

Real-Time Services
in Myrinet Based Clusters of PCs

Alejandro Garcia, Lisbeth Johansson, and Mattias Weckstén

January 2000

Acknowledgment

This Master's thesis is the concluding part of the master program in Computer Systems Engineering at Halmstad University. The students receive a Master degree in Computer System Engineering when the final examination is completed.

The work covered by the thesis has been carried out during the late spring 1999 to January 2000 at Centre for Computer System Architecture (CCA), Halmstad University.

We would like to thank our advisor, Associate Professor Magnus Jonsson for his support and guidance throughout the project, Associate Professor Thorsteinn S. Rögnvaldsson for clearing things up in the formulas, Magnus Moren for support on the LINUX system, the SPRINT-project and the review group for their point of view on the oral presentation, Nanette Boden, Glenn Brown, and Bob Feldermann et al. at Myricom, USA, for their support for GM and the Myrinet hardware, research programmer Kees Verstoep, and Professor Henri Bal at Vrije University, the Netherlands for their support for the LFC software, and James Otto, University of New Mexico, USA, who adapted a tutorial on how to write Myrinet Control Programs.

Abstract

High performance modular switch-based interconnection networks have become popular for both embedded computer systems and clusters of PCs/workstations. However, these networks have typically no, or only limited, support for real-time communication. Applications like computerized visualization and multimedia streaming need support from the network to guarantee bandwidth and latency.

In this thesis, a method is presented of how to get guaranteed hard real-time services for periodic traffic in a wide range of switched networks, which is verified by analysis and experiments on Myrinet where important parameters have been characterized. The solution does not require any additional hardware or modification of the switches, i.e., it is purely based on software implementation in the end nodes. The solution includes reservation of network links and bandwidth sharing using Time Division Multiple Access (TDMA). With reservation of network links and TDMA, blockage is avoided in the network and the worst case latency for a packet is kept short. Nor does any deadlock occur, which makes the system very stable without any constraints on the topology.

Contents

1	Introduction	1
1.1	Thesis Contribution	1
1.2	Project Boundaries	2
1.3	Possible Applications...	3
1.4	Thesis Overview	3
2	Background Technology...	5
2.1	Routing	5
2.2	Switching	6
2.3	Deadlocking	7
2.4	Topologies	8
2.5	Real-Time support in Switched Networks	9
2.6	High-Performance Networks	12
3	Available Software	15
3.1	User Level Protocol	15
3.2	PVM and MPI	16
3.3	MPI-RT	17
3.4	Real-Time Kernel	17
4	Resource Planning and Time Keeping	19
4.1	Reservation of Communication Links	19
4.2	Clock Synchronization Method	20
4.3	TDMA (Time Division Multiple Access)	22
4.4	Early Sending in TDMA	27
5	Implementation	29
5.1	The Myrinet Control Program	29
5.2	The Clock Synchronization	29
5.3	TDMA Implementation	31
5.4	The Scheduler	32

6 Results	37
7 Discussion	39
7.1 Other Implementations	40
7.2 Conclusion – Future Work	40
A Scheduling example	45

List of Figures

1.1	A system overview	2
2.1	Switching method relationships	6
2.2	Priority inversion. Message B with priority 4 wants to use the same channel as a message A with priority 2 already is using. Message A, however, is blocked by an downstream continuous flow of messages from C with priorities 3 resulting in message B being blocked for an unpredictable, long time.	9
2.3	A Clos network topology.	14
4.1	Downstream blockings that can occur when a source message tries to reach its destination.	20
4.2	A theoretical perfect clock compared to a clock with static and random drift. The linear regression of the real clock is indicated in the diagram.	21
4.3	TDMA cycle when the clock synchronization is separated from the rest of the data traffic.	23
4.4	The TDMA slot.	24
4.5	TDMA cycle when the clock synchronization is scheduled among with other data packets.	25
4.6	Clock synchronization in a 4×4 mesh.	26
4.7	Early sending example.	27
5.1	State-machine representation of the software present in the network interface.	30
5.2	The channel utilization for different packet sizes. The theoretical curve describes the optimal performance, the measured curve describes the implemented solution.	31
5.3	A structural overview of the system.	34

- A.1 The two setups used for the scheduling example. A is a network without redundant paths, and B is a network with redundant paths. 46

Chapter 1

Introduction

Most parallel computers are made for non-real-time number-crunching applications (e.g., scientific calculations) and with a hard-wired architecture. It would be desirable to have a cost-effective and dynamic real-time parallel computer since real-time support will gain importance in the future for both multimedia and calculation applications.

This master's thesis examines the possibilities to expand the field of parallel computing networks with real-time support. The intention has been to construct a platform and a topology-independent method to carry out real-time constraints over switched networks. A significant part of the work has been to do an implementation on a cluster of (Pentium-based) PCs coupled together by Myrinet. The major part of this work concentrate on examining real-time communication.

1.1 Thesis Contribution

This thesis shows that it is possible to introduce support for hard real-time demands in generic source routed switched networks using Time Division Multiple Access (TDMA) and resource allocation, implemented purely in software, and in the end nodes. To be able to use TDMA, all nodes has to have synchronized clocks, and it is shown that this can be accomplished by scheduling synchronization messages mixed with the original Myrinet load. Measurements on a small-scale implementation have been performed and are used when developing feasibility test for larger switched systems.

The approach is not limited by any particular topology, since resource allocation in a TDMA schedule avoids deadlock. The work has been focused on supporting periodic traffic, neither aperiodic or isochronous traffic is directly supported; however, it is possible to achieve this using periodic

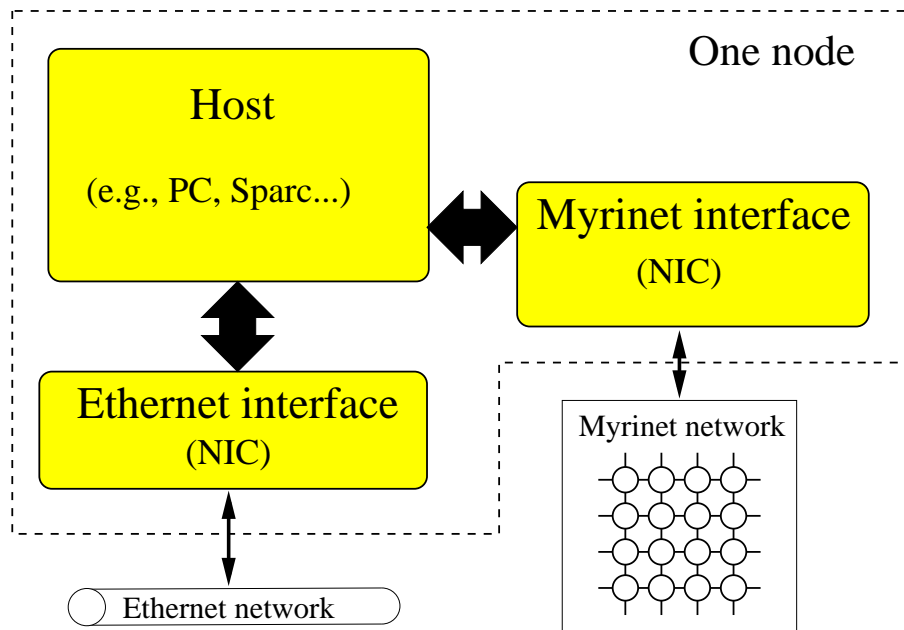


Figure 1.1: A system overview

reservations. The reservation of channels can be altered during runtime (i.e., dynamic allocation of logical channels with real-time support is possible). The resource allocation is not handled over Myrinet (although it is possible), Instead, Ethernet is used for this purpose, mainly because of the broadcast function. Figure 1.1, shows a system overview with the blocks of one node. Each node consists of a host computer (e.g., Pentium PC) with a Ethernet Network Interface Card (NIC), and a Myrinet NIC.

1.2 Project Boundaries

The scheduling algorithm for the TDMA is simple and non optimizing. However, this does not limit our results, and the algorithms can easily be exchanged. Fault tolerance is not handled in the scope of this thesis since the main point is to prove that it is possible to guarantee timely delivery of real-time messages over a switched network. On the other hand, it should not be complicated to add redundancy for error correction.

1.3 Possible Applications for High-Performance Real-Time Networks

To motivate the use of real-time services in a high performance network, real-time applications with bandwidth demanding tasks are needed. The following list shows typical possible applications.

- Radar computations, active antenna (regulated systems)
- Real-time simulation tasks (high performance real-time forecasting)
- Digital X-ray (tomography - graphic calculations)
- All graphic calculations (computerized visualization, 2d, 3d)
- Office (tele/ video/ data communication) backbone

1.4 Thesis Overview

In Chapter 2, networking theory is presented as an orientation on the subject. In Chapter 3, available software to develop from is presented, and different services is located. In Chapter 4, the resource allocation, clock synchronization and TDMA are discussed, and different solutions are presented. In Chapter 5, the implementation is presented and the different choices are motivated. In Chapter 6, the results from the tests on the implementation are presented. Finally the work is concluded and discussed in Chapter 7, and future work is presented.

Chapter 2

Background Technology and Related Work

This chapter summarize recent work done in the field of switched networks, and describe relevant aspects concerning routing techniques, switch techniques, related topologies, and how to deal with deadlocking. It also shows how others have introduced real-time support on switched networks. Finally is a description of two high-performance networks, of which one supports real-time by hardware.

2.1 Routing

Routing in switched networks is essentially a dynamic resource allocation problem. There is potential for deadlock in a situation where multiple PEs are attempting to acquire the same sets of route hops simultaneously.

Source Routing and Adaptive Routing

In source routing the packet header holds the routing information for every switch that it will pass on the way to the destination. In every switch the header will be analyzed to retrieve the routing information for this particular switch. Before the header is propagated the used routing information will be stripped from the packet header. This implies that routing information for all possible destinations has to be available at each node that wants to send messages over the network. The Myrinet switches use this strategy for routing.

In adaptive routing each switch has the necessary information on how to route an incoming message to the desired destination. This makes it possible

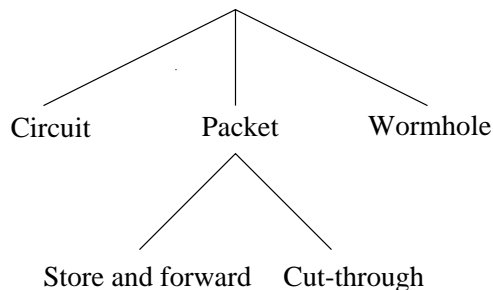


Figure 2.1: Switching method relationships

to change the routing path “on- the- fly”. The possibility for deadlocking is thus reduced, and the time a message is delayed because of already occupied paths is reduced. RACEWay (see Section 2.6) primarily uses source routing, but have in some network topologies an adaptive routing option.

2.2 Switching

The switching methods addressed to in this section are all related in some way. In spite of their similarity there are some distinct differences. For an overview of the relationships, see Figure 2.1.

A switched network consists of a number of nodes and a number of cross-bar switches. Each node is coupled to a switch. These connections are called ports. Between each pair of switches there can be a number of connections. Each coupling between two switches is called a trunk.

Store and Forward

Store and forward collects the whole packet and stores it in the switch before the packet is forwarded to the next switch or host. This is accomplished by adding memory buffers to the switch, and by adding constraints to the message length.

Virtual Cut-Through Routing

In virtual cut-through the message is passed on to the network and, if the outgoing link is free, directly propagated through the switch as soon as the address field has been evaluated. If the outgoing link is busy the message is drawn into the switch’s buffer (as in store and forward); this releases the incoming link for other tasks. When the outgoing link is available again the stalled message is resuming its transmission towards the destination.

Circuit Switching

Circuit switching is accomplished by reserving a physical channel from source node to destination node, this by sending out a routing probe on the network. When the desired route is reserved an acknowledgment will be propagated backwards to the source and the transmission can take place.

Wormhole Switching

When a package is propagated over the network using wormhole switching the message is directly propagated through the switch if the outgoing link is free. If the outgoing link is occupied with other traffic the message “worm” is halted and the data on the network medium is absorbed in tiny switch buffers along the route (the message propagation is hence stopped). All upstream links are now blocked and remain so until transmission can resume and the message is fully transferred.

Wormhole Switching with Virtual Channels

The virtual channel version of wormhole switching takes the advantage of using multiple small buffers for each link. If one message is stopped, and thus holding a number of link buffer resources, another message still can bypass this reserved channel by the use of a parallel buffer.

2.3 Deadlocking

In a switched network blockages can occur (i.e., when a data stream is to use a resource that already is occupied). If these blockages are cyclic, deadlock has occurred (i.e., all involved streams wait for a resource that is held by another stream). To handle deadlocking it either has to be completely avoided, or if it is allowed, a technique to solve this situation has to be applied.

Deadlock Avoidance

The most common deadlock avoidance strategy is always to allocate resources in a fixed order. When using a deadlock avoidance routing algorithm, the system will never reach the state of deadlock.

With wormhole routing, since the resources are allocated as the message propagates through a network, this affects routing strategy for a given topology. For example in a hypercube or a grid, deadlock free routing is possible to ensure by resolving the address one dimension at a time in ascending order.

Another method to achieve deadlock avoidance is to split each physical channel into virtual channels. Each virtual channel will use its own queue for in and out buffer, but larger memory capacity is required in each node.

Deadlock Resolving

When using a deadlock resolving system deadlock is allowed to appear; however, the routing algorithm will solve the deadlock situation. One method is to allow preemption of packets that will cause a deadlock situation. The packets can then either be discarded or rerouted.

2.4 Topologies

The importance of choosing a suitable network topology depends on the type of communication. When using real-time communication it is essential to have short communication paths and more than one path to the other nodes to reduce blockage probabilities.

Binary Hypercube

The direct binary hypercube [1] is one of the most widely used topologies for interconnecting resources in parallel computing systems, because it provides a short network diameter, each node is connected to n others, and its coordinates are one of the 2^n different n -bit sequence of binary digits. It can efficiently emulate the majority of the topologies frequently employed in the development of algorithms.

A typical and recommended topology using Myricom M2M-DUAL-SW8 switches is a binary hypercube [2].

Fat Tree

The fat tree topology [3] is suitable because the bottleneck in an ordinary tree will be removed by allocating a higher channel bandwidth to channels located close to the root node. The shorter the distance to the root node, the higher the channel bandwidth.

Cube-Connected Cycle

The cube-connected cycles [1] topology can be considered as an n -dimensional hypercube of virtual nodes, where each virtual node is a ring with n nodes, for a total of $n \cdot 2^n$ nodes.

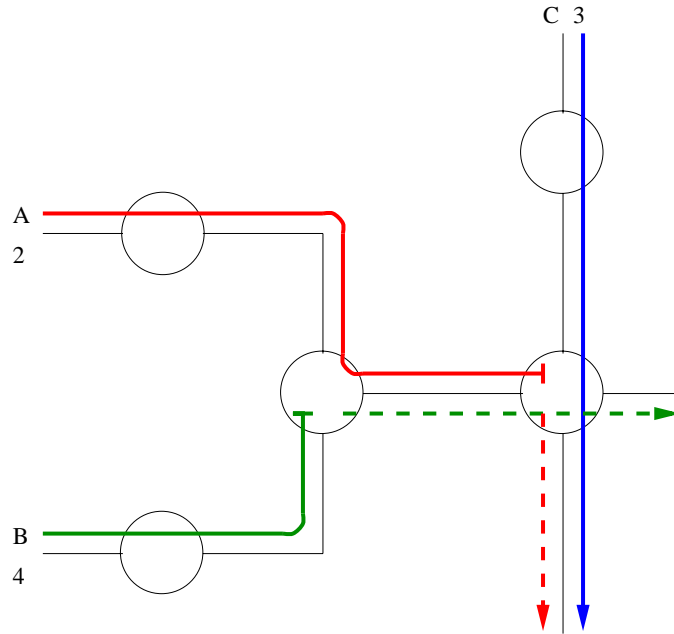


Figure 2.2: Priority inversion. Message B with priority 4 wants to use the same channel as a message A with priority 2 already is using. Message A, however, is blocked by an downstream continuous flow of messages from C with priorities 3 resulting in message B being blocked for an unpredictable, long time.

2.5 Real-Time support in Switched Networks

To introduce real-time support in generic switched networks, a method to guarantee that a message is delivered at a certain time is needed. In a switched network several nodes can access the same links, and this may cause messages to block each other. If the blockages are too many or last for too long it could cause other messages to miss their deadlines. Since the latency depends on the numbers of downstream blockages, the worst case latency will quickly get large when there are many nodes using the same network links and the guaranteed throughput will decrease as described in Sundaresan and Bettati [4].

In many approaches the use of priorities is an essential method[5][6]. One of the most important issues to solve in a switched network with priorities is priority inversion (Figure 2.2). This occur when lower priority message blocks messages with higher priorities. For instance, a message B with priority 4 wants to use the same channel as a message A with priority 2 already is using. Message A, however, is blocked by an downstream continuous flow of

messages from C with priorities 3 resulting in message B being blocked for an unpredictable, long time [7]. One solution could be priority inheritance, e.g., message A inherits message B's priority; however, this may be hard to implement in a wormhole switched network without a special crossbar switch suited for this purpose. Yet another approach is to offer a Quality of Service (QoS) to the traffic in the network. In a paper presented by Connelly and Chien [8], the Myrinet user level protocol Illinois Fast Messages (FM) [9] is extended with deterministic latencies and guaranteed bandwidths. QoS is commonly used in different networks e.g., in packet switched [10].

Many solutions includes changing the switch hardware or adding something to it. Some solutions includes attaching a computer to every switch in the network, by this a programmable switch behavior is achieved. However, all these hardware changes in the network makes it more expensive. Another essential thing to consider is the increased time it takes for a message to pass through a computer-switch. When it comes to solutions involving software only, they are rare. In the approach of this thesis the use of the time-slotting mechanism in TDMA (over reserved links in the network) is explored.

In the rest of this section, TDMA is explained and some available methods for guaranteeing real-time services over switched wormhole networks are discussed. Most available methods involves the use of additional hardware which often makes them very efficient, but expensive. The software based methods are cheaper but demands computational power.

TDMA (Time Division Multiple Access)

The TDMA protocol divides the network usage in the time domain. The time is divided in so called slots of certain length where hosts are allowed to send messages. By scheduling in an appropriate way this becomes a collision free and time deterministic protocol. A problem that can occur is that the CPU load from the scheduling task is so high that it degrades the host performance.

For the TDMA scheme to work properly and to avoid blockages, the clock in every node need to be synchronized with all other clocks. If at least one node has an unsynchronized clock it may disturb or block other messages in the network, making them miss their deadlines. One common method to solve the synchronization is to use a shared clock wire, all nodes can read the clock whenever they want. This is called a global clock as one and the same clock determine all the nodes clock values. Another approach is to use distributed clock synchronization where the nodes synchronize with each other at certain intervals.

There are some protocols available that use time-slotting (reserved time for transmission). In [11] a dynamic method for slot reserving is presented, in

this case it is used in a WDM (Wavelength Division Multiplexing) fiber-optic star network. The protocol is based upon TDMA and it provide services for both best effort messages and guarantee-seeking messages. Multiple channels exist due to WDM and the access to each channel is divided into cycles of time-slots. Each node have some reserved slots to service guarantee-seeking messages. If a node do not have any guarantee-seeking messages its reserved slots can be released for best effort messages (from other nodes or the same node).

Hardware Based Solutions

In a special router architecture called Throttle and Preempt, flit-level preemption and prioritized packets are implemented using virtual channels [7]. This strategy require a special crossbar switch with programmable interface as the behavior of the switch and the use of the switch's buffers must be altered. Another special router called the Time-Deterministic Communication Chip provides a low-level communication mechanism that supports time-deterministic delivery of messages [12]. Other solutions that involves the use of virtual channels are e.g., a flit-level preemption method presented in [13], and a priority mapping protocol that includes priority adjustment and message dropping [5]. For instance, there are many switches available for the Asynchronous Transfer Mode (ATM) network that supports priority. When using these priority queuing switches ATM is well suited for real-time purposes. In [14] Hansson et al. presents a method for providing hard real-time traffic through an ATM network .

Software Based Solutions

A protocol called LDCR-G (laxity based deterministic collision resolution protocol with guarantees), is discussed and extended in [15]. This protocol is especially suited for multimedia applications as it is dynamic, meaning that if a new message arrives at a node, it will not be scheduled unless it does not jeopardize guarantees made for previously admitted messages. Messages are admitted based on an estimate of worst case message service time, and their transmission is guaranteed if their service time do not exceed this estimate. Laxity is used for deciding the messages priority, lower laxity messages have higher priority than higher laxity messages. The nodes in LDCR-G LANs are mapped according to the Collision Resolution tree (CR), to resolve situations when collision in data transfer occurs. If no collision occurs, and if the notifier transmission is successful, the message is transmitted. This protocol requires

a method to determine if a collision has occurred and in that case withdraw the message it tried to send. Usually, this method is quite time consuming.

The approach of using time-division multiplexing to provide real-time over a switched network have been used earlier. For instance has it been implemented in a High-Performance Parallel Interface (HIPPI) network as presented in Bettati and Nica [16]. HIPPI is a circuit-switched network that uses a camp-on¹ feature to determine when a desired resource (e.g., a network link) becomes available. Bettati and Nica define a set of real-time channels based on the requested workload of the network. Channels that use the same network link can not be reserved during the same time slot. The solution presented by Bettati and Nica is similar to the one presented in this thesis, the major difference is that they are using a circuit-switched network instead of a wormhole switched one. When several Sources contend for a single Destination, the Camp-on feature allows the HIPPI switch to arbitrate and ensure that all Sources have fair access. Without Camp-on, the contending Sources would simply have to retry the connection repeatedly until it was accepted, and the fastest Source would usually win.

2.6 High-Performance Networks

Some available networks for commercial use are constructed for high speed data transmission. The next subsection describes Myrinet, the network used in the implementation in Chapter 5. This network does not give support for real-time transmissions in hardware. As an comparison, the RACEWay network that do support real-time traffic by hardware, is also presented.

Myrinet

Myrinet is a switched network built up of high performance communication-links with a performance of 1.28+1.28 Gbit/s, full duplex [17]. The network card is equipped with a RISC-processor and is programmable to show a predefined behavior. The Myrinet switches are *perfect*, meaning that packets do not conflict at a switch unless they are directed to the same outgoing link.

Myrinet uses source routing (Section 2.1), and wormhole switching (Section 2.2) which does not require large local buffers. The only two signals available from the switch are data_on and data_off flow-control. There are

¹When several Sources contend for a single Destination, the camp-on feature allows the HIPPI switch to arbitrate and ensure that all Sources have fair access. Without camp-on, the contending Sources would simply have to retry the connection repeatedly until it was accepted, and the fastest Source would usually win.

several different switches available, from four to eight ports with direct contact within a switch, and no broadcast functionality is available in hardware. The worst-case switching time within a switch is about 500 ns.

RACEWay

RACEWay is a dynamically switched and scalable fabric of interprocessor links developed by Mercury Computer Systems Inc. The RACEWay network consists of common 6 or 8-port crossbar switches and node-specific RACE port interfaces. It is designed especially for embedded real-time systems and is providing point-to-point communication between processors in the network with a transfer rate of 160 Mb/s. The system is capable of an aggregated bandwidth of over 1 Gb/s. Each port of the crossbar switch can be connected to any one (or more if broadcasting) of the remaining ports. The switching time in each crossbar switch is about 125 ns. Each crossbar switch is able to simultaneously provide up to three or four independent port-to-port connections. An important issue is that RACEWay provides for dynamic priority of data transfers which allows a high-priority data transfer to preempt one or more lower-priority data transfers if necessary. This means that the completion of the data-transfer can be guaranteed within a specific time limit determined at compilation. There is also a built in priority within the single switch, meaning that if both port 4 and 2 wants to send data (having the same packet priority) to port 1, the packet from port 4 will prevail, letting the other data transfer to either be held-off or to be temporarily suspended. The routing-technique used is source routing, but a adaptive routing option is available for fat-tree and Clos networks (see Figure 2.3), giving the network a more flexible choice of path from the source to the destination.

There are several network topologies that are applicable when using RACEWay. Especially well suited is the modified "Fat-Tree" network. It uses two ports of every crossbar switch to connect to other switches on a higher level in the tree, this creates two independent routes from one node to another in a two level tree [18]. Other possible network topologies are 2-D and 3-D mesh, and ring networks.

The data transfer packet consist of a two-word (64 bits) packet header and a data part of variable-length, up to maximum total packet-length of 2048 bytes. The packet header contains information that defines the data transfer path, the block starting address at the receiving node (usually a DRAM memory address), and other control information as packet priority, direction of transfer, broadcast enable. The first 27 bits of the header comprise the crossbar routing field and these bits are able to define the path through up to 9 crossbars in the network.

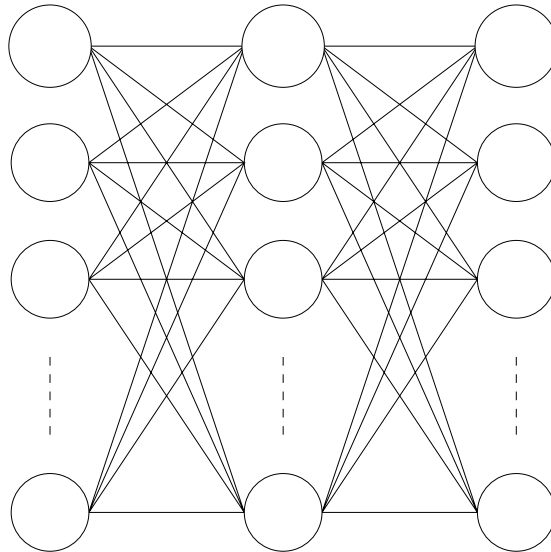


Figure 2.3: A Clos network topology.

Myrinet or Raceway?

The primary difference between RACEWay and Myrinet is that RACEWay uses priorities, both in hardware and software, another is the routing technique. One of the goals with this project is to introduce real-time communication into Myrinet. The use of priorities is one way of doing it, but priority bits in the packet header, as in RACEWay, would require modification of the switch-hardware. The proposed solution in this thesis is to use resource allocation in order to guarantee the packets deadlines. This is possible to realize using software only.

Chapter 3

Available Software

In this chapter the usability of available software, developed for message passing over Myrinet has been examined. In order to speed up the developing pace it is important to choose the right point to start development from, and to use as much work already done as possible. Both user level protocols discussed below showed to be either too complex to adapt or not having the proper functionality for the implementation done in this thesis. Two Myrinet protocols (GM and BIP) are presented below, yet another protocol has been considered, the LFC (Link-level Flow Control) [9] but it was also too complex to adapt. Higher level protocols like PVM/MPI are supported by most Myrinet protocols and need to be considered when creating a new Myrinet protocol. The functionality of PVM/MPI is explained below. A real-time kernel for Linux is a step in the development, since real-time is desired for the scheduling algorithm as the schedule need to be calculated simultaneously in all network end-nodes. The RT-Linux kernel was used in the experiments done in this thesis and is presented below.

3.1 User Level Protocol

The following two subsections shows protocols for Myrinet parallel computing. Neither of them supports real-time message passing. It gives an overview of how a communication package could be constructed. GM is a product from Myricom, who developed the Myrinet network.

GM

Myricom distributes a message-passing system, for their Myrinet cards, called GM [19]. GM provides reliable ordered delivery between hosts in the pres-

ence of network faults. GM will reroute packets around network faults when alternate routes exist. The GM system supports cluster sizes over 10,000 nodes. GM also provides two levels of message priority to allow efficient deadlock-free bounded-memory forwarding. GM allows users to send messages up to $2^{31} - 1$ bytes long, under operating systems that support sufficient amounts of DMAable memory to be allocated. Finally, GM is designed to support both network nodes with and without secondary processors. That is, programs using the GM message passing interface can be run directly on the LANai processor present on Myrinet interface cards, Myricom FPGA, nodes, etc. GM automatically maps Myrinet networks. GM is a light-weight communication layer, and as such has limitations that can be addressed by layering a heavier-weight interface over GM. Two such limitations are the following: (a) GM is unable to send messages from or receive messages into non DMAable memory, and (b) GM does yet not support any gather or scatter operations directly.

BIP

Another possible choice could be BIP (Basic Interface for Parallelism) [20], a small API (Application Program Interface) that were designed, and implemented on a Myrinet network and looks similar to GM with support for parallel programming. Since BIP is optimized and stripped down this gives greater performance on the limited operations. It delivers to the application the maximal performance achievable by the hardware. BIP messages are implemented for cluster of x86/Linux workstations, linked by Myrinet boards with the LANai4.1 processor. The implementation consists of a user-level library associated with a custom MCP (Myrinet Control Program) that will run on the Myrinet board.

3.2 PVM and MPI

PVM (Parallel Virtual Machine) is a software package that permits a heterogeneous collection of Unix and NT computers hooked together by a network to be used as a single large parallel computer. Thus large computational problems can be solved more cost effectively by using the aggregate power and memory of many computers. With thousands of users, PVM has become the De facto standard for heterogeneous cluster computing world-wide. The source is available free thru netlib and has been compiled on everything from laptops to CRAYs. MPI (Message Passing Interface) is an upcoming standard for clustering and parallel computing that will be more important

in the future [21].

3.3 MPI-RT

The MPI has been extended with real-time QoS which adds greater predictability and schedulability to the network-communication [22] [23]. The API gives ability to advance planning with resource reservation. MPI/RT is a scalable message passing standard suitable for large parallel system, but requires early-binding¹ organization and detailed program design. MPI/RT supports three parallel programming models; data-parallel, task parallel, and coarse-grained data-flow comprising cascades of data-parallel groups. As in MPI, MPI/RT offers the functionality of a virtual channel. This gives the ability to exploit persistent communication common for high performance real-time applications, deadlock and livelock avoidance, virtual channels guarantees for properties critical for timing correctness, and more efficient resource usage. The real-time programming paradigms supported are time-driven, event-driven, and priority-driven. The event-driven paradigm is usually used in conjunction with priority-driven or time-driven paradigm. MPI/RT also provides limited functionality for fault handling.

3.4 Real-Time Kernel

There exists a number of real-time solutions where RT-Linux [24] is one. RT-Linux is interesting since it introduces a small, real-time executive that runs the Linux operating system as just another task, with its lowest priority. Linux is thus made completely preemptable. The executive is then able to give undivided service to its user-specified "real-time" tasks, whose functionality, behavior and interactions are supported by RT-Linux capabilities. The gain with RT-Linux is that the kernel remains unharmed and will work as usual for non real-time tasks.

¹Early reservation of bandwidth, for instance during the startup sequence of the network. This is possible for periodic messages.

Chapter 4

Resource Planning and Time Keeping

To pass messages with hard real-time constraints over a generic switched network with no specified topology, a method to guarantee the bandwidth for messages is required as well as a way to determine the network topology. The determination of the network topology is possible to accomplish in software, as described by Myricom [2]. Although, this method will not be discussed in this thesis.

To allow multiple data-streams to be transmitted over a shared media, time domain multiplexing could be used in combination with reservation of every single trunk and port in the system in each frame of time. This works only if all nodes has an unified apprehension of the time (i.e., some kind of clock synchronization is needed).

4.1 Reservation of Communication Links

One of the most important approaches in the project is the reservation of communication links in the network, since this is essential in order to avoid blockages between different messages in the network switches. Blockages make the worst case transmission time of a message to grow [4]. It grows exponentially with the number of switches that has to be passed from the source to the destination. Figure 4.1 shows the downstream blockages that can occur when a source message tries to reach its destination. In every switch the source message has to pass, it might be forced to wait for messages that have the same port destination in this switch, and which reached the switch earlier. The messages e , and f might also be blocked in every switch they have to pass, increasing the blockage time for the source message. The worst

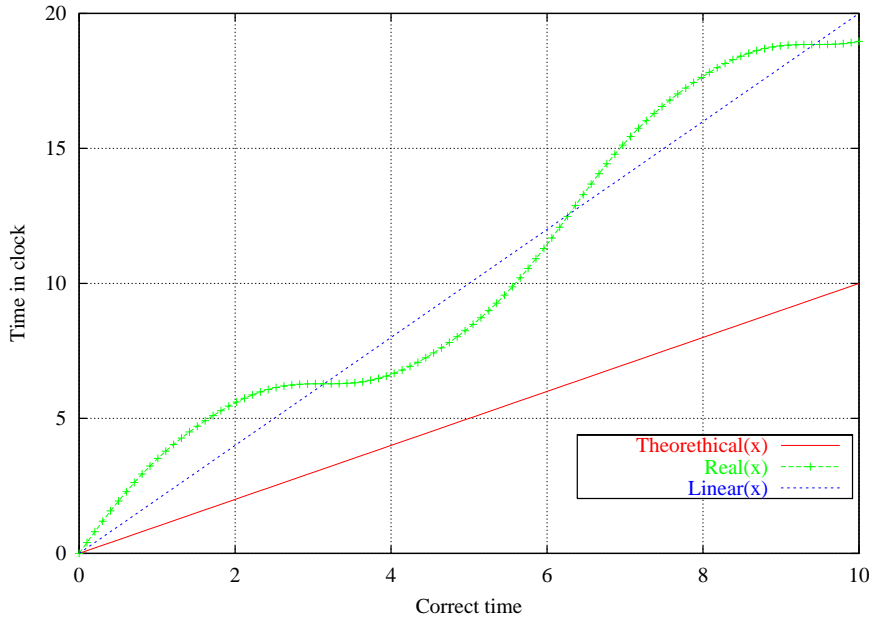


Figure 4.2: A theoretical perfect clock compared to a clock with static and random drift. The linear regression of the real clock is indicated in the diagram.

An easy approach to synchronization is to assume that one clock in the overall system (i.e., the master clock) is correct and that all other clocks need to synchronize. If the clocks are synchronized often, the error, $f_0(t_c) - f_n(t_c)$ (where $f_0(t_c)$ represents the master node's time and $f_n(t_c)$ represent the time in slave node number n), will go toward zero. If the nodes are not allowed to synchronize often, the static drift will result in a large error. Assuming that the random drift is independent of earlier values and has a zero mean it is seen that the linear regression will have the same inclination as the static error (Figure 4.2). Since the clock drift for any clock compared to the perfect theoretical clock is assumed to be a linear function, the difference between any two clocks drifts will be a linear function too. This implies that the error could be handled by calculating the mean difference between the two nodes total drifts:

$$\begin{aligned} \langle f_0(t_c) - f_n(t_c) \rangle &= \langle (k_0 - k_n) \cdot t_c \rangle + \langle r_0(t_c) + r_n(t_c) \rangle \\ \langle f_0(t_c) - f_n(t_c) \rangle &= (k_0 - k_n) \cdot t_c + 0 \end{aligned} \quad (4.2)$$

and compensating for the drift. Since the master node is assumed to hold the correct time (i.e., $k_0 = 0$) the drift compensation for clock n could be

calculated from all previous errors in mean.

$$\begin{aligned}
 k_n &= -\frac{\langle f_0(t_c) - f_n(t_c) \rangle}{t_c} \\
 k_n &= \frac{\langle f_n(t_c) - f_0(t_c) \rangle}{f_0(t_c)}
 \end{aligned}
 \tag{4.3}$$

With the help of the proposed method, the actual time can be held very precise since the clock deviation could be calculated at every time needed between synchronizations. The only weakness is that the model of the clock is not valid in larger time frames. In a larger time frame, variations because of the temperature will be observed since this will alter both the constant and the random drift.

Since the theories for clock synchronizing are valid only under the circumstance that the synchronization period is small (i.e., thousands of milliseconds), this has to be accounted for when the slot lengths and TDMA-cycle lengths are chosen. When the clocks has to be synchronized, a number of synchronization messages are sent, one for each slave node. It is very important that these messages are not blocked at all in the network. Possible blockages could appear because of the clock drift in a node. To assure that a synchronization message is not blocked, a margin could be inserted for these messages.

In smaller networks, the synchronization could be allowed to be a series of messages from one synchronization master (see Figure 4.3). In larger networks this method is not efficient enough since the synchronization period would grow linearly with the number of nodes. In this latter case one could think of methods that divide the workload at every synchronized node or a master-master that first synchronizes the masters and then allows the masters to synchronize their slaves (i.e., tree distribution). Different methods for reserving resources for synchronization, and data transmission, are discussed in the following sections.

4.3 TDMA (Time Division Multiple Access)

When using TDMA the access to the network is divided into small time-slots. The reservation of the slots are handled by a scheduling algorithm that decides which data packet that is allowed to be sent in which time-slot (Figure 4.3). By using the different periods of the data traffic in the network a scheduled cycle can be calculated using the least common multiple of the period. In this way the scheduled cycle can be reused until a new cycle is created. If the allowed data periods are restricted to only a few periods the scheduled cycle can be made short.

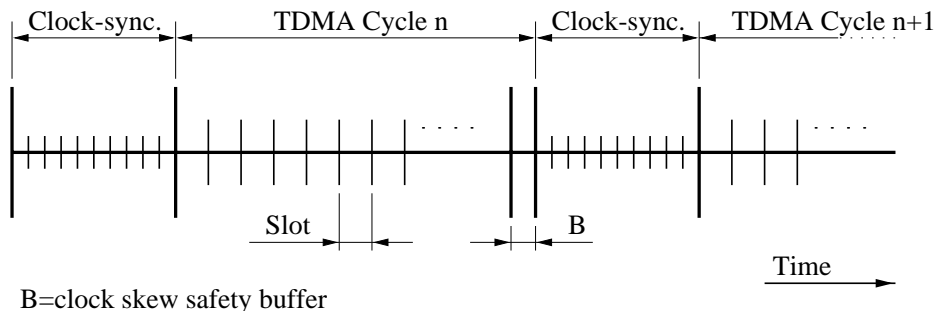


Figure 4.3: TDMA cycle when the clock synchronization is separated from the rest of the data traffic.

TDMA - Slots and Cycles

The size of each TDMA slot compared to the maximum allowed data packet size decides the maximum utilization of the network. Small data packets in much larger TDMA slots makes the unused margins large, and the network utilization gets low (Figure 4.4). See Section 5.3 for further information about the slot-length and data packet size ratio. If the network have large clock drift the margins in the slots need to be larger (than if the clock drift is small) in order to achieve non-blockages in the network. The alternative is a more frequent clock synchronization. If the network handle collisions of messages by blocking one of them (i.e., Myrinet, Section 2.6) the margins can be kept small as the network is able to handle blocking situations without destroying any messages. If a message begins its transmission a short time (relatively to the slot length) before it is allowed to, it will be held up by the network if the needed links are occupied with a data packet belonging to the previous TDMA slot (see Section 4.4).

Two different approaches for the creation of the TDMA cycle have been considered in this work. The first approach was to have the clock synchronization part separated from the rest of the TDMA cycle. By this the utilization of the bandwidth gets better as the clock synchronization messages are much smaller than the maximal allowed data packets (e.g., many clock synchronization messages can be transmitted during the length of an ordinary TDMA slot). For instance, the necessary time required for clock synchronization is about $5 \mu\text{s}$ for each node, and the time required for sending a message of 3400 bytes is about $30 \mu\text{s}$. However, this makes the TDMA cycle fixed to a specific length in order for the clock synchronization to reappear at certain intervals (Figure 4.3). As the clock synchronization period occupies a continuous period of time when no other traffic is allowed it affects the minimum time period of data packets. The minimum time period has to be larger

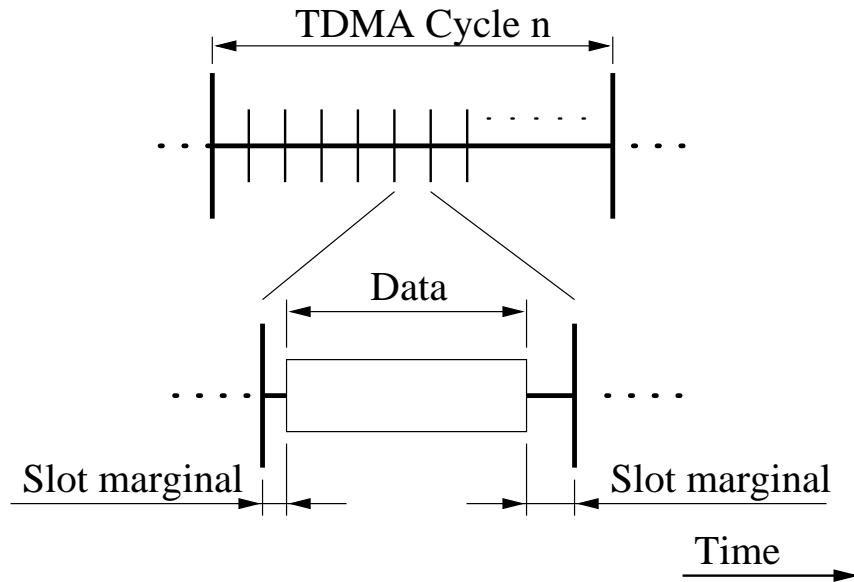


Figure 4.4: The TDMA slot.

than the total clock synchronization time. As mentioned earlier is the length of the TDMA cycle in this approach fixed to a specific length. The length of the TDMA cycle depends on the least common multiple of the different messages' period times, making the allowed period times restricted to only a few (in order to make the least common multiple fit into the fixed amount of TDMA-slots). Between the TDMA cycle and the clock synchronization, a small clock skew margin is necessary to secure that the network is empty from data traffic before the clock synchronization phase begins.

The other approach considered schedules the clock synchronization messages among with all other real-time messages in the network (i.e., logic channels are established for clock synchronization in the same way as for normal data)(see Figure 4.5). In this way, a whole TDMA slot will be used for one clock synchronization message (e.g., normal TDMA slot approximately $30 \mu s$, clock synchronization message transmission time approximately $5 \mu s$). However, only the necessary network links will be occupied, i.e., the rest of the network can be used for data transfers as usual. When using this approach the TDMA cycle is almost unlimited in size, and the minimum packet period gets down to the size of two TDMA slots.

Regarding the second approach problem do occur when a master node have many clock synchronization messages to send (see Section 4.2). If the maximum allowed slave nodes per synchronization master are 20, the slot length is $30 \mu s$, and the clock synchronization period, during which all nodes

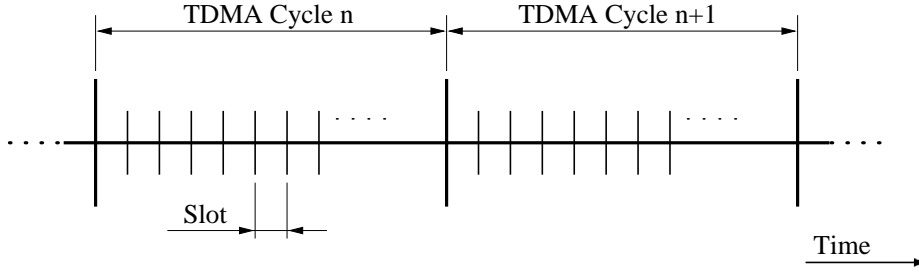


Figure 4.5: TDMA cycle when the clock synchronization is scheduled among with other data packets.

have to synchronize, is for instance 40 slots (1.2 ms), half of the master node's bandwidth will be occupied with clock synchronization messages. As the clock synchronization messages are much smaller ($5 \mu s$) than the slot length, the utilized part of every clock synchronization slot is only $\frac{1}{6}$. However, this utilization problem can be solved by making the TDMA slots smaller, almost down to the necessary time for transmitting a clock synchronization message ($5 \mu s$). But if every packet in the network is made that small the overhead for transmitting them would be large. To solve this problem it is possible to alter the TDMA sending algorithm so a block of successive messages to the same destination with the same route could be transmitted without separate headers (i.e., instead of sending three messages in a row, one long message is sent). However, not only the utilization will drop with smaller slots, the scheduling task will increase in size and demand more computational power.

Clock Synchronization Example

A common real-time traffic example, e.g., telecommunication applications, have a period of $125 \mu s$. Assuming a data size of 1300 bytes per transmission, the necessary slot size for this message size is approximately $12.5 \mu s$ in Myrinet. The $12.5 \mu s$ is calculated in Equation 4.4, using a measured setup time of $3 \mu s$ (T_{setup}) for a zero copy message, the maximal clock synchronization difference between two clocks in the network ($T_{maxdrift}$) (used value of $T_{maxdrift}$ presented in Section 5.2), and the size of the message (M) in bytes divided by the transmission rate (c) for Myrinet in bytes per second.

$$T_{setup} + T_{maxdrift} + \frac{M}{c} = 3 \mu s + 1 \mu s + \frac{1300}{160} \mu s = 12.125 \mu s \quad (4.4)$$

Consider a network consisting of a 4×4 mesh of switches, where a group of end-nodes are connected to each of the 16 switches, every group consist of 16 end nodes, one sub clock-master and 15 slaves (see Figure 4.6). If the

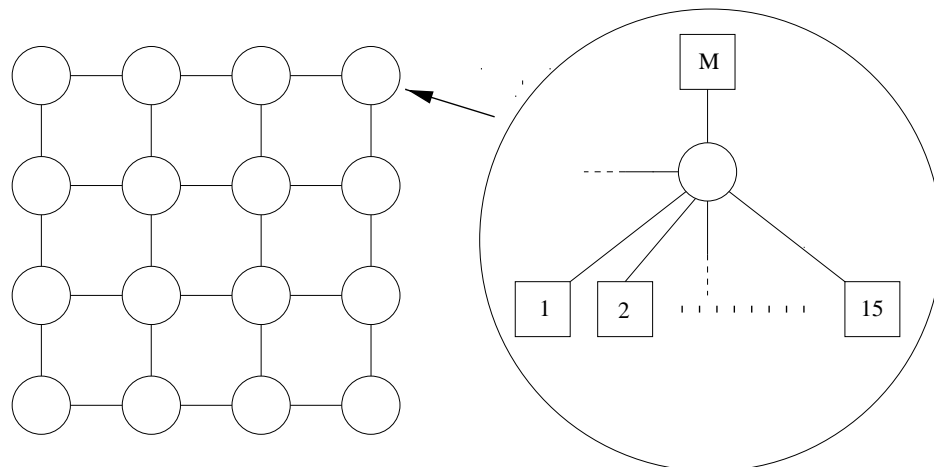


Figure 4.6: Clock synchronization in a 4×4 mesh.

network is not allowed to have a clock drift of more than $1\mu s$, a clock synchronization period of $5000\mu s$ is needed as described in Section 5.2. Previous in this section two different approaches for creating the TDMA-cycle were discussed. Using the first method (i.e., the method with the clock synchronization separated from the TDMA-cycle) the time to synchronize the whole network is calculated to be:

All sub-master nodes $15 \cdot 5\mu s = 75\mu s$, $5\mu s$ for each node, all the nodes in the network, $75\mu s + 75\mu s = 150\mu s$ (i.e., $75\mu s$ is the time for all sub-clusters synchronize in parallel), plus margin of $30\mu s$ gives a total time of $180\mu s$ ($150\mu s + 30\mu s = 180\mu s$), this results in a need of 3.6% ($\frac{180\mu s}{5000\mu s} = 3.6\%$) of the total bandwidth for clock synchronization purposes.

Using the second method (i.e., with the clock synchronization packets are scheduled among ordinary traffic) the total slots needed in order to synchronize the whole network are:

To synchronize all the sub-master nodes 15 slots are needed, plus 15 slots for each sub-master cluster (i.e., all sub-clusters synchronize in parallel) gives a total of 30 slots ($15 + 15 = 30$), and with a slot-length of $12.5\mu s$ the total time is $375\mu s$ ($30 \cdot 12.5\mu s = 375\mu s$). This results in a need of 7.5% ($\frac{375\mu s}{5000\mu s} = 7.5\%$) of the total bandwidth for clock synchronization purposes. However, ordinary traffic is allowed in this case, and the period times and deadlines for the traffic can be kept shorter.

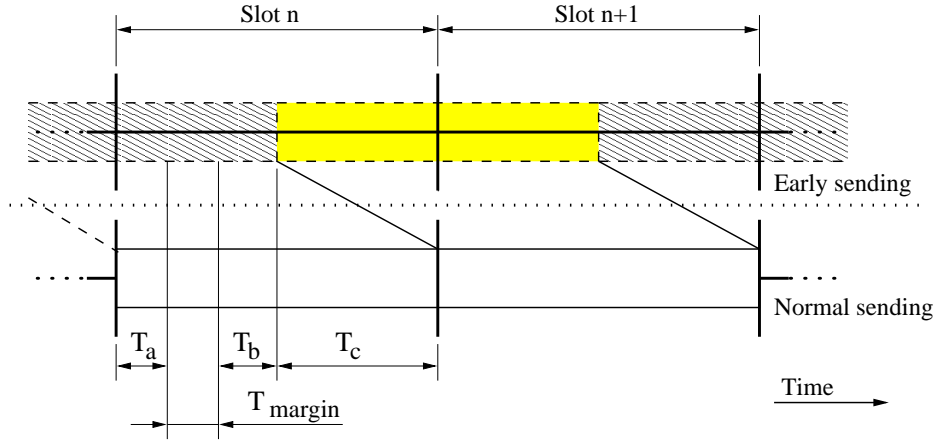


Figure 4.7: Early sending example.

4.4 Early Sending in TDMA

In networks that handle collisions between messages by blocking all but one, and not destroying them in any way, it is possible to utilize the network better by taking away all margins (see Figure 4.4). The margin before the message actually starts is not needed in a wormhole network since a blockage in the network only pause the transmission until the blockage resolves. This is also true for the latter part of previous slot (i.e., all reserved paths in the slot is transmitting or has transmitted).

If the transmission of the message to be sent in time slot $n + 1$ instead is initiated before the end of time slot n (Figure 4.7), spare time in time slot n can be used. However, the transmission of the message belonging to time slot $n + 1$ need to be delayed until it is certain that all message heads belonging to time slot n has staked their way through all switches between source and destination (delayed time: T_a). As some clock synchronization drift is allowed in the network it also must be taken into account before allowing early sending initiation. Using a margin (T_{margin}) that is larger or equal to the maximal difference ($T_{maxdrift}$) between two clocks in the network will solve the problem. In other words, the early sending can be initiated when all trunks and ports used for the transmission in time slot n are reserved. The early sending of a message belonging to slot $n + 1$ is not allowed to be initiated until a delay of T_{early} has passed from slot-start of slot n , where $T_{early} = T_a + T_{margin} \geq T_a + T_{maxdrift}$. It is most likely that a switch will stop the message belonging to time slot $n + 1$ because of elements in the network already being utilized by message belonging to the previous time slot (blocking time: T_b). As soon as messages belonging to time slot n complete their transmission, the resources will be released and the blocked

message can start its transmission (early sending time: T_c).

The latest time for the blocked message to start its transmission will be when time slot $n + 1$ start ($T_{slotstart_{n+1}}$), clock synchronization drift not encountered. The early sending method can not be used for clock synchronization messages as their total transmission time is vital (see Section 4.2).

Chapter 5

Implementation

The project implementation is limited to handle periodic traffic with hard real-time constraints, without error handling since this does not affect the results. The cluster nodes are a Pentium PCs with Ethernet and Myrinet network interfaces, where the Myrinet cards hold special designed device drivers. The PCs utilize the Linux operating system with the RT-Linux extension.

5.1 The Myrinet Control Program

To utilize as much as possible of the on-board processing capabilities, functions as the time slotting, clock synchronization between nodes and receive initialization will be handled by the on-board processor, the LANai.

Starting the state machine (Figure 5.1) in state idle, the program waits for a time-slot to start (a). When the time-slot has begun and there is data to send, the send DMA will be initialized and the program will return to the Idle state. If there is an IRQ (c) a context switch will be carried out, and the correct routine will be handled (d, e or f). If the interrupt signaled an incoming message, the receive DMA is initialized. If the incoming data signals a synchronization, the clock will be adjusted according to the chosen algorithm. Finally, if the interrupt signals that a receive transaction is completed, registers will be updated to make the host aware of the new data.

5.2 The Clock Synchronization

In the tests, we have clocked the maximum drift between two nodes to less than 100 μ s/s. Given this, and the synchronization period being less than

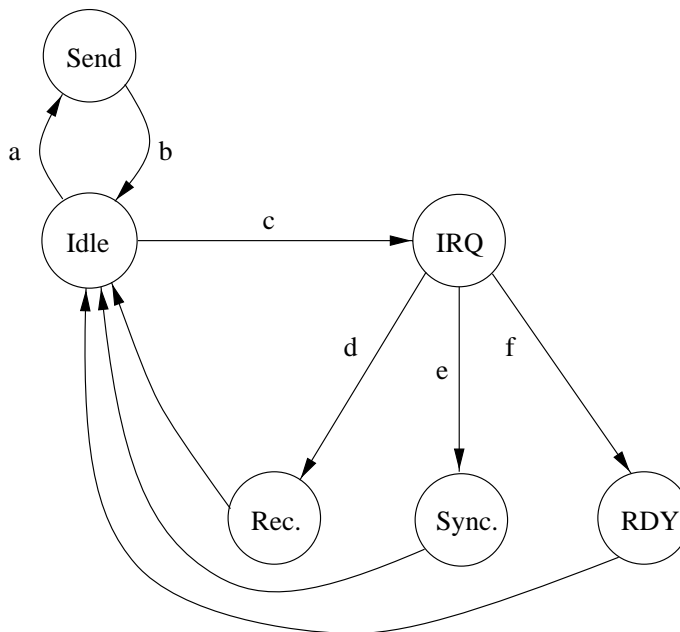


Figure 5.1: State-machine representation of the software present in the network interface.

5 ms, a theoretical accuracy of 500 ns (the clock register resolution) could be achieved by simply setting the clocks to the master node time. The master node time is sampled and introduce by this a possible error of 500 ns. This gives a total maximum drift ($T_{maxdrift}$) of 1 μs . This means that the synchronization algorithm could be less complex than the one suggested in Section 4.2 and will thereby have a smaller demand on computation power, but at the same time demand more frequent synchronizations. Assuming that synchronizations could be performed every millisecond, the variant of the synchronization algorithm described below is chosen.

Here follows an overview of how the synchronization implementation look like:

1. The synchronizing master node starts up by reading the local real-time clock (RTC) register.
2. To this value a constant is added; the constant reflects the time spent transmitting the synchronization message.
3. A synchronization message is formed and sent to a node.
4. The receiving node will write the received value in the local RTC register and thus synchronize.

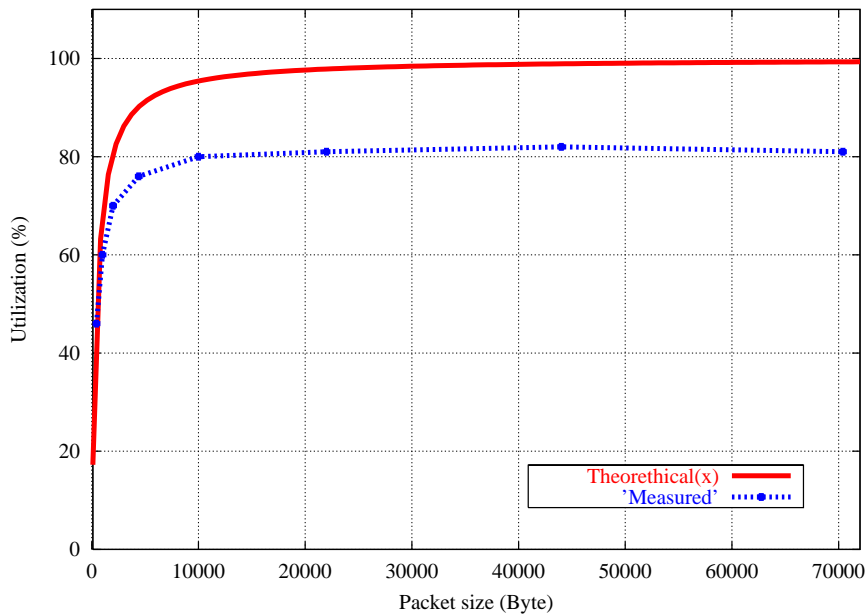


Figure 5.2: The channel utilization for different packet sizes. The theoretical curve describes the optimal performance, the measured curve describes the implemented solution.

5.3 TDMA Implementation

To keep the latencies down in the network, data streams has to be converted into several smaller packages. Each packet is then transmitted on its own, to be reassembled to a single data stream at the destination. In the conversion from the stream to packages, control information has to be added. This control information (the overhead) consists of the routing information, and the packet length (i.e., the packet header). If the data stream is divided into smaller packets, the ratio of the useful transfered data per packet will drop. On the other hand, as packets grow large, the transmission time increase and this will affect the lower limit of the deadlines. Not only the control information cause overhead, the transmission time also increases because of the time spent in the connection phase for each packet (i.e., startup time for DMA). As shown in the Figure 5.2, utilization is plotted against packet length for both the theoretical and measured case.

To model the optimal utilization of a channel (η), the connection time and the time to transmit the overhead is measured (τ) which will be the time to transmit a packet with zero bytes of data. Other parameters needed for this computation is the message length (M) and the transmission rate (c).

The optimal utilization is:

(5.1)

$$\eta = \frac{1}{\frac{\tau}{M} + 1} = \frac{1}{\frac{c\tau}{M} + 1}$$

For the TDMA implementation, a slot length has to be chosen along with the slot margins and the TDMA cycle length. According to Section 4.4, the margins could be totally removed if clock synchronization is performed so often that slots never overlap so much that synchronization packets are affected. This could be done since the clock synchronization messages utilize less than a half slot and thus does not “disappear” at any time. When it comes to the choice of slot length, as small slots as possible is wanted since this keeps down the latency. Small slot on the other hand, decrease the throughput and a mix between the both has to be found. It would be desirable with slots in the range of 10-100 μs , giving a TDMA cycle of 10-100 slots. The throughput reaches 90% of its maximum at a slot length of 30 μs or 3400 bytes as shown in Figure 5.2.

5.4 The Scheduler

Some definitions used in this section are:

- S_i Source host of message stream i .
- D_i Destination host for message stream i .
- p_i Period time of message stream i .
- d_i Separate deadline (incremented by the period of the traffic stream) associated with message stream i , or the latest time for the whole message stream i to reach D_i .
- s_i Minimum number of time slots required for message stream i .

The traffic over Myrinet using TDMA has to be scheduled in some way. In order to release the burden of the LANai it seems to be a good solution to calculate the schedule in the host. By using distributed scheduling¹ among

¹Every node in the network execute the same algorithm using the same input resulting in the same output.

with Ethernet multicasting of every node's altered bandwidth demand, the burden on the Myrinet network is reduced. The traffic over Myrinet will only consist of clock synchronization and real-time messages. As the schedule has to be available at the start of a new TDMA cycle, it would be preferable if it runs in real-time and is synchronized with Myrinet. For this purpose it is vital to use an operating system with real-time support in the host. Linux with the real-time extension RT-Linux is used in this implementation. To be sure that all nodes have finished the calculation of the new schedule before the change of schedule, Ethernet could be used for acknowledge messages. Switching of schedules can not be allowed before every node has sent their acknowledge messages.

In this implementation the routing information is statically added in advance for simplicity. However, it is possible to let a program determine the network topology with the help of the characteristic behavior of the switches and the nodes. Supposing a network where restricted reconfigurations are allowed during run-time this information has to be retrieved a a regular basis. Since our implementation will describe the system during a static period, this is not implemented. The chosen TDMA approach, according to the ones presented in Section 4.3, is the one where the clock is scheduled among with all other data packets.

All traffic handled is periodic and a separate deadline (incremented by the period of the traffic stream) is associated with each message stream. At the start of every new period the message has to be available for sending. Every message stream i is characterized by the following tuple: $\{S_i, D_i, p_i, d_i, s_i\}$. The time periods used are chosen to get a reasonable short TDMA cycle (e.g., 40 slots), The shortest deadline is one slot length, the shortest period is two slot lengths (e.g., if the slot-length is $30 \mu s$ the shortest period is $2 \cdot 30 \mu s$) since the clock synchronization messages requires at least one slot per node. The longest allowed period and deadline is as long as the TDMA cycle length (e.g., $40 \cdot 30 \mu s$). The clock synchronization period used is 40 slots (1.2 ms) and a separate deadline for the clock synchronization messages set to half the clock synchronization period, forcing the clock synchronization messages to be scheduled in the first half of the TDMA cycle. The longest elapsed time between two consecutive clock synchronizations, between the same nodes, will be maximum 1.5 clock synchronization period apart (i.e., if there are several clock synchronization periods in the same TDMA cycle). Before being scheduled all bandwidth demands are sorted according to their individual deadlines. This is an established method for making a good schedule. However, the algorithm is not designed to deliver an optimized schedule. If desired, it is possible to use any other scheduling algorithm. The created schedule runs repeatedly until a new schedule is available. To guarantee that

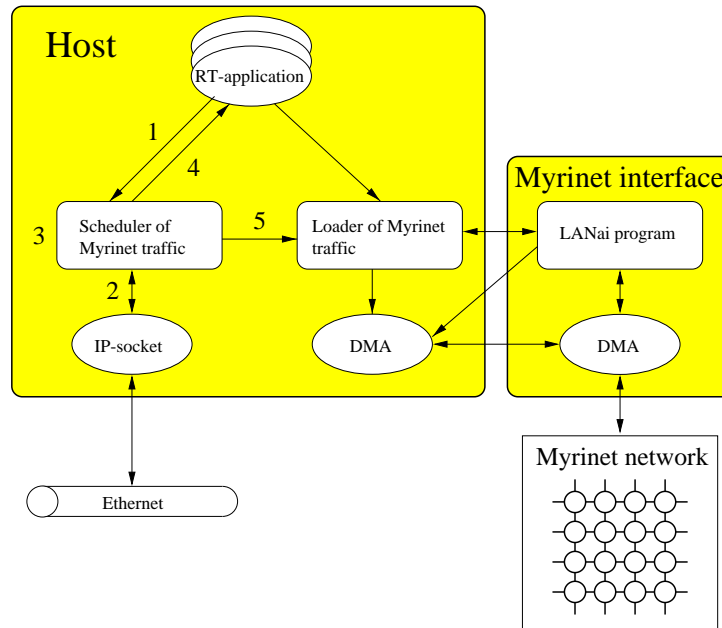


Figure 5.3: A structural overview of the system.

it does not take unpredictable time to create a new schedule, it is possible to allow a certain amount of changes for each new schedule.

The functionality of the scheduler can be divided into a number of steps:

1. Every host gather the bandwidth demands for periodic real-time traffic from the tasks that runs within it.
2. Use Ethernet via TCP/IP socket to send the node's own bandwidth demands, and receive demands from other nodes, then sort them according to their deadlines. Remove old traffic that are no longer needed according to collected messages from Ethernet.
3. Determine the new schedule (see Appendix A, for scheduling example). Schedule traffic according to their individual deadlines (i.e., schedule traffic with shorter deadline before those with longer deadlines), make sure that old traffic really gets scheduled before accepting the new schedule.
4. Inform the node's own tasks if their demands were fulfilled or if they were rejected.
5. Deliver the node's own output schedule to the task-to-network loading process.

The loader in point 5 keeps track of where in the DMA (Direct Memory Access) area the messages to be transmitted are stored, and in which time slot they are to be transmitted. After message generation, but in time before the messages are due to be transmitted, the loader sends information to the LANai about the messages location and in which time slot they are to be transmitted (Figure 5.3). The output information from the scheduler contains information about which slots belongs to which task and the path that is allowed when transmitting the message.

Chapter 6

Results

By the use of network link reservation, giving deterministic network routing, and time-slotting, hard real-time guarantees are achieved. The worst case latency is kept small (i.e., $T_{setup} + T_{maxdrift} = 3 \mu s + 1 \mu s = 4 \mu s$) as no blockages are allowed, and this makes the guaranteed throughput high.

Central parts for the performance of the network solution are the clock synchronization, the packet length, the slot margin, and the TDMA (Time Division Multiple Access) cycle length. The clock synchronization sets the lower limit of the slot-length, as well as the slot-margin. With a larger packet length the overhead will decrease, but gives a lower bandwidth utilization if the slot-size is large and not fully utilized because of a large amount of smaller packages. The TDMA-cycle length has to be at least as long as the clock synchronization period i.e., in Myrinet the longest clock synchronization time period is 5 ms for an accuracy of 1 μs .

The Clock-Synchronization

In our test the clocked drift between two nodes is shown to be less than $\pm 100 \mu s/s$. Given this, and the synchronization period being less than 5 ms, an accuracy of 1 μs is achieved as shown in Section 5.2. The synchronization algorithm can be less complex than the one suggested in Section 4.2, it has a smaller demand on computation power, but at the same time it needs more frequent synchronizations. With clock synchronization every 5 ms, the chosen synchronization algorithm in Section 5.2, performs a clock synchronization accuracy of 1 μs , and this is sufficient for the management of slot-lengths down to of 5 μs (i.e., the smallest time required to transmit a clock synchronization packet).

Chapter 7

Discussion

In this thesis, a method to implement hard real-time services over generic switched networks is presented and tested. By using resource allocation combined with a time-slotting mechanism, blockages and consequently deadlocks are avoided in the network. When no blockages occur, the determined worst case transmission time only depends on delays in the network hardware, and is not affected by other traffic in the network. Consequently it is possible to guarantee the delivery for messages even if their deadlines are short. In other reported work where only software is used where to guarantee real-time traffic over a generic network, the worst case transmission time includes all possible blockages that can occur in each switch (Section 4.1). To reduce the worst case transmission time, special routing techniques are often used (e.g., X-Y routing). In the implementation presented in Chapter 5, no such routing technique is required. Instead, the exact topology over the network (i.e., a switch graph where the place of each node and link are known) is necessary. The scheduler uses the scheme when reserving the network links. Since the scheduler has complete control over the network link usage, no blockages will occur, by this a shorter time for packet transmission can be guaranteed compared to the admission control scheme proposed in [4]. Also, a high bit rate can be guaranteed for a given connection. However, the scheduling algorithm gets more heavy to execute, especially when the nodes and the links in the network increase.

The slot margins mentioned in Section 4.3, are not needed in network that support round-robin in the switches (e.g., Myrinet) as long as the slots are longer than the required time for a clock synchronization message ($5 \mu\text{s}$ in Myrinet). The message will be transmitted in a certain time interval even if it is blocked. But in networks that does not support round-robin the slot margins are vital to keep the guarantees. By using slot margins our solution can be transferred to any network supporting source-routing. For example,

the network described in Bettati and Nica [16] can be exchanged with other round-robin networks only. Our solution on the other hand, does not have this requirement.

7.1 Other Implementations

Not all networks have a programmable network interface card (NIC) as used in the method implemented in Chapter 5. The program in the NIC can be implemented in the host computer instead. This makes it possible to replace the Myrinet network with e.g., a switched Gigabit Ethernet network, but the disadvantage is that the host computer will spend more CPU-time on network traffic tasks.

7.2 Conclusion – Future Work

The implementation presented in Chapter 5 is limited to handle periodic traffic, and the tests has only been performed with a couple of Pentium PCs in the network. It is not possible to mix the implemented Myrinet communication protocol with other available User Level Protocols for non real-time traffic (e.g, GM, LFC, BIP in Chapter 3). It is possible to solve the transmission of aperiodic, isochronous, and non real-time traffic. One method is to reserve periodic time-slots to guarantee transmission of aperiodic or isochronous traffic when they arrive; however, the bandwidth utilization will be reduced, since not all reserved slots will be used. To handle non real-time traffic where e.g., GM is used, a similar method can be used by reserving an amount of continuous TDMA-slots. The implementation of fault tolerance and fault handling is omitted to future work, since this does not affect the results at the current level of implementation. Since there are many methods, and ongoing research on scheduling techniques for real-time traffic, no efforts on developing an optimized scheduler has been made. Instead a simple scheduling algorithm is used in order to fulfill the tests.

Another future task is to test the solution in a network with more nodes, switches, and more powerful computers. Furthermore, tests using different topologies, and perhaps mixing different operating system on the end nodes in the network can be carried out.

Bibliography

- [1] S. Kambhatla. Hypercube vs Cube-connected Cycles: A Topological Evaluation. Oregon Graduate Institute of Science and Technology, April 1997.
- [2] Myricom Inc. *Myricom Home Page*, September 1999. <http://www.myri.com>.
- [3] J. Duato, S. Yalmanchili, and L. Ni. *Interconnection Networks: an Engineering Approach*, pages 13–20. IEEE Computer Society Press, 1997.
- [4] S. Sundaresan and R. Bettati. Distributed Connection Management for Real-Time Communication over Wormhole-Routed Networks. In *Proc. of the 17th Int. Conference on Distributed Computing Systems*, pages 209–216, May 1997.
- [5] J.-P. Li and M.W. Mutka. Priority Based Real-Time Communication for Large Scale Wormhole Networks. In Howard Jay Siegel, editor, *Proc. of the IEEE 8th Int. Parallel Processing Symposium (IPPS '94)*, pages 433–438, Los Alamitos, CA, USA, April 1994. IEEE Computer Society Press.
- [6] S.L. Hary and F. Özgüner. Real-Time Interprocessor Communication for Point-to-Point Networks Using Wormhole Routing. In *Proc. of the 4th Int. Workshop on Parallel and Distributed Real-Time Systems*, pages 157–163, 1996.
- [7] H. Song, B. Kwon, and H. Yoon. Throttle and Preempt: A New Flow Control for Real-Time Communications in Wormhole Networks. In *Proc. of the ICPP '97, 1997 Int. Conference on Parallel Processing*, pages 198–202, Washington - Brussels - Tokyo, August 1997. IEEE Computer Society Press.

- [8] K.H. Connelly and A.A. Chien. FM-QoS: Real-time communication using self-synchronizing schedules. In ACM, editor, *SC'97: High Performance Networking and Computing: Proc. of the 1997 ACM/IEEE SC97 Conference: November 15–21, 1997, San Jose, California, USA.*, pages ??–??, New York, NY 10036, USA and 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1997. ACM Press and IEEE Computer Society Press.
- [9] R.A.F. Bhoedjang, T. Rühl, and H.E. Bal. User-Level Network Interface Protocols. *Computer*, 31(11):53–60, November 1998.
- [10] D.D. Kandlur, K.G. Shin, and D. Ferrari. Real-Time Communication in Multihop Networks. *IEEE Transactions on Parallel and Distributed Systems*, 5(10):1044–1056, October 1994.
- [11] M. Jonsson, K. Börjesson, and M. Legardt. Dynamic Time-Deterministic Traffic in a Fiber-optic WDM star Network. In *Proc. of the 9th Euromicro Workshop on Real Time Systems*, pages 25–33, June 1997.
- [12] J. Jonsson and J. Vasell. Implementation of a Time-Deterministic Communication Chip. Technical Report 206, CTH, Dept. of Computer Engineering, Computer Architecture Laboratory (CAL), MMP, 1995.
- [13] B. Kim, J. Kim, S. Hong, and S. Lee. A Real-Time Communication Method for Wormhole Switching Networks. In *Proc. of the Int. Conference on Parallel Processing*, 1998.
- [14] H. Hansson, M Sjödin, and K. Tindell. Guaranteeing Real-Time Traffic Through an ATM Network. In *Proc. of the 30'th Hawaii Int. Conference on System Sciences*, volume 5, pages 44–53. IEEE Computer Society Press, 1997.
- [15] S. Norden, G. Manimaran, and C.S.R. Murthy. New Protocols for Hard Real-time Communication in the Switched LAN Environment. In *Proc. of the 23rd Annual Conference on Local Computer Networks*, oct 1998. Boston, MA, USA.
- [16] R. Bettati and A. Nica. Real-Time Networking over HIPPI. In *Proc. of the Fourth Workshop on Parallel and Distributed Real-Time Systems*, Santa Barbara, California, April 1995.

- [17] N. Boden, D. Cohen, R.E. Felderman, A.E. Kulawik, C.L. Seitz, J.N. Seizovic, and Wen-King Su. MyriNET — A Gigabit-per-Second Local-Area Network. *IEEE-Micro*, 15(1):29–36, February 1995.
- [18] T. Einstein. RACEway Interlink — A Real-Time Multicomputing Interconnect Fabric for High-Performance VMEbus-Systems. VMEbus Systems, Mercury Computer Systems Inc, 1996.
- [19] Myricom Inc. *Myricom GM Myrinet Documentation*, 1998. <http://www.myri.com>.
- [20] High Speed Networks and Cooperative Applications. *BIP Documentation*, 1999. <http://lhpc.univ-lyon1.fr>.
- [21] L. Clarke, I. Glendinning, and R. Hempel. The MPI Message Passing Interface Standard. In K.M. Decker and R.M. Rehmman, editors, *Programming environments for massively parallel distributed systems*, pages 213–218, Boston, MA, USA, 1994. Birkhäuser.
- [22] A. Kanevsky, A. Skjellum, and A. Rounbehler. Real-Time Extensions to the Message-Passing Interface (MPI). The MITRE Corp., January 1997.
- [23] A. Kanevsky, A. Skjellum, and A. Rounbehler. MPI/RT — An Emerging Standard for High-Performance Real-Time Systems. In *Proc. 31st Annual Hawaii Int. Conference on System Sciences*, pages 157–166, 1998.
- [24] M. Barabanov and V. Yodaiken. Introducing Real-Time Linux. *Linux Journal*, 34, February 1997.

Appendix A

Scheduling example

Some definitions used in this section are:

S_i	Source host of message stream i .
D_i	Destination host for message stream i .
p_i	Period time of message stream i .
d_i	Separate deadline (incremented by the period of the traffic stream) associated with message stream i , or the latest time for the whole message stream i to reach D_i .
s_i	Minimum number of time slots required for message stream i .
R0-R5	full duplex network links.
RX_o	o stands for output line to node X .
RX_{in}	in stands for input line to node X .
RX_R	R stands for right transmitting line between two switches.
RX_L	L stands for left transmitting line between two switches.

In a scheduling example for topologies according to Figure A.1, given the network workload presented in Table A.1, a schedule is calculated for each topology using a simple, non-optimizing scheduling algorithm. The first message of each stream is assumed to be generated at the beginning of slot 1. The implemented schedule focus on the possibility of keeping control over the usage of every network link, as well as exploring the possibility of using an alternate path if the first hand choice path is unavailable.

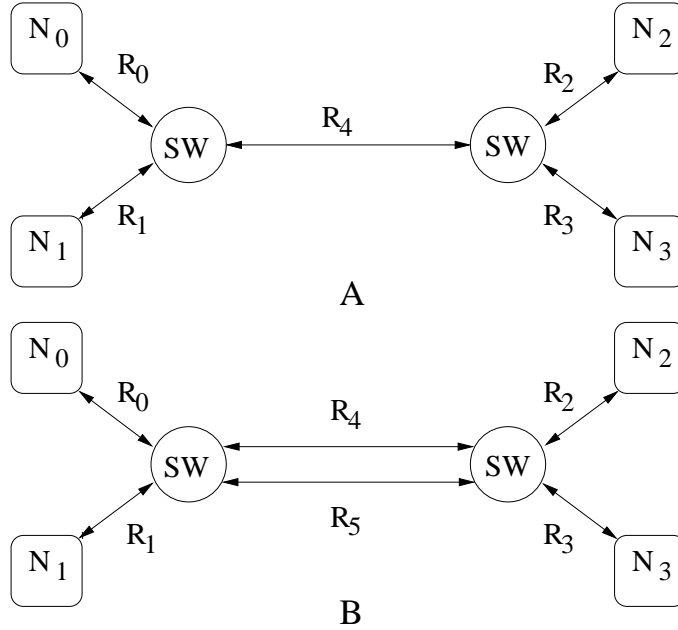


Figure A.1: The two setups used for the scheduling example. A is a network without redundant paths, and B is a network with redundant paths.

Every node has its allowed set of message stream identification numbers (i), node 0 (clock master node in this example) has the message stream identification number set 1 - 30, node 1 has the set 31 - 60, node 2 has the set 61 - 90, and node 3 has the set 91 - 120. The number of different message streams allowed from each node in this example are 30, but only a few are used as many of them are bandwidth demanding. The total number of message streams in this example are 18. The TDMA cycle is 40 slots (calculated using least common multiple). The message streams 1 - 3 are the clock synchronization messages. A separate deadline for the clock synchronization messages set to half the clock synchronization period force them to be scheduled in the first half of the TDMA cycle. This to have the longest elapsed time between two following clock synchronizations, between the same nodes, at maximum 1.5 clock synchronization period apart (e.g., if the TDMA cycle is longer than the clock synchronization period).

When running the scheduler using the topology A in Figure A.1. Message streams 31, 32, and 34 are not schedulable, when switching to topology B all message streams are schedulable as an alternate path is available. The first hand choice of path is used as far as it is possible before alternate paths are explored. The implemented scheduling algorithm focus on keeping down the required time to create a new schedule, instead of optimizing it.

i	S_i	D_i	p_i	d_i	s_i
31	1	2	10	8	1
34	1	3	10	8	2
91	3	2	10	8	1
32	1	3	10	8	2
93	3	1	10	10	2
62	2	0	10	10	2
63	2	3	20	10	2
13	0	3	10	10	1
33	1	2	20	12	2
12	0	1	20	15	3
61	2	0	20	16	4
35	1	3	20	16	3
11	0	2	20	16	3
2	0	2	40	20	1
1	0	1	40	20	1
92	3	0	20	20	4
3	0	3	40	20	1
14	0	2	20	20	3

Table A.1: Network workload for scheduling example.

slot	R0 _o	R0 _{in}	R1 _o	R1 _{in}	R2 _o	R2 _{in}	R3 _o	R3 _{in}	R4 _R	R4 _L
1	12	62		12	62					62
2	12	62		12	62	91	91			62
3	12			12						
4				93			93			93
5				93			93			93
6	1		33	1	63	33		63	33	
7			33		63	33		63	33	
8	13							13	13	
9		61	35		61			35	35	
10		61	35		61			35	35	
11		61	35		61	91	91	35	35	
12	11	61		93	61	11	93		11	93
13	11			93		11	93		11	93
14	11	92				11	92		11	92
15	2	92				2	92		2	92
16	3	92					92	3	3	92
17	14	92				14	92		14	92
18	14	62			62	14			14	62
19	14	62			62	14			14	62
20	13							13	13	
21				93	63	91	91	63		
22				93	63		93	63		93
23	13						93	13	13	93
24				12						
25	12	62	33	12	62	33			33	62
26	12	62	33	12	62	33			33	62
27	12									
28		61	35		61			35	35	
29		61	35		61			35	35	
30		61	35		61			35	35	
31	11	61			61	11			11	
32	11	92				11	92		11	92
33	11	92				11	92		11	92
34	14	92				14	92		14	92
35	14	92				14	92		14	92
36	14	62			62	14			14	62
37	13	62			62	91	91	13	13	62
38				93			93			93
39				93			93			93
40										

Table A.2: The resulting schedule for topology A; messages 31, 32, and 34 are not schedulable.

slot	R0 _o	R0 _{in}	R1 _o	R1 _{in}	R2 _o	R2 _{in}	R3 _o	R3 _{in}	R4 _R	R4 _L	R5 _R	R5 _L
1	12	62	31	12	62	31			31	62		
2	12	62	34	12	62	91	91	34	34	62		
3	12		34	12				34	34			
4			32	93			93	32	32	93		
5			32	93			93	32	32	93		
6	1		33	1	63	33		63	33			
7			33		63	33		63	33			
8	13							13	13			
9		61	35		61			35	35			
10		61	35		61			35	35			
11		61	35		61	91	91	35	35			
12	11	61	34	93	61	11	93	34	11	93	34	
13	11		34	93		11	93	34	11	93	34	
14	11	92	32			11	92	32	11	92	32	
15	2	92	32			2	92	32	2	92	32	
16	3	92	31			31	92	3	3	92	31	
17	14	92				14	92		14	92		
18	14	62			62	14			14	62		
19	14	62			62	14			14	62		
20	13							13	13			
21	12	62	31	12	62	31			31	62		
22	12	62	34	12	62	91	91	34	34	62		
23	12		34	12				34	34			
24			32	93			93	32	32	93		
25			32	93			93	32	32	93		
26			33		63	33		63	33			
27			33		63	33		63	33			
28	13							13	13			
29		61	35		61			35	35			
30		61	35		61			35	35			
31		61	35		61	91	91	35	35			
32	11	61	34	93	61	11	93	34	11	93	34	
33	11		34	93		11	93	34	11	93	34	
34	11	92	32			11	92	32	11	92	32	
35	14	92	32			14	92	32	14	92	32	
36	14	92				14	92		14	92		
37	14	92				14	92		14	92		
38		62	31		62	31			31	62		
39	13	62			62			13	13	62		
40												

Table A.3: The resulting schedule for topology B; all message streams are schedulable, the first choice path (using link R4) is more frequent used than the second choice path (link R5).

