI[21] [27]. In this topology, a set of counter rotating rings is used. FDDI uses the same access method as Token Ring [28], but the counter rotating ring gives it resilience, since the station that detects the error can send the traffic out on the other ring in the opposite direction. In this topology, the outer ring is used for data, and the inner ring that goes in the other direction is redundant, and only used during failures. This process of reconfiguration is called self-healing.

[19] Metro Area Network
[20] Wide Area Network
[21] Fiber Distributed Data Interconnect

5 Implementation

In is section we are going to describe the implementation for the fault-tolerant routing approach. The implementation is done in the routing module of the IRM driver. Dolphin's SCI driver and utilities are available for download on Dolphins webpage [29] under GPL license. The implementation and integration in the driver is done in cooperation with engineers at Dolphin Interconnect solutions. The entire IRM is implemented in the C programming language. We will focus on the implementation of the fault-tolerant 2D method in this section. From now on and during the remainder of this thesis the fault-tolerant 2D mode will be referred to as 2D HA[22] mode.

We will begin this chapter by looking at the changes needed in the driver to support local rerouting, before investigating the interrupt mechanism the driver uses to detect problems on a link. We will continue with taking a look at the two different handlers in the driver that has the responsibility for recovering from a fatal error, as well as for setting up the link controllers and calculateing the new routing. Next step is the changes we had to do to the existing routing to support High Availability, both on the local node level, and from a global perspective.

We will end this chapter with a quick description of a tool that is developed to measure the communication downtime.

More information about the driver, the structure of the driver, and where to find the edited files can be found in the Appendix.

5.1 Changes in the driver to support local rerouting

Up to now in standard 2D and 3D mode, the driver is not able to do rerouting decisions on a local node. It will just wait until the link is back up and running again, or waits for the network manager to send a new routing table, and disable links if needed.

[22] High Availability

The driver holds special modes on how it will handle routing. The card must be configured for this mode the first time it is used. After the first setup, the card will remember this mode, but can be reconfigured with the *sciconfig* tool. The driver stores this active topology information globally, and we can use simple if-sentences whenever we want to do a special thing with a topology in the code. The modes that are available for the driver are: ring topology, 2D, 3D, 2D Direct HA, 3D Direct HA, 2D HA and 3D HA. The Direct HA is a more static HA topology, and not a part of this assignment.

Dolphin's latest generation of SCI cards detect errors with interrupts. The PSB chip contains several registers that get set if something is wrong. Cable and link problems are registered in a registry called the IO_DATA register [2]. If a bit is set in the IO_DATA register, it will also trigger a bit in the interrupt status register (ISTAT) [2]. ISTAT is the primary register for the interrupt mechanism on the SCI card. The ISTAT register is connected to a register called IMASK [2]. This register allows the driver to mask (hide) interrupts from the hardware. This is very useful in the event that we want to disable one or more link controllers, and not have them to trigger any interrupts. The IO_DATA register also have this possibility to hide errors from the hardware.

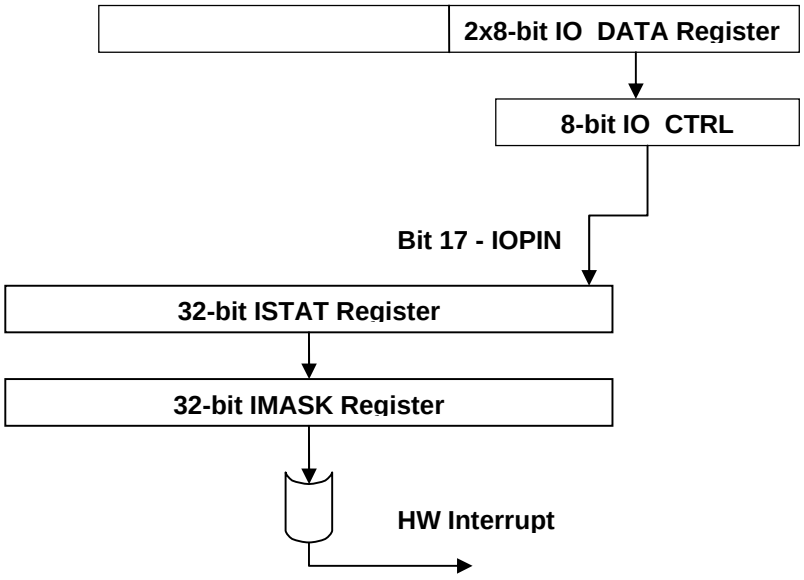

Figure 16: Interrupt registers relevant to rerouting

In Figure 16 we can see a simple overview of the registers relevant for rerouting. In the IO_DATA register the 8-bits are duplicated. The lower 8-bits are used by the interrupt mechanism, and can be masked by the IO_CTRL register. The upper 8-bits are duplicates of the lower, but they are only used by the driver and the network manager, not the interrupt mechanism.

We also have a mechanism built into the driver that enables it to simulate an interrupt that will require a rerouting. We call this a forced reset. When we force a reset, we set a global parameter in the driver. This will trigger the error handler, making a complete loop through the procedure for rerouting. As seen in Figure 17. This was originally used by the network manager to enable the routing tables it sent out, but we also use it in our approach. (Figure 18)

It’s important to notice that when one card does a reset, and goes through this process, it will affect all the nodes in the network. This is because a reset stops the synchronization signal on the SCI Link while the link controller is configured, thus resetting the other nodes in the network.

Now we will take a quick overview on how the driver handles an error that requires a rerouting when set in 2D HA mode:

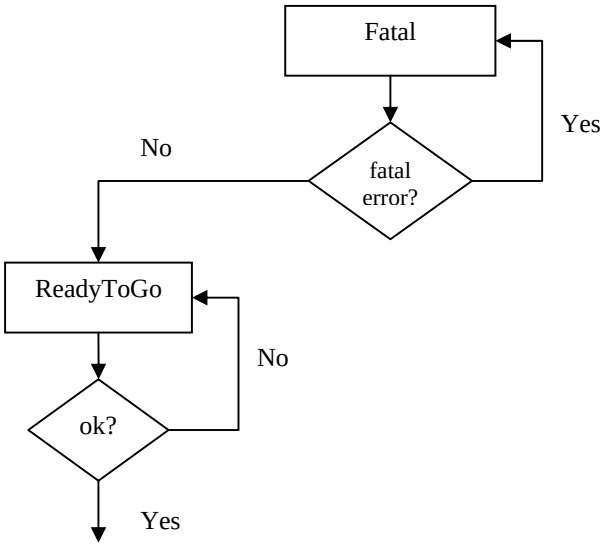

Figure 17: Error handling overview

In Figure 17 we can see a basic overview of error handling in the driver. The process is divided in two steps. The first step is the fatal error handler, called “Fatal” Whenever an error is detected the driver initializes a timer, which is currently set to 30 milliseconds. During this time, the driver will disable interrupts, and clear them. After the timer is finished, the fatal handler will check the card if there still are some fatal errors present. If we still find fatal errors, we make a new loop through “Fatal”, trying to clean up. If no fatal errors are present, we continue to “ReadyToGo”.

“ReadyToGo” is another adjustable timer, which currently is set to 50 milliseconds. The card will now initialize the link controllers, recalculate routing tables, and check the connectivity on the SCI link. After this timer is done, we check if everything is ok. If it is, we can resume operations. However if a problem is detected, another loop through the “ReadyToGo” section is required. The reason for needing another loop through “ReadyToGo” is often that other nodes on the ring are still not operational, and the connectivity fails.

We will now take a more detailed look at how the functions work in the driver.

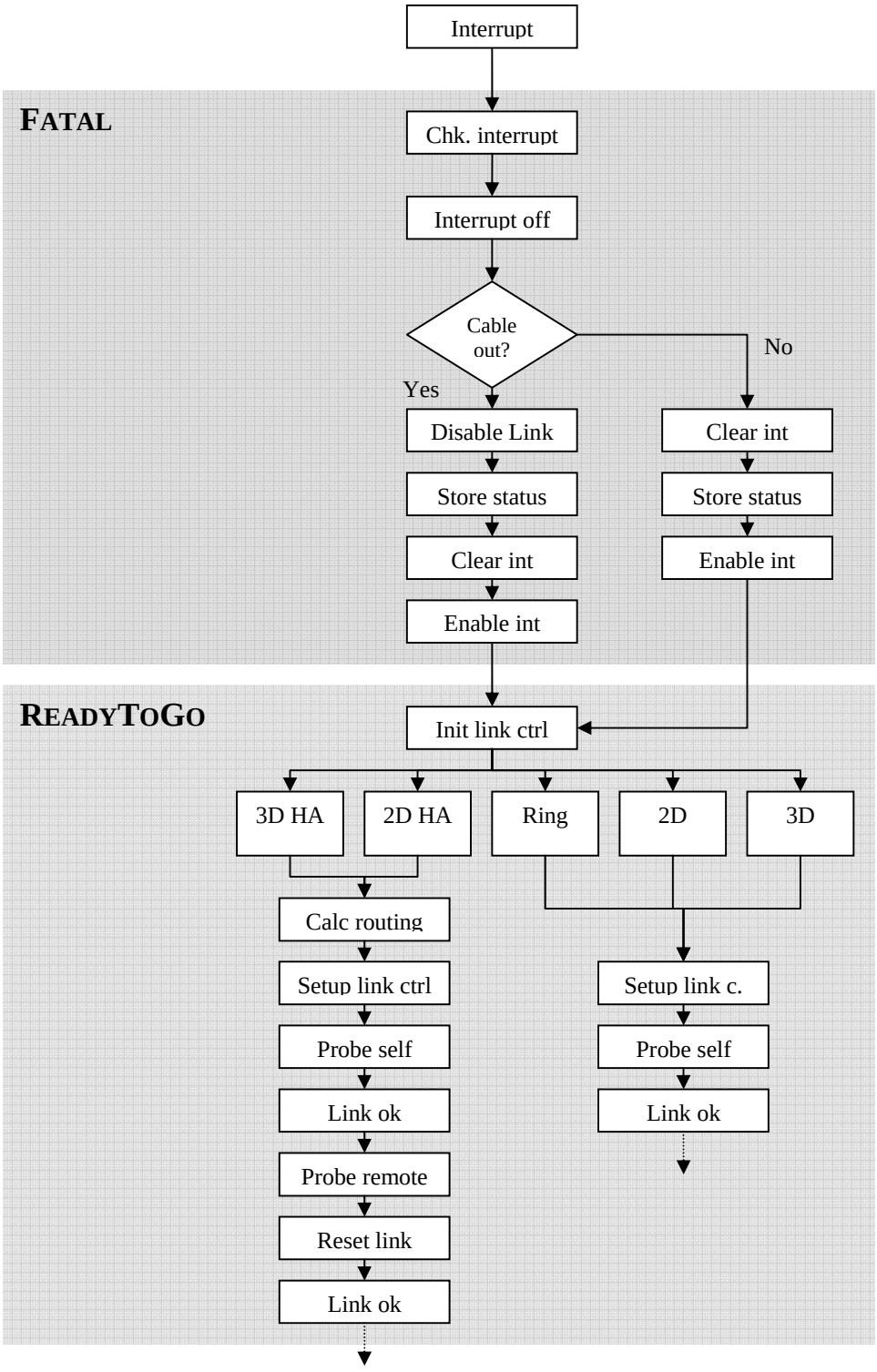

Figure 18: Detailed error handling

Figure 18 is a detailed flow of the functions called by the driver after an interrupt have been initialized. We will now go into details on every step for a cable out event on the "2D HA" path in Figure 18.

The rerouting process in the driver starts with an event on the link controller. This event is an interrupt triggered in the ISTAT interrupt register. All link controller related errors get reported on the IOPIN (bit 17) to the ISTAT register (Figure 16). This will trigger the interrupt handler, and the interrupt will be classified as a *fatal interrupt*. The fatal handler will be started as seen on Figure 17 and Figure 18. The fatal handler will begin with setting the card in a protected mode. This is done by masking all the interrupts with the IMASK register. The reason for doing this is that we do not want more interrupts while handling a fatal interrupt. When the hardware enters fatal mode, all communicating sessions will be disabled, and paused.

The next event is a cable out check. The removed cable will be triggered if the input or output cable on the link controller is not properly connected, if the controller looses the synchronization signal on the ring, or if the status is changed. A broken cable somewhere else in the ring will also be detected because of the synchronization signal. That type of event is also defined as a cable out case.

Now we assume that a cable is out. Next step is to disable the link with the problem. The link controller will still be operational, but we are hiding it from the interrupt mechanism with the IO_CTRL register. The driver also disables the possibility for the link controller to send and recive packets. The disabling of one or more link controllers can only be done in 3D HA and 2D HA mode. For Ring, 2D and 3D this can only be done by the network manager if it is present.

After the link is disabled, we have to store the status of the card. This information is required to generate the new routing tables. We save information about the controllers, if they are enabled or disabled, if the local cables are plugged in, and the cable status of the remote upstream node. After this step is completed, we have to clear all unwanted

interrupts in the PSB, when interrupts are cleared, we enable interrupts again. This is done by removing the masked out bits in the IMASK register.

When the fatal timer in the driver is finished, a check will be done to see if there are any fatal errors left. If we find more fatal errors, the fatal handler has to be restarted, and run again. If no fatal errors are present, it will set up the *ReadyToGo timer* and the driver will start preparing the link controllers for communication.

The ReadyToGo timer is a tunable parameter. For normal 2D and 3D topology with rerouting from the manager, it is set to 3 seconds. The reason for having such long waiting time in these topologies is to allow the network manger to finish. During the development of the HA mode, we used 500 milliseconds on the ReadyToGo timer, but in the finished product, and for the tests later in this thesis, 50 milliseconds is used.

ReadyToGo begins with the initialization of the link controller. Here we send our node ID to the link controller, and we tell it what frequency to operate at. Default is 166 MHz, but on some clusters with cables at 10 meters and longer, 150 MHz is used. After the link controller is set up, we continue, and check which topology that is selected. For this example, we still assume that 2D HA mode is chosen.

Next step is to calculate new routing tables. When the new routing tables are calculated, we first generate a normal case routing table, next we use the status stored away from the fatal handler section to help us to do the calculation for the new topology after the error occurred. The function that makes this table will be described in detail in the next section of this thesis. After calculation is complete, we have a new set of link controller tables, and B-link tables. The step after routing table calculation is link controller setup. In this step, we take the tables generated, and set them in the link controller. After the link controller is ready, we probe our own address on the SCI ring. The probe request is done directly from the routing module down on the PAL-layer (Figure 8) and is the lowest level of accessing the SCI link. This probe request will travel all the way around the ring, and when we recive it, we send an echo back. This is done to make sure that we have

connectivity on the ring. This is also often the step in the process that fails. The reason for this might be that in a rerouting, some of the other nodes in the ring can also be in the reset process, and unable to forward our request. If the probe self step fails, as seen in Figure 17, we have to start the ReadyToGo section again from the beginning. When the probe self step completes, we set the link status to ok, but we do not enable communications yet.

In our first approach, this was where we enabled communications, but due to an issue detected while implementing, we had to make an extra step. This issue will be discussed in more extent in section 5.2.3. In this step, we are probing the node upstream on our X-link. In this probe, we ask for the upper part of the IO_DATA register on the remote node. This register tells us if the Y-link on this node is disabled, or has a cable out. If we get a reply with a cable out or controller disabled from the remote node, we ask for a reset in the driver, forcing a rerouting. This is because we need to generate a new set of routing tables with the changes we got from the upstream node.

The reset will take us through both the fatal and the ReadyToGo section again. After we have done this reset, we set the link status to ok, and enable communication, and the sessions we paused when we started the fatal handler can resume.

The reason for using timers to check on fatal and ReadyToGo after a given time is that there is a small risk that the driver stops on one of the steps, and a the timer prevents this by breaking in, and doing a check if the problem is still there. If it is a new loop through fatal or ReadyToGo is forced.

The approach for the traditional routing modes: Ring, 2D and 3D will just set up the link controller with default routing, or a user routing sent from the network manager if it is used. Check the ring for connectivity, and finally set the link as ok, and start the paused sessions.

We also needed another change in the driver. After a link controller is disabled, a watchdog checks if the cable is plugged back in. When we disable the link controller as described above, we mask the IO_CTRL register. This makes the hardware unaware of anything that happens with that link controller. Hence, the watchdog we use checks the status for the local link in the upper 8-bit of the IO_DATA register, to look if the cable is reinserted. It is programmed to check this every second. The reason for such long timer is that we already have a solution running now, and it's not so time critical to make a new solution. If this link watchdog discovers a change on the cable status of the disabled SCI links we force a reset in the driver, making us loop through the rerouting again.

5.2 Changes to the routing algorithm to support local rerouting

After the driver was modified for local rerouting, the routing algorithm also needed a change. A part of this change is to speculatively set it up to handle any problem, because remote nodes from where the problem happened have no information about the issue, except that they might get packets that they usually never would see in a normal case.

5.2.1 Global changes to the routing

For a global change in the routing, we want the link controller table on all nodes to accept packets for all nodes, except for destinations on the packets' current link. The graph routing algorithm helps us to some extent in this case, since it is set to take the packets to where they are supposed to go in the current dimension before changing to the next dimension.

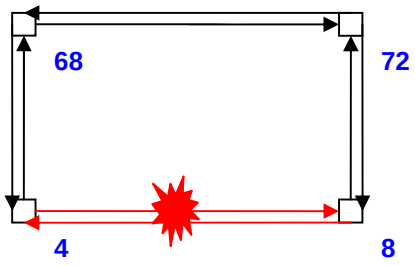

Figure 19: Example of a link with error

The number besides the nodes in Figure 19, and during the rest of the thesis is the node ID. This ID is used as a node name in Dolphin's SCI implementation.

In Figure 19, let us assume that we have a session between node 4 and node 8. They are on the same X-dimension, so the normal traffic would just go directly from node 4 to node 8. In this case, the link between node 4 and 8 breaks down, and node 68 must be ready to accept the packet. In normal operation, node 68 is never supposed to see traffic to node 8 on the Y-link from node 4. When node 68 accepts the packet, and sends it up to the B-link. Graph Routing have already set up the B-link table correctly (route X-dimension first, then Y-dimension), so that a packet with node 8 as destination is supposed to travel through 72, change dimension and then go down to node 8.

5.2.2 Local changes when an error is detected

If we detect a problem on the local node, we first have to check which link has the problem. This information is already saved in the fatal handler illustrated in Figure 18. The problematic SCI link has already been disabled in the fatal handler, so we do not need to do anything with the link table. As described in the interconnect section the link table only controls which packets we are going to take out from the SCI ring, and send to the B-link.

The B-link tables on the other hand must be changed. They tell which packets the link controller should take from the B-link. It is also very important to that we don't have two

link controllers that try to take the same packets. That will cause a B-link timeout error. So we take the B-link table from the disabled link controller and use an OR operation to combine it with the table of the functioning link controller. After the OR operation is complete, we set the disabled link controller's table to 0. If both link controllers are down, we just set both tables to 0

In pseudo code, we use the following algorithm to fix the B-link table:

```
if ((linkUp_0 == FALSE) && (LinkUp_1 == FALSE)) {
        /* Both SCI Link 0 and SCI Link 1 are disabled */
        */ SET B-link table on Link 0 and Link 1 to 0 */
}
else if ((linkUp_0 == FALSE) && (LinkUp_1 == TRUE)) {
        /* SCI Link 0 is disabled, use Link 1
        /* Take B-link Table from Link 1 and OR in table from Link 0 */
        /* Set B-link table 0 to 0 */
}
else if ((linkUp_0 == TRUE) && (LinkUp_1 == FALSE)) {
        /* SCI Link 1 is disabled, use Link 1
        /* Take B-link Table from Link 0 and OR in table from Link 1 */
        /* Set B-link table 1 to 0 */
}
else if ((linkUp_0 == TRUE) && (LinkUp_1 == TRUE)) {
        /* Both SCI links are OK */
        /* Use default routing */
}
```

Figure 20: General algorithm to fix B-link table

We end up with the following general algorithm for local rerouting:

- Use B-link tables of the dead link and combine them (OR) with the table of the link still alive.
- Link controller tables on all nodes must be prepared to recive packets for all destinations, except destinations in the current dimension.

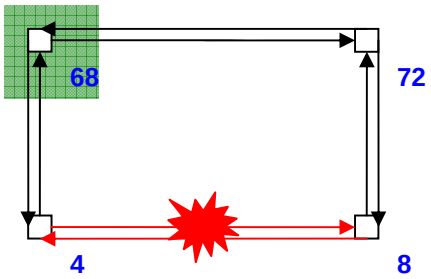

Figure 21: Local rerouting

In Figure 21 we have communication between node 4 and node 8. Let us assume that the X-link between these two nodes breaks down. Node 4 will then take the B-link tables from the X-link (link 0) and combine them with the tables from the Y-link (link 1). The packets will now be sent to node 68 (marked green). On node 68 the link controller routing table for the Y-link (link 1) is already set up take packets to destinations not on node 68s Y-link. The B-link table in node 68 is already set up for sending packets to node 8, and the packet will continue normal via node 72 to its destination.

5.2.3 Problem encountered that needed a workaround

Early in the implementation and testing phase we encountered a problem. Our method was developed before Graph Routing was implemented on SCI hardware. In the routing algorithm prior to Graph Routing, both response and request packets were travelling the same path, but with Graph Routing, both response and request packets were following the X-dimension first, then the Y-dimension rule.

Removing X-cables did not have any issues, but when we wanted to remove Y-cables we got a case where the response message went around the ring once, and got scrubbed by the hardware mechanism that protects the rings from accumulating packets. An example is shown in Figure 22 for a standard 4 node cluster. We checked the Link Controller 3 specification [3] if the scrubber feature could be disabled, but since it is completely controlled by hardware it is not possible to disable it, or temporarily change its behaviour.

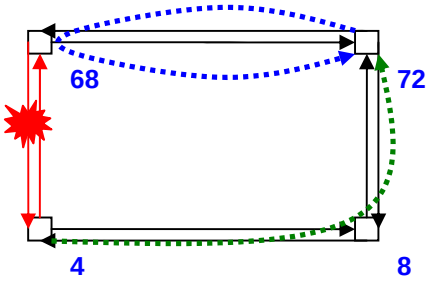

Figure 22: Problem with some broken Y-links

Figure 22 is an example on the issue we encountered. A communication session is in place between node 4 and node 68. The red link is the disabled link, the green line is the SCI request, and the blue line is the SCI response. Since node 72 has no information about the broken link between node 4 and node 68, response packets from node 72 is sent via node 68. Hence, the X-first rule. Node 68 has changed its B-link tables as it is supposed to, and sends it back on the same ring causing it to be scrubbed on node 72.

We are able to reproduce this issue in all clusters every time a Y-link breaks. The problem occurs when the node upstream of the broken Y-link wants to send a packet to a node on the broken Y-dimension, except for the node on our current X-dimension.

Several solutions to the problem were discussed during the implementation phase, before two were selected:

1. **Add an extra redundant Y-ring**

   The first solution to be discussed was the adding of a redundant Y-ring on the end of the cluster. In a 2x2 cluster, it would require that customer to have two extra nodes with SCI hardware and cables available only for fault tolerance. This would increase the price on the cluster, even though the machines for the redundant link did not have to be high end computers, since their only role was to forward packets.

   It would also be technically impossible with the existing hardware to make this ring only accept packets that were affected by a problem on another Y-link. This

redundant link would have ended up with taking a huge portion of the traffic in the Y-direction making the ring a hotspot. Therefore this solution was quickly discarded.

2. **Use 3D cards and use the Z-link as a redundant Y-link**
   This solution was supposed to use 3D cards in a 2D cluster, and configure the third link (SCI Link 2 or Z-link) as a backup Y-link. When the Y-link lost connectivity the B-link and link tables were supposed to move to SCI Link 2, and communication would start. This solution would have demanded that all nodes were ready to recive data on these redundant links.

   This solution would also increase the cost. 3D SCI cards are much more expensive than 2D SCI cards, because of the extra link controller, and a more complex PCB[23]. Extra SCI cables were also needed for the redundant Y-links, and if the customer already had 2D SCI cards all cards had to be swapped with new 3D SCI cards.

   From the drivers standpoint this solution is simple to implement, sine it uses the already existing algorithms, just with SCI Link 2s routing table set to 0 from the start. This solution was also discarded because of a higher cost for the customers, and the need to replace existing hardware.

3. **Use the session mechanism to check the cable status on the other nodes**
   This solution was the initially preferred solution and it involved using the session heartbeats already in use, to not only pass a 32-bit counter, that is incremented on every heartbeat like today, but use the upper 3-bit of this 32-bit message to send cable status from the remote nodes. This cable status can be read out of the upper 8-bit of the IO_DATA register on the receiving node. The heartbeats are by default automatically sent 20 times per second between communicating nodes to make sure that both nodes is still alive. This session heartbeat parameter is also tunable.

[23] Printed Circuit Board

When this solution was developed we ran into several problems. Even though both are in the IRM part of the software, sessions are at a higher level than routing, since routing have to be ok before sessions are activated. This took too much time, and when we did get the information and recalculated the routing, we needed to make an extra reset on the hardware to include these changes, and because of the extra time this took on the other nodes in the cluster, the possibility that some cards needed several loops through ReadyToGo increased. It was also a requirement to enable a communicating session between all nodes (all-to-all), just to make the rerouting work. This further complicated the implementation, and required extra resources in the cluster.

After several attempts in debugging the session heartbeats this solution was also discarded as slow and too complicated.

4. **Make changes in hardware to support routing Y-link first, instead of X-link first**
   This problem could easily have been solved if the existing SCI hardware was able to support two addresses per node, allowing one of the addresses to route X-dimension first, and the other address to route Y-dimension first. If the nodes now encountered a problem the driver would have changed the destination address on the packets, and the nodes would have handled them differently.

   In the spring of 2006 Dolphin Interconnect Solutions started the development of the p2s project. p2s is a new SCI chip, integrating one PSB66 and one Link Controller 3 in one chip to lower the production cost on SCI hardware. This chip is planed to be complete by August 2006.

   Because of feedback from this thesis, Dolphin has implemented support for changing the destination addresses in the new hardware. This involved extending the routing tables (Routing RAM) from 2^8 entries (256 nodes) to 2^{13} entries (8192 nodes). The 14^{th} bit in the Routing RAM is reserved to indicate which address the node has. If bit 14 is 0, then we route X-first, and if the bit is set to 1, we route Y-first.

This hardware change is unfortunately not ready before after this thesis is handed in, but this feature is simulated, and working.

5. **Probe the downstream node with the probe mechanism**

   This solution is similar to the session heartbeat mechanism, but instead, we use the low level probe mechanism, and ask the probe to return the cable status. The probe mechanism is available just after the link controller is set up with routing tables, and does not give the delay we have if all sessions must be enabled first. We also have the advantage here that we do not need sessions to all nodes in the cluster. Probe works directly on the PAL-layer or the driver.

   Every node will now probe the upstream node on the ring, and ask for cable status from the same part of the remote nodes IO_DATA register as the session mechanism was supposed to check. If it detects a broken Y-link it will store this information in a broken link list. This list has 256 entries, and holds information about remote nodes' cable status. It is only the remote Y-links cable that we are interested in, since these are the only links that are giving us some issues.

   If we find a broken Y-link we store the information in the array, and request a reset to the driver. This reset will, as described earlier, trigger a rerouting. If no broken Y-links are detected, the driver will update the broken list with "cable ok" and continue. The next step in the procedure is to enable sessions and communication.

   Since the problem we detected applies to all nodes on the broken Y-link, we use a function in the driver to generate a vector with nodes on the broken Y-link. The result will be returned in an array.

   This solution is implemented in the driver, it is tested and it is working. This was also the solution we choose the overcome the problem we encountered.

After the probe method was implemented in the driver, we were able to support broken Y-links anywhere in the cluster.

If our downstream node has a broken Y-link, we add the following general following algorithm to the existing rules:

- If the node is on our own X-dimension (SCI-Link 0), send directly to the node.
- If our own SCI-Link 1 is operational, and destination node is on the affected Y-link, send packet out on SCI-Link 1 (Y-dimension).

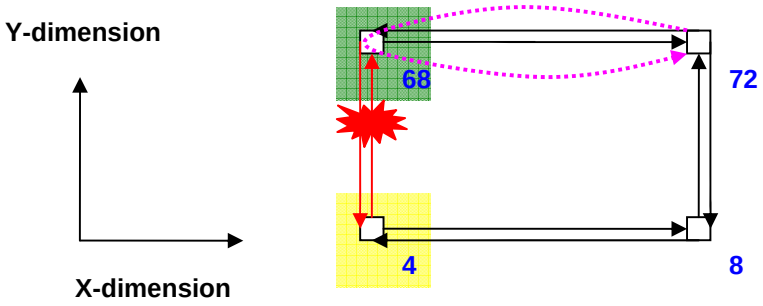

Figure 23: Solution to broken Y-link problem

In Figure 23 the Y-link between node 4 and 68 has just broken down. Let's place ourselves in the position of node 72, and see how the driver handles rerouting.

Node 72 will start the process of rerouting when it detects the loss of synchronization signal on the X-link because of node 68 doing a reset. Node 72 will detect that both its links are operational, and it will probe (dotted pink arrow) the upstream node, which is node 68 in this case. The probe will tell that node 68 has a broken Y-link. Node 72 will detect that 68 (marked green) is on its own X-dimension, so packets to node 68 will go normally on the X-link. It will also calculate that node 4 (marked yellow) is on node 68s Y-vector, so packets to node 4 will be sent out on node 72s Y-link, thus traveling to node 8, before being sent to node 4.

5.3 Test tools

To test the efficiency of our new high availability routing method several tools were needed. Programs that measured bandwidth and latency were already developed and standard in the Dolphin driver. This program is called *scibench2* and will be discussed in more detail in chapter 6.

The tool that we needed to develop was a tool to measure the time the link is down on the system. The tool is developed using the SISCI API, and written in the C programming language. Two options were available, either SISCI or SCI SOCKET. SISCI was chosen, because it is the lowest level API. The tool has to be started as server on one machine, and client on another machine, and is named *downtime*. The output to the user is the time the connection is down in milliseconds.

Downtime checked in as a part of Dolphins SCI driver, and is located in the: *DIS\src\SISCI\cmd\test\downtime* directory of the driver, and have been tested on Windows and Linux.

5.3.1 Downtime

The tool downtime is developed on the SISCI API, and it begins executing after the user has set up a server and a client. The program will allocate memory, and set up memory segments before it starts a transfer from the client to the server. Instead of ending the session after the transfer is complete, the program enters a loop. With this tool, we can determine how efficient the HA implementation is. It measure the time elapsed between a cable or machine problem is detected until the communication is enabled. Downtime has the following flowchart:

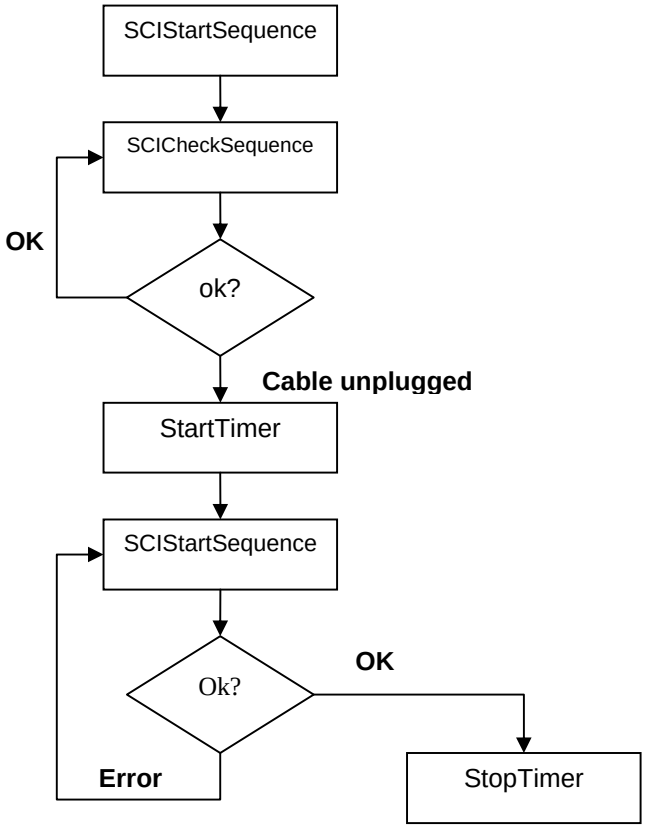

Figure 24: Program flow for downtime measurement

Figure 24 show the program flow during the measurement. We begin the test with the SISCI function SCIStartSequence. This function initializes a communication sequence. After it returns, we enter an eternal while loop, and call SCICheckSequence. This function checks with the driver, if everything is operational. If SCICheckSequence returns an OK, we tell the system to sleep for a millisecond, to give away some CPU time. Then we unplug the cable. If SCICheckSequence returns error, the function StartTimer is called. SCICheckSequence is a function in SISCI for error checking. It checks the transferred sequence, and returns an error if it is not complete. This function measures the numbers of CPU-ticks, and uses the clock frequency to calculate milliseconds.

The timer is now running, and we call SCIStartSequence again, until it returns OK. When it returns ok, the driver has initialized communication again, and the timer is stopped. If the program is configured to run forever, it will just jump back to the SCICheckSequence step. If the program is configured to run once, it will break out of the loop, and before it tells the user how long the program has been running without an error.

6 Evaluation

In this chapter we are to do several tests on the new routing algorithm. The tests will be done on real hardware, and we will introduce errors in the form of a broken cable, while measuring parameters like bandwidth, latency and downtime on the communication. When the word error is used we refer to the cable out event that requires a rerouting.

The tool for measuring bandwidth and latency is called *scibench2*, developed by Dolphin Interconnect Solutions, and is available open source as a part of Dolphin's SCI driver. Scibench2 uses the SISCI API to interface with SCI. For measuring communication loss time, my own downtime tool (described in the chapter 5) is used.

```
1: scibench2 -server –rn 4 –ko 1 –loops 50 –iloops 2000 –errcheck
2: scibench2 -client –rn 8 –ko 1 –loops 50 –iloops 2000 –errcheck
```

Figure 25: Example on a scibench2 commands

Scibench2 relies upon a simple client server approach. Figure 25 illustrates an example on the command used for starting a benchmark between node 4 and node 8 with scibench2 on Linux. Line 1 in Figure 25 is the command to start the server on node 8, we can see the parameters are "server" and "-rn 4", this tells the node to act as a server and the remote node is node 4. Line 2 is started on node 4. It acts as the client, and the remote node is node 8.

The client sends 100000 packets to the server, using different sizes. Sizes available are from 4 bytes to 64kB. We use 64kB to measure bandwidth. Latency is measured with the 4 bytes packets. The benchmark program uses a timer to check the time it takes to transfer all 100000 requests, and do a simple division with time/number of packets. The latency result is divided by two, because we are interested in one way latency, not both ways. We have also configured scibench2 to check for errors during the transmission. This option is call "*errcheck*" and is done with a SISCI function, and makes sure that the application pauses while the cluster is doing reconfiguration. The other parameters we tune are "*ko*". This is the key offset in the memory, and this option is made to allow more

than one instance of the benchmark to be executed at once. The two last parameters are *"loop"* and *"iloop"*. The first, loop, tells us how many results we want printed out, and iloops is how many times we want the 100000 requests sent before printing out the result (loop).

A good extension of this testing would include tests on broken Y-cables. However, due to the fact that support for Y-cables was finalized only a month before the deadline of this thesis, the time available would not have been sufficient.

All the tests are started independently on each machine, and they are synchronized on the point where the error is detected. This point is marked with a red line in all graphs. As a result it can be hard to compare measurements from different sessions directly with each other, except around the occurrence of the error.

6.1 Test setup

In the test a 4 node cluster is used. This cluster is connected together in a 2D torus topology. The machines have the same specifications in hardware. The operating system on all the machines is CentOS 4.2, based upon Linux kernel 2.6.9-11.EL compiled for i686, CentOS is a free re-built version of Red Hat Enterprise Linux, and uses the Linux kernel from Red Hat.

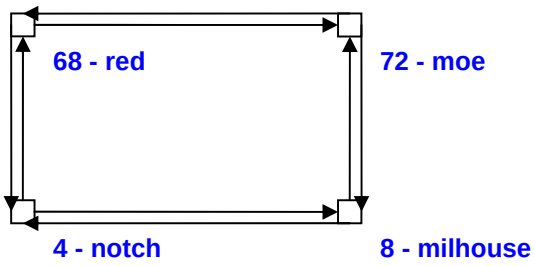

Figure 26: Test cluster with node ID and name

Hardware on test cluster:

- AMD Athlon XP 2000+ Processor, (1667 MHz), 256kB Level 2 cache.
- 512 MB RAM (266 MHz DDR SDRAM with ECC).

- AMD 760-MPX Chipset and AMD-768 I/O Bridge.
- 80GB Western Digital hard drive, 7200rpm, ATA100 interface.
- Dolphin Interconnect Solutions PCI 2D SCI Adapter D334 (PSB66).

All tests on this cluster were done with Dolphin's SCI driver version 3.1.7.1 checked out from CVS and compiled on the 3rd of June 2006.

6.1.1 Test 1 – 4 nodes and one session

The first test is a basic test of a broken X-link between two nodes. We are having one communicating session between node 4 and node 8 (Figure 27, green dotted line). The test lasts for about 50 seconds, and after 30 seconds we remove the X-link cable between node 4 and 8.

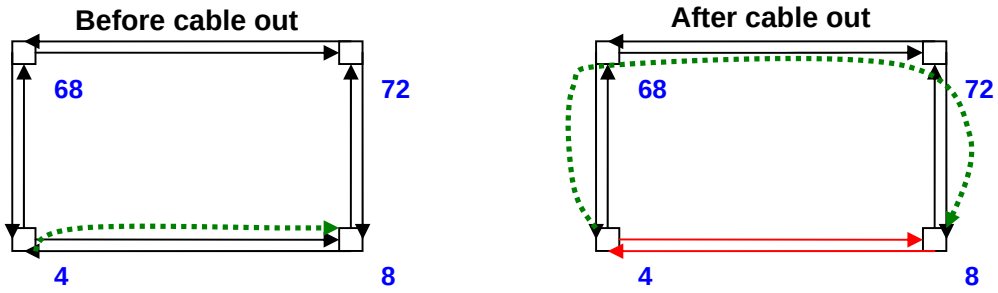

Figure 27: 4 nodes, one session

Parameters for scibench2:

```
scibench2 -(server/client) -rn (4/8) -ko 1 -loops 50 -iloops 2000 -errcheck
```

Path for the request and response before error:	
4 → 8	8 → 4
Path for the request and response after the error:	
4 → 68 → 72 → 8	8 → 72 → 68 → 8

Before the error there is only one link to travel. Theoretically this takes 140 nanoseconds (PSB to SCI link takes 70 nanoseconds, and we do it two times). After the error we have two extra B-link jumps, so the theoretical time it takes is 740 nanoseconds, a 600 (one B-link jump takes 300 nanoseconds) nanosecond increase compared with before the error.

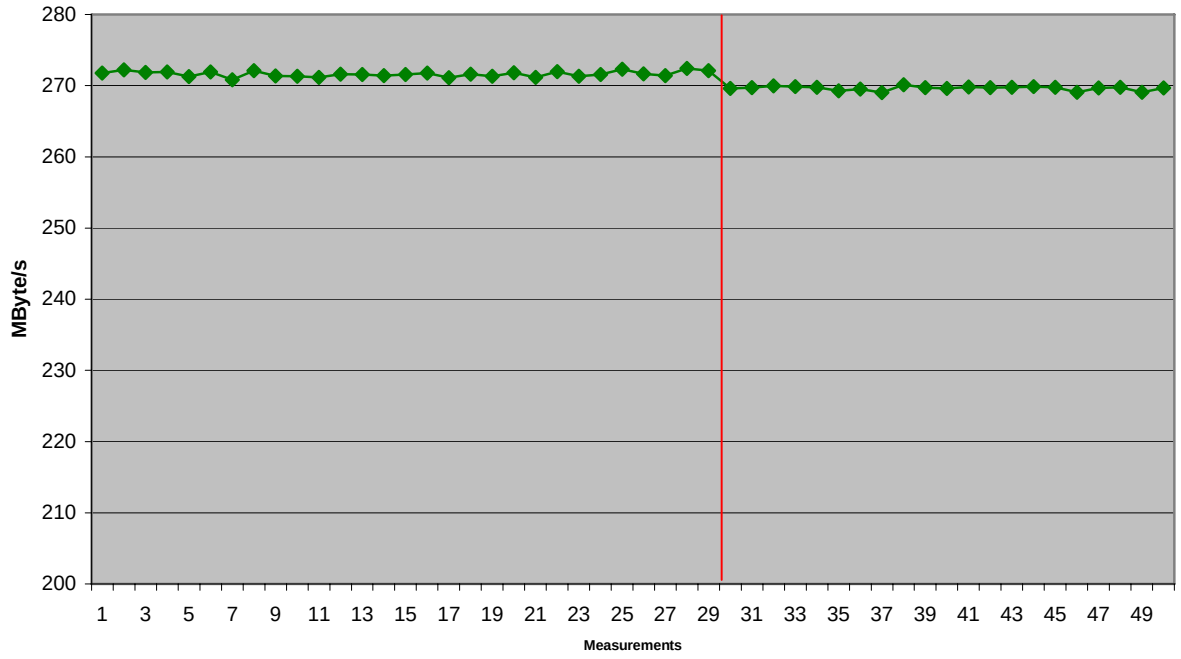

Figure 28: 4 Nodes, one session - Bandwidth

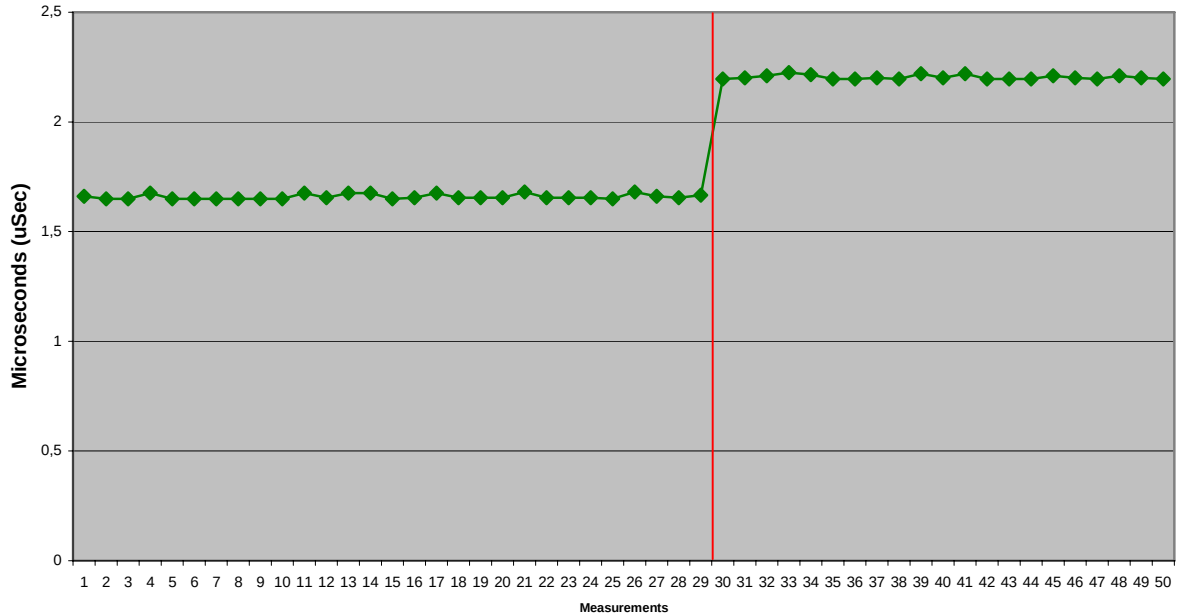

Figure 29: 4 Nodes, 1 session – Latency

Figure 30 is a printout from the Linux command line. The downtime tool prints the result out in text. These printouts from the Linux command line will be used to present the communication downtime during the remainder of this thesis.

```
[root@notch ~]# ./downtime -client -rn 8 -ko 2
 ./downtime (HKS-edition) compiled Jun  1 2006 : 12:36:54

- Sessions OK: Starting test -
Sessions NOT OK: Problem detected
- Sessions OK: Downtime = 189.24 ms
```

Figure 30: 4 Nodes, one session - Communication downtime

Figure 28, Figure 29 and Figure 30 are the results from the first test on the cluster with one session running between node 4 and node 8, and a broken cable on the X-link between the two nodes. The read line in Figure 28 and Figure 29 indicates when the error occurred in the test.

The first figure from this test is Figure 28. This test shows the bandwidth. In this test, we only have one session, and that is node 4 sending data to node 8 and since SCI is a request, response protocol there will be traffic in both directions. The limitation on this test of around 270 Mbytes per second is limited by the PCI-bridge on the host computers. SCI bandwidth is very sensitive to how long time the host chipset uses to prepare a PCI transfer, and how big bursts are available. A PCI burst transfer is defined by one PCI bus transaction, and it includes both the transaction setup, and the data transfer. The error occurs right before measurement 29. We can see that the bandwidth on the transfer drops down by around 3 Mbytes per second. This can be due to the longer transfer, as seen in Figure 27. The traffic must now go via node 68 and node 72, before reaching its target on node 8.

The next test, Figure 29, is latency. We measure one-way latency. In the figure we can see an increase in latency of around 0.6 milliseconds after the error is resolved. This is a

larger drop in performance than with bandwidth, but it can be explained with the longer way the packets have to travel. The direct path between node 4 and node 8 involves sending the packets directly out from the PSB, via the B-link and out on the link controller, and directly to node 8. As seen on Figure 27, the path after the cable is out have two extra B-link jumps on node 68 and node 72. A jump on the B-link costs between 200 and 300 nanoseconds, compared with a jump directly from the PSB through the FIFO, which costs 70 nanoseconds. The increase of 0.6 milliseconds (600 nanoseconds) on the latency fits well with the theoretical number of two extra B-link jumps.

The last test is the communication downtime test, seen in Figure 30. This test was done with 50 milliseconds ReadyToGo timer, and 30 milliseconds Fatal-timer. It took 189 milliseconds from the nodes detected the cable-out problem, until communication resumed to normal. There is always some delay before the driver detects the fatal interrupts, but the value of 189 milliseconds might indicate that we had to go through the fatal-loop two times. A test will be done later on the ReadyToGo timer, to check how low values on the timer will affect the time it takes to solve the problem. As described in the implementation chapter, a to too low timer value can cause us to attempt continuation before everything is prepared. This will force the driver to do another loop through ReadyToGo, fatal, or in the worst case, both.

6.1.2 Test 2 – 4 nodes and two sessions

In this test, we have as shown in Figure 31, two sessions in the cluster. One session is between node 4 and node 72 (green dotted line) and the other session is between node 8 and 68 (blue dotted line). The test lasts 50 seconds, and after 30 seconds, the cable between node 4 and 8 is removed.

One of the differences between test 1 and this test is that we are not giving the packets longer way to travel. Number of links to travel, and number of B-link jumps are the same

both before cable out, and after cable is removed. The only difference is that requests and responses uses the same path.

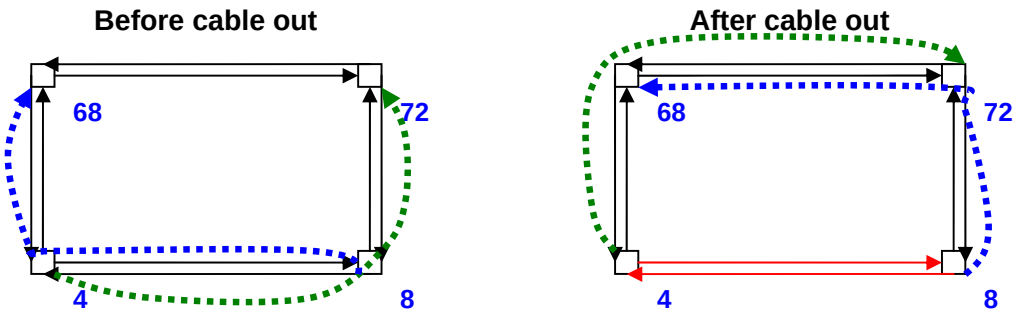

Figure 31: 4 nodes, two sessions

Parameters for scibench2:

Node 4 / 72:

```
scibench2 -(client/server) -rn (4/72) -ko 1 -loops 50 -iloops 2000 -errcheck
```

Node 8 / 68:

```
scibench2 -(client/server) -rn (8/68) -ko 2 -loops 50 -iloops 2000 -errcheck
```

Node 4 / 72:

Path for the request and response before error:	
4 → 8 → 72	72 → 68 → 4
Path for the request and response after the error:	
4 → 68 → 72	72 → 68 → 4

Node 8 / 68:

Path for the request and response before error:	
8 → 4 → 68	68 → 72 → 8
Path for the request and response after the error:	
8 → 72 → 68	68 → 72 → 8

The communication distance before and after the error in this test is still the same. One B-link jump is needed. The change in this test is that the request and response share the same path. Before the error, the response used another path. After the error, we can see that both responses and requests will use the link between node 68 and node 72 in this test. SCI response packets are only 8 bytes, but they are still taking up a place in the send queue.

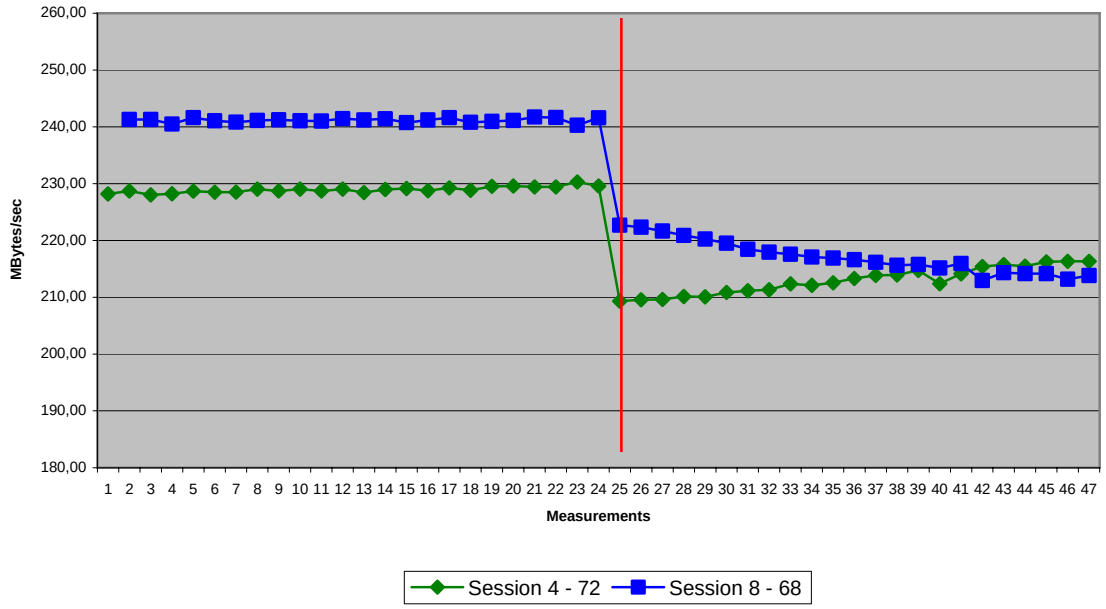

Figure 32: 4 nodes, two sessions - Bandwidth per session

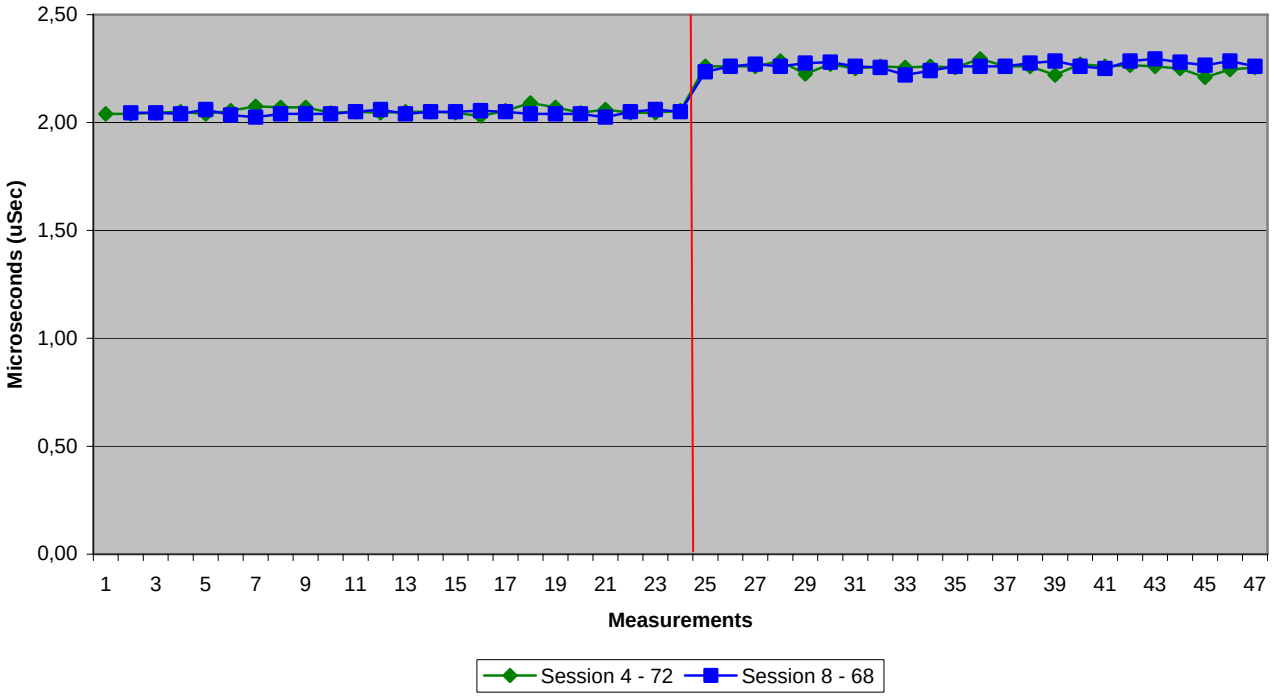

Figure 33: 4 nodes, two sessions - Latency per session

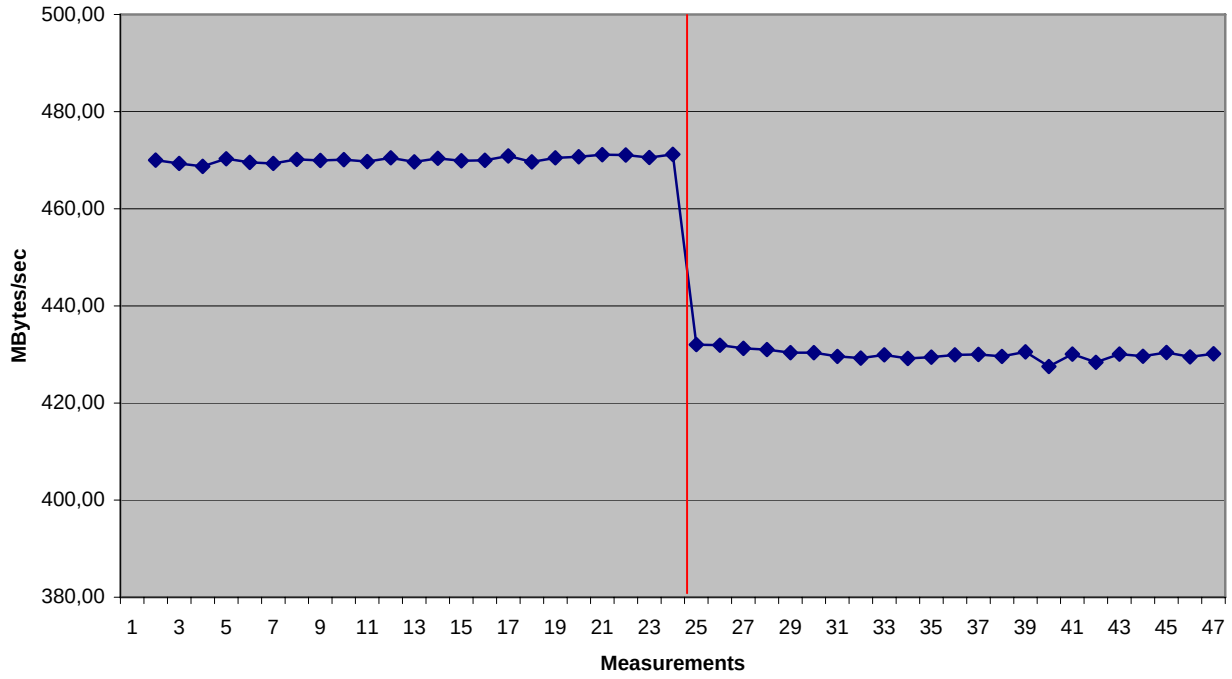

Figure 34: 4 nodes, two sessions - Total bandwidth

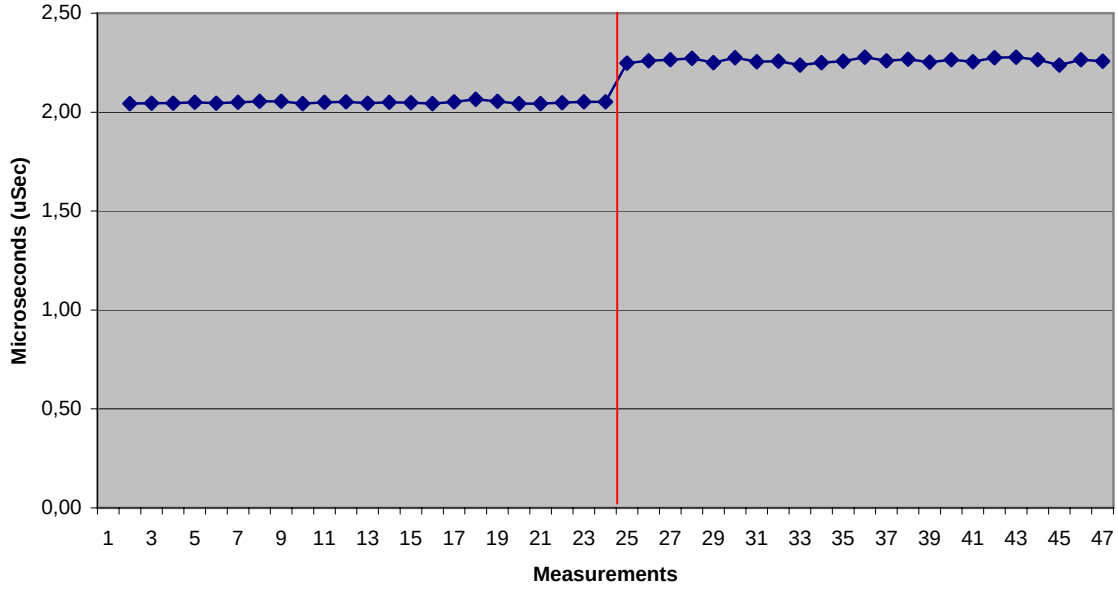

Figure 35: 4 nodes, two sessions - Average latency

NODE 4 – 72

```
[root@notch ~]# ./downtime -client -rn 72 -ko 5
 ./downtime (HKS-edition) compiled Jun  1 2006 : 12:36:54

- Sessions OK: Starting test -
Sessions NOT OK: Problem detected
- Sessions OK: Downtime = 188.32 ms
```

NODE 8 – 68

```
[root@milhouse ~]# ./downtime -client -rn 68 -ko 6
 ./downtime (HKS-edition) compiled Jun  1 2006 : 12:36:54

- Sessions OK: Starting test -
Sessions NOT OK: Problem detected
- Sessions OK: Downtime = 189.35 ms
```

Figure 36: 4 nodes, two sessions - Communication downtime

This test is different if we compare it with test 1. We now use two sessions instead of one, and the test is set up so that we have the same length along the packet paths both before and after the broken cable. In both cases, the packets do one B-link jump and have to travel two links. There are also two sessions competing for bandwidth and resources this time.

Figure 32 is a plot of the bandwidth for both sessions separately. When the error occurs, both sessions loose about 20 megabytes of bandwidth, it looks as if this might be due to more traffic on the X-link between node 68 and 72. It is important to remember that SCI is a request and response protocol, and for every packet we send out, a response is coming back. Under normal conditions, with all cables in place, and an optimal routing calculated by Graph Routing is in place, the traffic is evenly distributed through the 2D mesh. Both request and response messages are routed X-dimension first, then Y-

dimension. We are going to investigate this with an extra test with only one session between node 4 and 72 to see if it is competing traffic from the session between node 8 and 68 that decreases the performance. If we look at Figure 34, which is the total bandwidth in the cluster, we can also see that the total performance drops about 40 megabytes per second after the cable is out. We can also see in Figure 32 that the bandwidth on the two sessions evens out after the cable is removed. The total bandwidth in Figure 34 is almost constant after the broken cable.

If we look at the latency test in Figure 33 and Figure 35, we can also see the same trend. Latency is increasing. Since both before and after cable out has the same numbers of links to travel, and the same numbers of B-link jumps, the increase in latency is not expected, but it might be explained with more competing traffic. The extra test between node 4 and node 8 will reveal that the increase in latency is due to competing traffic.

The communication downtime values in Figure 36, for both sessions look like test 1. The values are around 190 milliseconds, which might indicate that the driver had to loop through fatal or ReadyToGo two times. The driver is still running the same parameters that were used in test 1. Fatal timer is set to 30 milliseconds, and ReadyToGo timer is set to 50 milliseconds.

Now we are going to do an extra test, with the same parameters that we used in the test 2, to investigate if the loss of bandwidth in test 2 was due to competing traffic. This time we are going to remove the session between node 8 and node 68. The downtime application is not used in this test, only scibench2 to check bandwidth and latency. The test will use the same parameters as in test 2.

Node 4 / 72:

Path for the request and response before error:	
$4 \rightarrow 8 \rightarrow 72$	$72 \rightarrow 68 \rightarrow 4$
Path for the request and response after the error:	
$4 \rightarrow 68 \rightarrow 72$	$72 \rightarrow 68 \rightarrow 4$

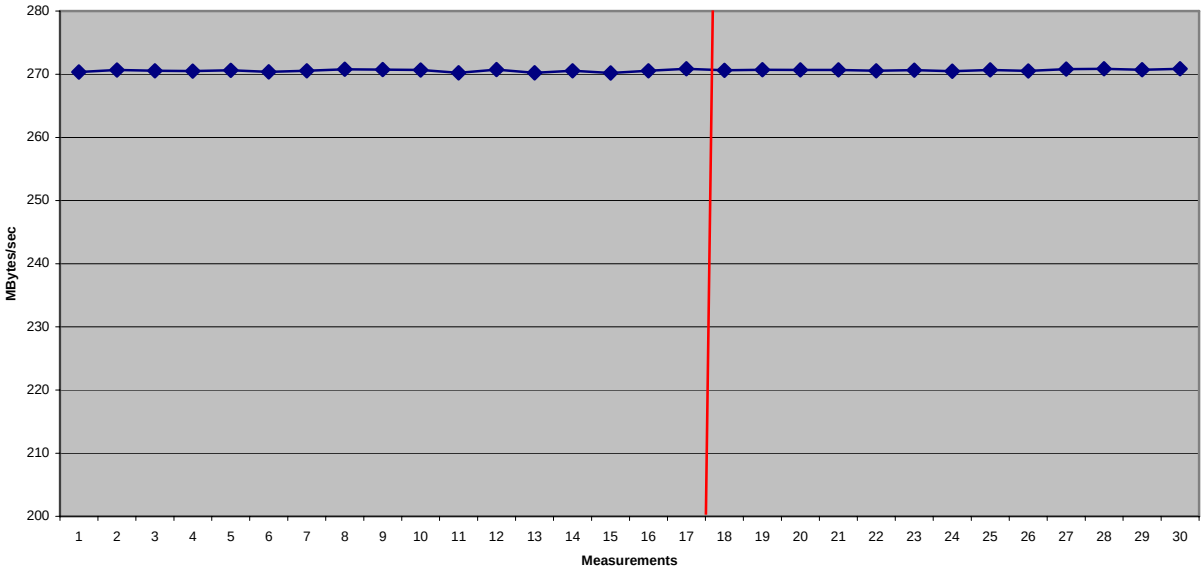

Figure 37: Extra test, only one session – Bandwidth

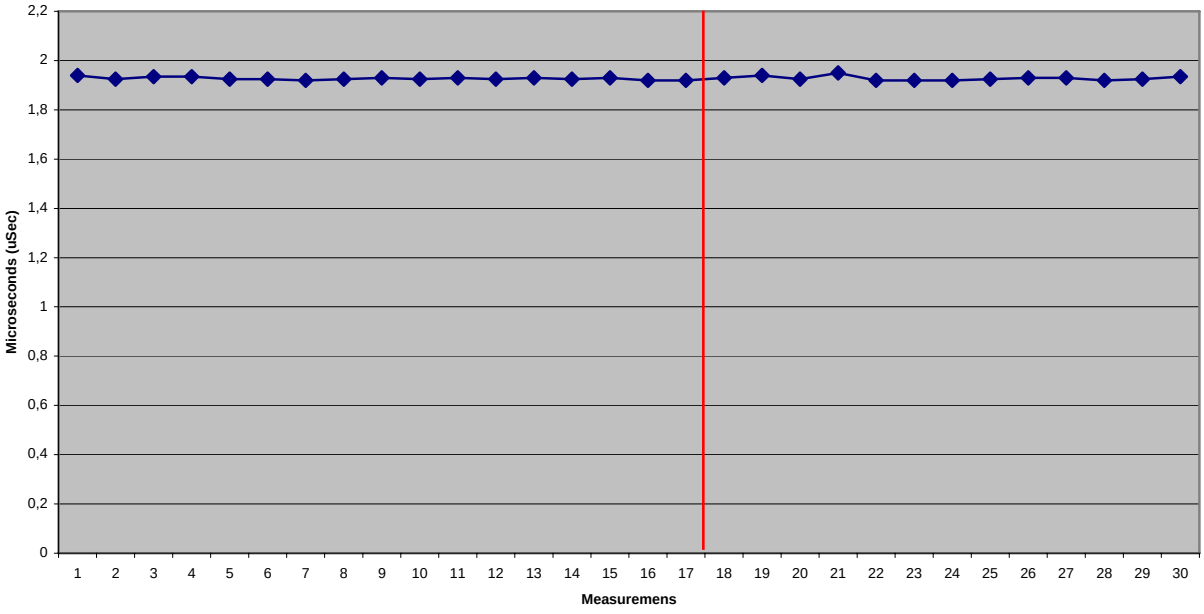

Figure 38: Extra test, only one session – Latency

Figure 37 and Figure 38 shows the results from the extra test. We only run one session between node 4 and node 72. We can see both in the bandwidth test and the latency test that the number does not change after the cable was removed. The last 20 results were

removed, since they did not have any relevance for this test. When we look at the latency and bandwidth after the cable is out, we can clearly observe that our approach has no penalty to the performance if the packet does not have a longer way to travel or more B-link jumps in its path.

With this data we can also conclude that the performance drop seen with two sessions in the first part of test 2 is because of the competing traffic on the X-link between node 68 and node 72. We can also conclude that even though this test sends requests and responses in the same path after the error, we do not se any degradation in performance. One session with requests and responses does not manage to use all the resources available on the SCI adapter.

6.1.3 Test 3 – 4 nodes, worst case

In this test we will try to put more load on the network. We are going to use 4 sessions in the network, and two of them to send both ways. This test ran much longer than the earlier test, and the runtime is now 5 minutes. After 3 minutes, the cable between node 4 and node 8 was removed. This test will be a combination of test 1 and test 2 with even more load added.

The parameters for this test only call for 30 measurement points (loops), but we increased the number of transfers before printing out (iloops) to 5000 to compensate for this. It is also important to notice that the measurements are synchronized around the point where the error is detected, marked with a red line.

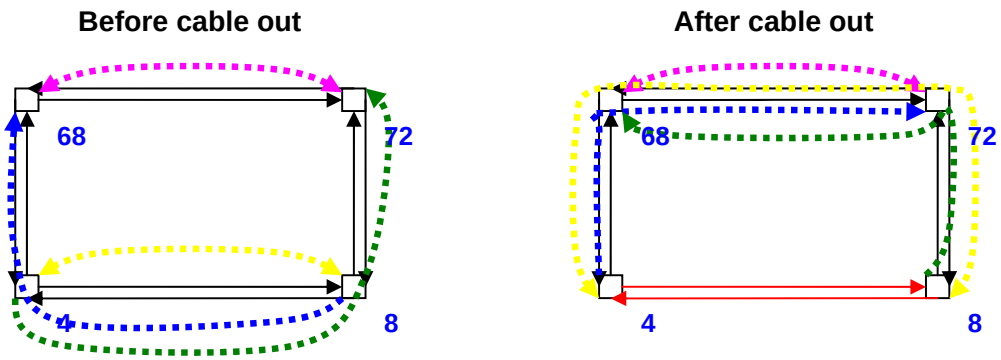

Figure 39: 4 Nodes, worst case

Parameters for scibench2:

Node 4 / 8:

```
scibench2 -(client/server) -rn (4/8) -ko 7 -loops 30 -iloops 5000 -errcheck
```

Node 8 / 4:

```
scibench2 -(client/server) -rn (8/4) -ko 8 -loops 30 -iloops 5000 -errcheck
```

Node 4 / 72:

```
scibench2 -(client/server) -rn (4/72) -ko 1 -loops 30 -iloops 5000 -errcheck
```

Node 8 / 68:

```
scibench2 -(client/server) -rn (8/68) -ko 2 -loops 30 -iloops 5000 -errcheck
```

Node 68 / 72:

```
scibench2 -(client/server) -rn (68/72) -ko 3 -loops 30 -iloops 5000 -errcheck
```

Node 72 / 68:

```
scibench2 -(client/server) -rn (72/68) -ko 4 -loops 30 -iloops 5000 -errcheck
```

Node 4 / 8:

Path for the request and response before error:	
4 → 8	8 → 4
Path for the request and response after the error:	
4 → 68 → 72 → 8	8 → 72 → 68 → 4

Node 8 / 4:

Path for the request and response before error:	
8 → 4	4 → 8
Path for the request and response after the error:	
8 → 72 → 68 → 4	4 → 68 → 72 → 8

Node 4 / 72:

Path for the request and response before error:	
4 → 8 → 72	72 → 68 → 4
Path for the request and response after the error:	
4 → 68 → 72	72 → 68 → 4

Node 8 / 68:

Path for the request and response before error:	
8 → 4 → 68	68 → 72 → 8
Path for the request and response after the error:	
8 → 72 → 68	68 → 72 → 8

Node 68 / 72:

Path for the request and response before error:	
68 → 72	72 → 68
Path for the request and response after the error:	
68 → 72	72 → 68

Node 72 / 68:

Path for the request and response before error:	
72 → 68	68 → 72
Path for the request and response after the error:	
72 → 68	68 → 72

This test is a combination of test 1 and test 2, with the addition of a two-way session between node 68 and 72. The session between node 4 and node 8 is also two-way. The sessions between node 68 and 72 is the only sessions that have the same path both before and after the error. The amount of competing traffic will however be dramatically increased.

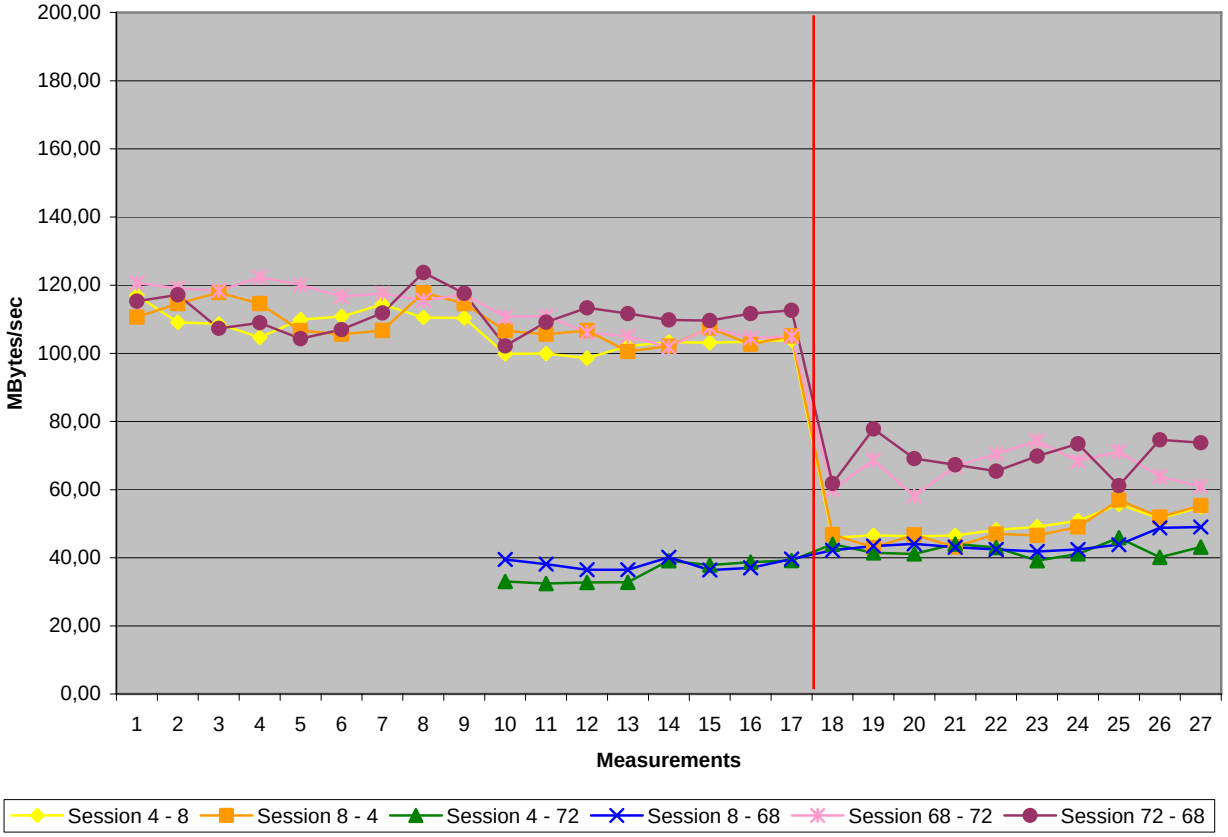

Figure 40: 4 Nodes, worst case - Bandwidth per session

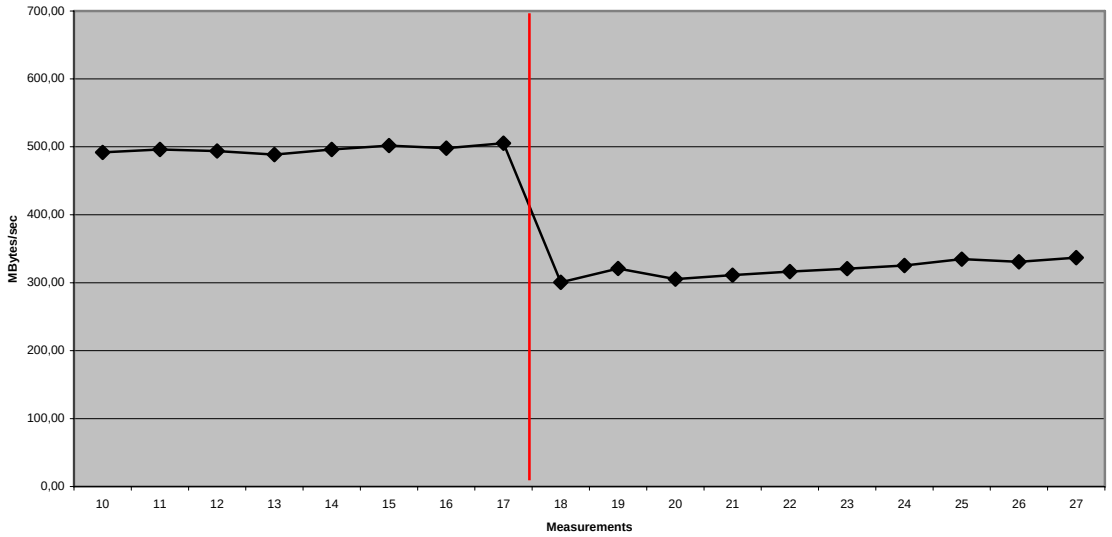

Figure 41: 4 Nodes, worst-case - Total bandwidth

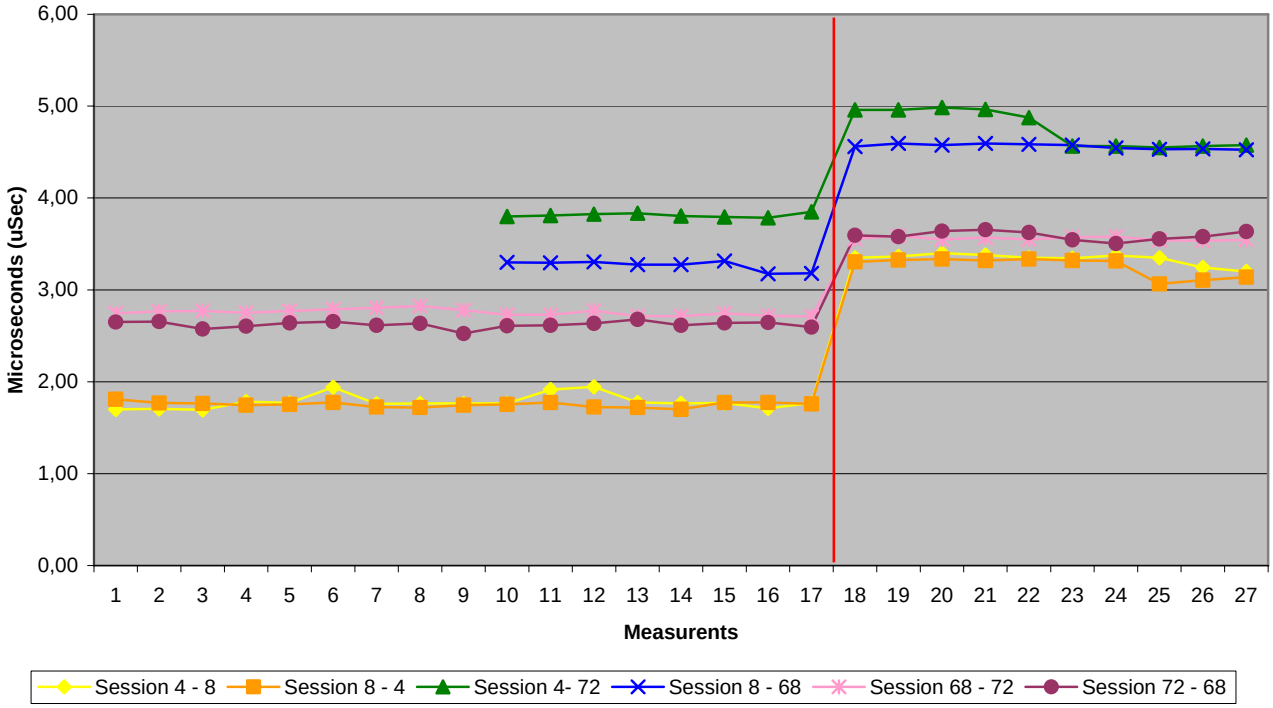

Figure 42: 4 Nodes, worst-case - Latency per session

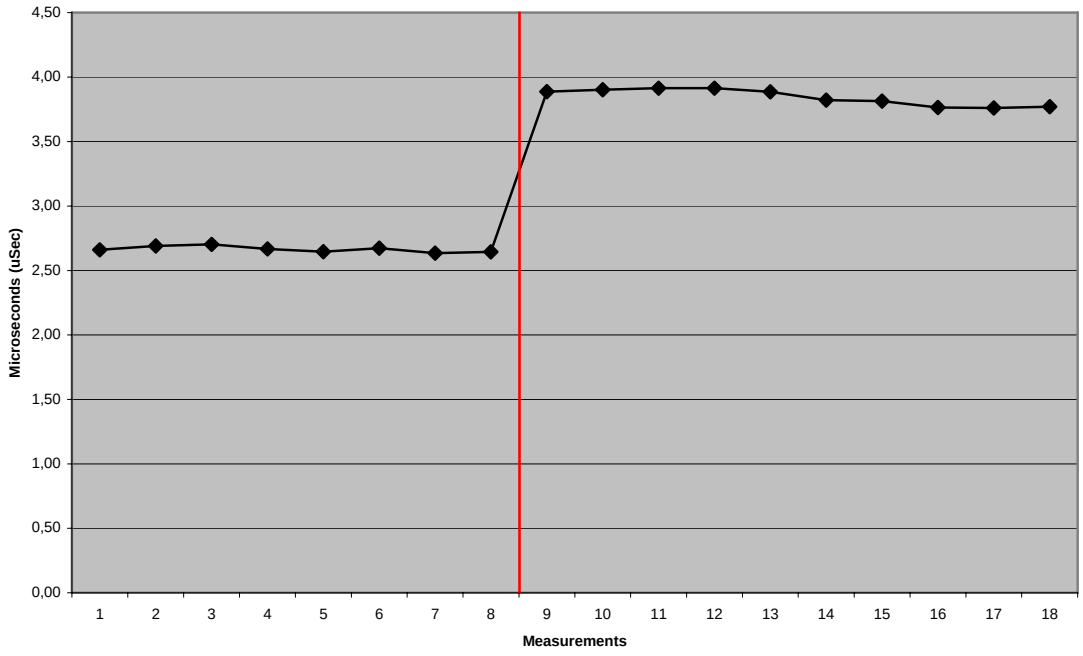

Figure 43: 4 Nodes, worst-case - Average latency

NODE 4 – 8

```
[root@notch ~]# ./downtime -client -rn 8 -ko 9

 ./downtime (HKS-edition) compiled Jun  1 2006 : 12:36:54


- Sessions OK: Starting test -

Sessions NOT OK: Problem detected

- Sessions OK: Downtime = 320.27 ms
```

NODE 68 – 72

```
[root@red ~]# ./downtime -client -rn 72 -ko 10

 ./downtime (HKS-edition) compiled Jun  1 2006 : 12:36:54


- Sessions OK: Starting test -

Sessions NOT OK: Problem detected

- Sessions OK: Downtime = 187.54 ms
```

NODE 4 – 72

```
[root@notch ~]# ./downtime -client -rn 72 -ko 5

 ./downtime (HKS-edition) compiled Jun  1 2006 : 12:36:54


- Sessions OK: Starting test -

Sessions NOT OK: Problem detected

- Sessions OK: Downtime = 185.16 ms
```

NODE 8 – 68

```
[root@milhouse ~]# ./downtime -client -rn 68 -ko 6

 ./downtime (HKS-edition) compiled Jun  1 2006 : 12:36:54


- Sessions OK: Starting test -

Sessions NOT OK: Problem detected

- Sessions OK: Downtime = 320.68 ms
```

Figure 44: 4 Nodes, worst-case - Communication downtime

Figure 40 is the first test on the worst-case scenario, and we test bandwidth per session first. We can see that the sessions that only have to travel over one link (4 to 8, 8 to 4, 68 to 72 and 72 to 68) are the links with the highest bandwidth. The two sessions that go from 4 to 72 and 8 to 68 have a much lower bandwidth. We suspect that is because of heavy competition with the other sessions, and the fact that they have to do a B-link jump. This might be one of the factors that limit the performance in this case.

After the error occurs we can see a bandwidth drop. The sessions between 4 and 72 and 8 and 68 takes almost no performance loss. The sessions between 68 and 72 looses performance, probably due to competing traffic on the link. The sessions between node 4 and node 8 takes the biggest loss of performance. They have to travel three times the distance, and do two B-link jumps, while competing with all the other sessions for bandwidth.

If we look at the latency test per session in Figure 42 we can see the same trend. The sessions between node 68 and 72 only get a penalty of 0.8 microseconds after the error, this is probably due to more traffic on the link, and new sessions to share the bandwidth with. The sessions between node 4 and 72 and 8 and 68 get an increase of around 1.1 microseconds. They still have to do a B-link jump, but the sessions between 4 and 8 also have to do this B-link jump, twice. Therefore we can also see that these sessions have the highest increase in latency. Their increase is around 1.6 microseconds. One B-link jump takes about 300 nanoseconds.

The two graphs (Figure 41 and Figure 43) that show average latency and total bandwidth does not show any surprises, both are even before and after the error. In the total bandwidth case, we lose around 200 megabytes per second when the link fails, and the average latency increases with around 1.5 microseconds.

The communication downtime shown in Figure 44 shows no surprises either. The sessions between 4 and 8 showed 320 milliseconds downtime, compared with 187 milliseconds for the other sessions. This might be due to the fact that node 4 and 8 have

to rely upon node 68 and node 72 to communicate, and both these nodes have to be ready first, if not the driver will get an error. The long downtime of 320 milliseconds suggest that these nodes had to go at least two extra times through the fatal and ReadyToGo handler.

The error will force all communication in the cluster to use the X-link between node 68 and 72. Before the error, the link was used by the traffic between nodes 68 and node 72. All the traffic on the X-link between node 68 and 72 have to travel through the B-link on node 68 and 72.

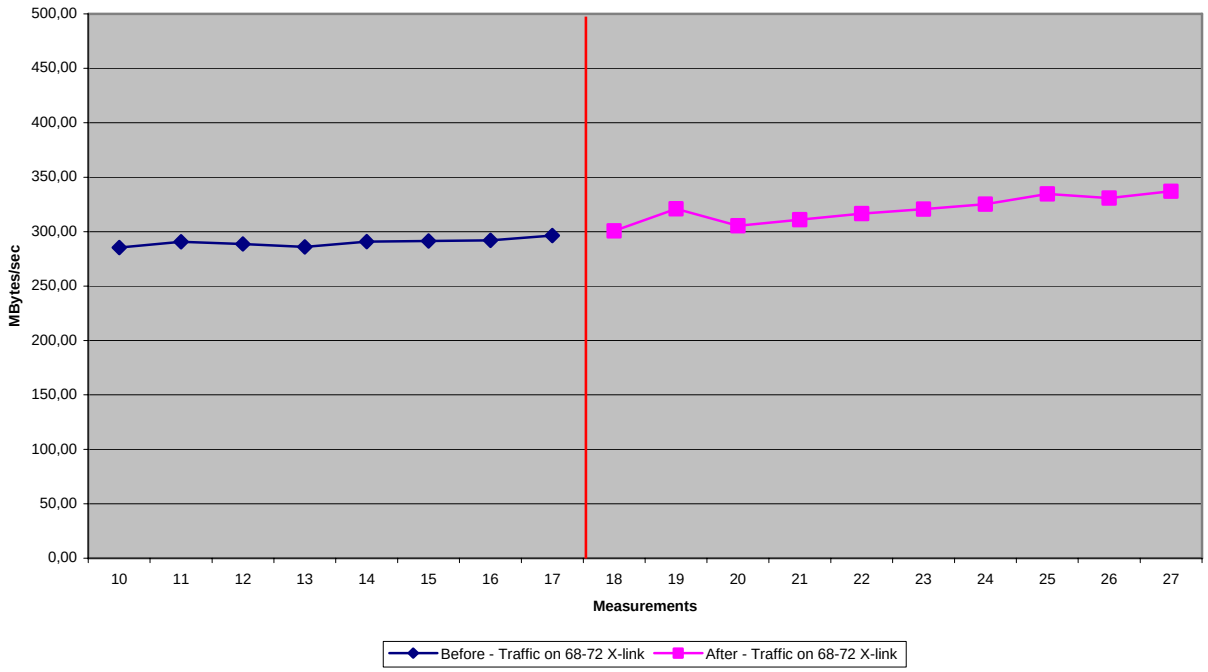

Figure 45: 4 Nodes, Traffic on the X-link between node 68 and 72

In Figure 45 we can see the total bandwidth of the sessions using B-link on node 68 and 72. Before the error there is less than 300 megabytes per second, and after the error it increases slowly from 300 megabytes up to almost 350 megabytes. The B-link is a bus between the PSB and the link controllers. The maximum theoretical bandwidth on the B-link is 640 megabytes per second, however these tests do not measure the bandwidth required by the response packets. Some of the limitations of the B-link are due to the fact

that the B-link is a bus, and can not be used by two communicating link controllers at the same time.

The limitations on the B-link are also one of the reasons to make the Graph Routing algorithm to do as few as possible jumps on the B-link. In a normal case for a 2D topology the Graph Routing algorithm will only do one B-link jump.

The B-link limitation can also be a problem in a switch, where up to eight link controllers are connected together over a B-link. This was one of the motivations for Dolphin to develop the Bx-bar in the current generation of link controllers (LC3). The Bx-bar is as described in the interconnect chapter a crossbar between the link controllers. Bx-bar is not used on SCI cards today, due to lack of support in the PSB, and the requirements for a more complex PCB. Bx-bar is implemented in the latest generation of Dolphin switches called D535.

6.1.4 Test 4 – Different timer values on ReadyToGo

The last test we choose to do was a test on how the system would behave with different values on the ReadyToGo timer. As described in the implementation chapter, ReadyToGo is a timer that starts after the fatal error handler is completed, and runs while the system is calculating routing tables and setting up link controllers. The timer is used so that the system does not get stuck on a step in the setup process.

During tests we have suspected that some of the higher communication downtime values are due to the fact that the system thinks the link controllers are ready, before they are finished with the configuration. All the tests were run with a ReadyToGo timer of 50 milliseconds. We are now going to use values 10, 20, 50, 100 and 250 to check the downtime values we get. This test is done by setting up the downtime program between node 4 and 8, and use a similar setup used in test 1. We are going to remove the cable 10 times, and plot the minimum value, average value, and maximum value.

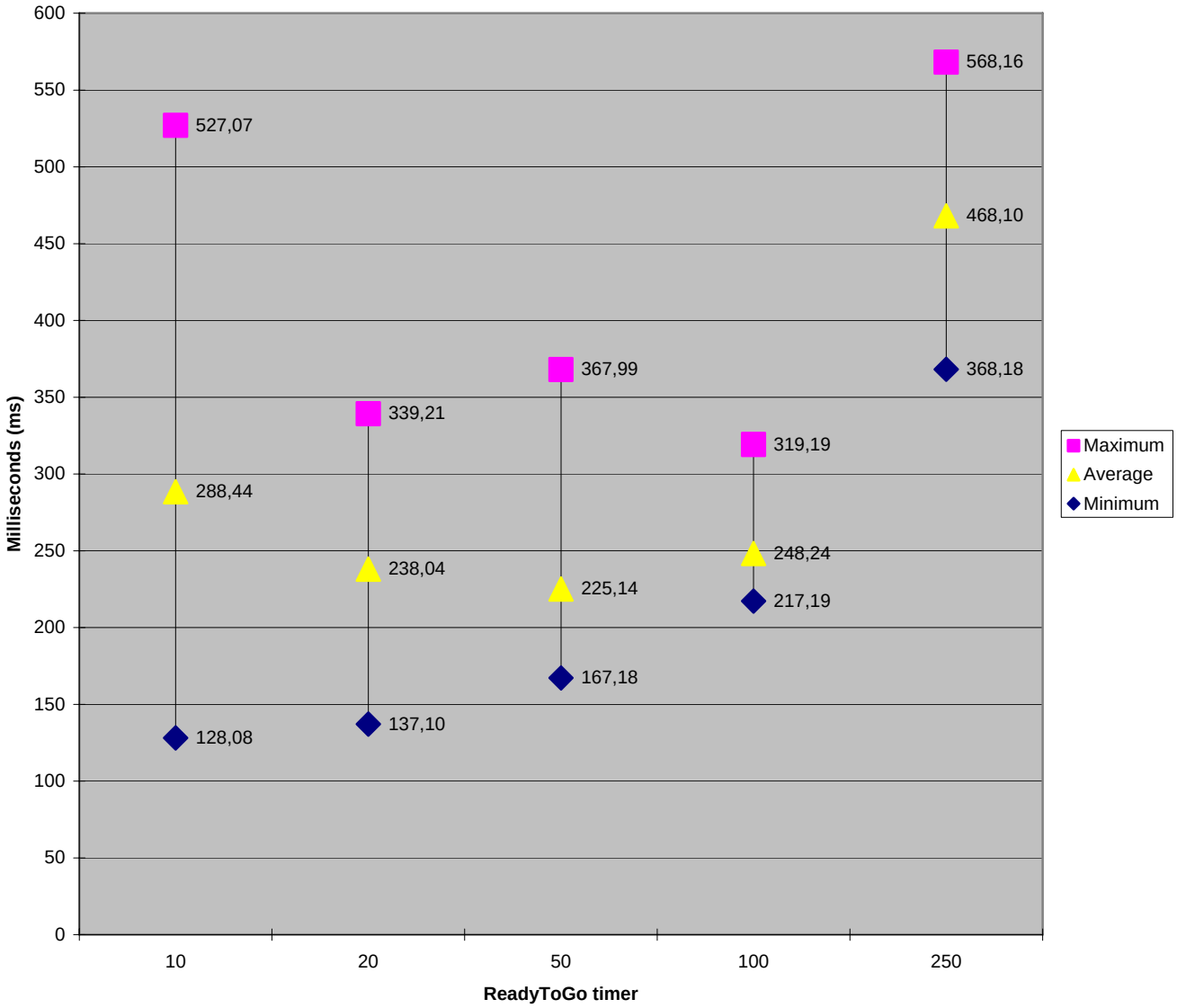

Figure 46: Downtime plot

Figure 46 confirms what we suspected during the test. Too low values on the ReadyToGo timer will affect the communication downtime in a negative way. A ReadyToGo timer value of 10 is way to low for the system, even though we are lucky a few times, and get the lowest downtime value, the average point and worst-case is high. This test indicates that 50 is the best setting if we want the lowest average value, whereas 100 have the lowest values for worst case.

6.2 Discussion

High Availability (HA) routing has not been implemented in SCI prior to this thesis, so we don't have any other solutions to compare with. Even though SCI is an old standard, it is only one company that has implemented it, so there is no competing company's solutions to compare with. The only technology to compare our HA routing with is the existing network manager in use on Dolphin's SCI technology today. Both technologies have their week points and their advantages.

One of the advantages with the network manager is its availability to see the entire network before creating a solution. High Availability routing can only see the situation from one node, and the other nodes in the network have to be ready to adapt to these changes. This can sometime lead to the fact that some spots in the SCI network become more loaded than other parts and hotspots might develop. We have seen some tendencies for hotspots in the worst-case scenario tested in test 3. These hotspots are however no issue for SCI, except that performance is decreased, since SCI has several mechanisms to prevent buffer overload and starvation. Graph Routing and the network manager have the whole picture of the cluster and can avoid having some areas with more traffic. Graph Routing and the one-dimension-first principle helps to distribute the traffic evenly in the cluster. The advantage HA routing has over manager is speed. HA routing can make decisions locally, and does not rely upon another medium (Ethernet) like the manager. If a node looses the Ethernet connection, and does not reply, the manager will automatically assume that the node is dead, and disable it, even though it works.

A possible improvement would be to integrate the HA routing option in the network manager. Then the Manager could detect that HA mode was enabled on the card, and the poll timer could for instance be increased to one poll per five minutes. Then the Manager could check the cluster and see if a rerouting had taken place, and if so, information could be collected from the nodes, and optimized routing tables could be calculated by the manager, and compared with the tables running in the cluster. If a difference is found the optimal solution could be installed in the cluster. This possible integration also has its

negative aspects. There is currently no way for the manager to know if the fix it has is an optimal solution, so we might end up doing a reconfiguration that causes the cluster more communication downtime, and no performance gained.

Another possible extension for the HA routing is to integrate a connection with the session mechanism, so that when a heartbeat in the session mechanism fails a rerouting is triggered. The session mechanism is already implemented to disable the session if a node fails to send three heartbeats in a row. This has no immediate negative consequences, and are being considered as an improvement in the next version of the HA routing algorithm. Tests carried out by Dolphin in large clusters have not shown any nodes failing to reply to heartbeats, while they still are alive.

If a cable is out, the link controller with the cable connected is disabled. This is, as described in the implementation chapter, done by masking out the interrupt registers associated with that link controller. When the link controller is disabled, the hardware has no possibility to detect if the cable is inserted again. This is why we implemented a watchdog in the driver to check cable status every minute. Several improvements are possible on this approach. If we have a cable with a bad connector that makes the card do frequent resets, it will cause the performance in the cluster to be degraded because of frequent rerouting. A possible extension here is to implement a counter that tells the system how many times this cable has been out for a period. If the counter exceeds the selected value, the system would just ignore the cable.

Another possibility is to disable the entire cable watchdog and only have the HA routing to handle the cable out event, and only use the manager to detect that the cable is back in. However this solution will also have the need for a counter that records how many times the cable recently have been out.

High availability can also be compared with redundant hardware. Dolphin’s SCI technology has the possibility to have one or more redundant links that can take over the communication if the primary link fails. The positive effect of this is that if a redundant

link fails, there will be almost no downtime at all, and there is no need to perform rerouting, since all the paths are still the same. Redundant hardware in SCI however has two weak points. The first weak point is that all the redundant links in a ring have to go through one node. If this node fails, all the rings will fail. HA routing, however, supports the loss of a node, since the remaining hardware has the possibility to reconfigure after the loss of the node. The other weak point of redundant hardware is the cost. You often pay the double price, because most equipment has to be duplicated, and you are only able to utilize half the available resources, as the duplicates are only used for backup.

SCI's competing technologies are often based upon a switch approach. If a node dies here, it only affects that node, since all nodes are connected to switches. But if a switch is lost, connectivity to all the nodes on this switch will also fail. If this switch is on the path to other switches, a reconfiguration is needed in the network. SCI also have switches available, but they are not produced anymore, due to no demand in the market, and a very high price compared with 2D and 3D torus.

Dolphin has developed another topology that is static, and offer high availability. This topology is called Direct HA. Direct HA is available both for 2D cards (with maximum three nodes) and 3D cards (with maximum 4 nodes). Direct HA is not related with the HA routing developed in this thesis, and it works on the same principle as a crossbar. All nodes are connected to all nodes. This solution is very robust, however the solution does not scale since three links is the maximum number of links implemented on SCI cards today, and this only allows for connecting four nodes.

It has also been suggested to try to optimize the reset method in the cluster. Currently, all the nodes in the cluster will do a reset if one starts to reset. This can unfortunately not be optimized farther, because of limitations in hardware. When a node does a reset, the link looses the hardware synchronization signal, since the link controller is also reset. So, when one node in the cluster does a reset, this will spread out to all nodes.

The 7th of July, Dolphin Interconnect Solutions launched HA routing as a product called Booster Kit for databases. In the first version of the booster kit, HA routing is targeted towards databases. Databases is one of the applications that that can benefit from the low downtime during reconfiguration. We typically want the system to be up and running before the timeout on the database system triggers a reconfiguration in the database system.

The HPC[24] market has traditionally been the primary user of interconnect cards. HPC applications are not sensitive to communication downtime. HPC applications often use MPI between the computers, and the time it takes for the cluster to reconfigure is not an issue.

Applications that need HA are as mentioned in the fault-tolerance chapter, real-time systems. Systems like video streaming and telecom systems, and they often have a demand for a downtime below 60 milliseconds. The downtime we achieve at around 200 milliseconds is unfortunately not good enough for meeting these real-time requirements. The limitations that hold us back are limitations in hardware. The current revision of Dolphin SCI cards was finalized in 2001. The next revision of the SCI technology from Dolphin, called p2s has the capabilities to do much faster rerouting, because of hardware support for two addresses per node, where each address can employ a different routing strategy. The addition of hardware support for high availability in the p2s chip is a result of feedback from this thesis. These chips are, as this is written completed, and finished samples are expected back for testing in mid August. Simulations of the prototype chip have proven that it works, and we can expect to be able to meet the demands of the real-time systems with this chip.

[24] High Performance Computing

7 Conclusion

In this thesis we have implemented and tested a fault-tolerant routing algorithm in Dolphin Interconnect Soultions SCI product. Several tests have been done, and bandwidth, latency and communication downtime have been measured. In the tests we have seen that there is no degradation in performance if the packet has the same conditions, like distance to travel and competing traffic. In the other tests, the performance degradations we observed can be explained with more competing traffic on the same link, and longer distances and more B-link jumps for the packets.

The new routing algorithm is now a part of Dolphin Interconnect Solutions SCI driver under the name High Availability routing, and feedback from this thesis has lead to the implementation of hardware support for an improved fault-tolerant mechanism in the next revision of Dolphin SCI hardware.

7.1 Further work

Based on this implementation of a fault-tolerant routing algorithm in SCI networks I can suggest the following topics for further work:

- In this thesis, the algorithm was tested on a 4 node cluster. It can be interesting to test it on a lager cluster, i.e. a 16 node cluster. The tests in this thesis have also been done with simple tools for measuring bandwidth, latency and communication downtime. Tests should also be done with real applications like, for instance Oracle SQL server and MySQL to see what the improvements are compared with the existing network manager. Further optimization in the driver should also be considered.
- Investigate the possibility of integrating the new routing algorithm with the network manger. Have the fault-tolerant routing handling the error as soon as it is detected, and afterwards, use the network manager to analyze the temporary fix, and see if there is something to gain on doing a reconfiguration of the network.

References

1. IEEE, *Std. 1596 "Scalable Coherent Interface (SCI)"*. IEEE, 1992.

2. Dolphin Interconnect Solutions, *"PSB66 Specification D667"*. 2001.

3. Dolphin Interconnect Solutions, *"Link Controller 3 Specification D666 - LC3"*. 2002.

4. IEEE, *Std. 1596.3 "Low voltage differential signals (LVDS) for Scalable Coehrent Interface (SCI)"*. IEEE, 1992.

5. IEEE, *Std. 1596.8 "Parallel links to the Scalable Coherent Interface (SCI)"*. IEEE, 1992.

6. H. Kohmann, et al., *"Low-level SCI software functional specification"*. Esprit Project 23174 - Software Infrastructure for SCI (SISCI), 1999.

7. H. Kohmann and F. Seifert, *"SCI SOCKET - A Fast Socket Implementation over SCI"*. 2003.

8. E. W. Dijkstra, *"A note on two problems in connexion with graphs"*, in *"Numerische Mathematik 1"*. 1959. p. 269-271.

9. InfiniBand Trade Association, *"InfiniBand Architecture Specification"*.

10. Top500 Supercomputer List. http://www.top500.org.

11. Quadrics, *"QSNetII: An Interconnect for Supercomputing Applications"*. 2003.

12. N. J. Boden, et al., *"Myrinet: A Gigabit-per-Second Local Area Network.* IEEE MICRO, 1995.

13. Advanced Switching Interconnect Special Interest Group, *"Advanced Switching Technology - Tech Brief"*.

14. IEEE, *Std. 802.3 "CSMA/CD (Ethernet)"*. IEEE, 2002.

15. G. Coulouris, J. Dollimore, and T. Kindberg, *"A ring-based election algorithm"*, in *"Distributed Systems - Concepts and Design"*. 2001, Addison-Wesley Publishers Limited. p. 432-434.

16. I. Theiss and O. Lysne, *"FRoots, a Fault Tolerant and Topology-Flexible Routing Technique"*. 2005.

17. IETF, *RFC2328: "Open Shortest Path First Version 2"*. IETF, 1998.

18. A. S. Tanenbaum, *"Link State Routing"*, in *"Computer Networks"*. 2003, Prentice Hall. p. 360-366

19. O. Lysne, T. M. Pinkston, and J. Duato, *"A Methodology for Developing Dynamic Network Reconfiguration Processes"*. International Conference on Parallel Processing (ICPP'03), 2003.

20. T. M. Pinkston, R. Pang, and J. Duato, *"The Double Scheme: Deadlock-free Dynamic Reconfiguration of Cut-Through Networks"*. 2000.

21. M. E. Gómez, et al., *"A Routing Methodology for Achieving Fault Tolerance in Direct Networks"*. IEEE Transactions on Computers, 2006.

22. IBM BG/L Team, *"An Overview of the BlueGene/L Supercomputer"*. Proc. ACM Supercomputing Conference, 2002.

23. O. Lysne and T. Skeie, *"Load Balancing of Irregular System Area Network through Multiple Roots"*. International Conference on Communication in Computing, 2001.

24. M. D. Schröder et al, *"Autonet: a High-Speed, Self-Configuring Local Area Network Using Point-to-point Links"*. SRC Research Report 59, DEC, 1990.

25. F. Davik, M. Yilmaz, S. Gjessing, and N. Uzun, *"IEEE 802.17 Resilient Packet Ring Tutorial"*. IEEE Communications Magazine, 2004.

26. IEEE, *Std. 802.17 "Resilient Packet Ring"*. IEEE, 2004.

27. D. E. Comer, *"Another Example Ring Network: FDDI"*, in *"Computer Networks And Internets"*. 1997, Prentice Hall. p. 114-115.

28. IEEE, *Std. 802.5 "Token ring access method and physical layer specifications"*. IEEE, 1985.

29. Dolphin Interconnect Solutions. http://www.dolphinics.no.

Appendix

I Dolphin's SCI driver

We are now going to take a quick look at the driver and the structure of the driver. The driver is enclosed on a CD, and it contains the source code for the HA-driver, and for the tools used to measure communication downtime. Figure 47 shows the basic structure of enclosed driver.

The routing component is stored in the *DIS\src\IRM\drv\src\psb66\adapter* directory. And the routing functions specific to the 2D HA mode is stored in the *ha.c* file. Changes have also been done in the *lc3.c* file, stored in the *DIS\src\IRM\drv\src\psb66\adapter* directory. There have also been done changes in the *inter.c* and other IRM-related files in the *DIS\src\IRM\drv\src* directory.

The tool we made to measure communication downtime is also a part of this driver, and the source code is available in the *DIS\src\SISCI\cmd\test\downtime* directory.

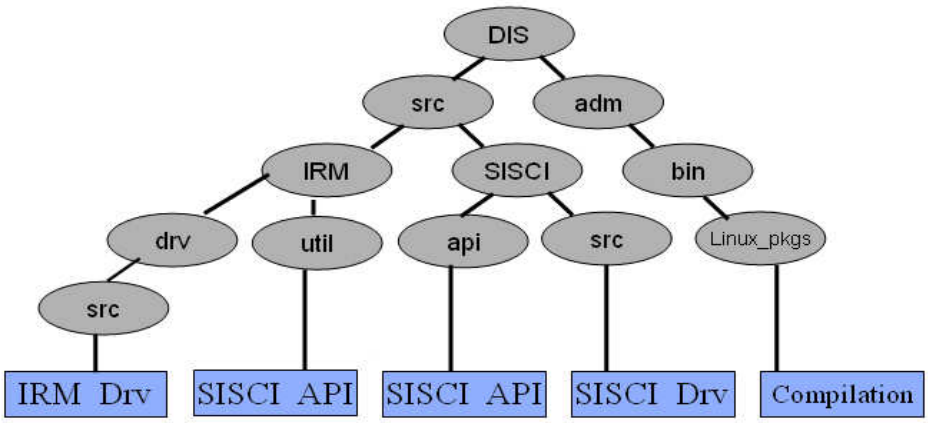

Figure 47: Basic structure of Dolphins driver

II Results

The output files from scibench2 and downtime used in the tests presented in the evaluation chapter, and a dump of the Linux kernel messages when the tests were run on each node is enclosed on the CD. These files are in the "*Test results*" directory, and they are sorted on the test number they have in the thesis.