Design and Implementation of a SCI-based Real-Time CORBA

Stefan Lankes, Michael Pfeiffer, Thomas Bemmerl
Lehrstuhl für Betriebssysteme, RWTH Aachen
Kopernikusstr. 16, 52056 Aachen, Germany
E-mail: contact@lfbs.rwth-aachen.de

Abstract

Keywords: *Real-Time CORBA, minimumCORBA, RTOS, embedded systems, real-time networks, Scalable Coherent Interface (SCI)*

The Real-Time CORBA and minimumCORBA specifications in the forthcoming CORBA 3.0 standard are important steps towards defining standard-based middleware which can satisfy real-time requirements in an embedded system. This article describes these new specifications and an implementation called ROFES.

ROFES supports different network architectures, for example the Scalable Coherent Interface (SCI). Furthermore this article examines the SCI-network and whether it possesses real-time characteristics.

1. Introduction

A real-time system is defined as a system whose correctness does not only depend on the logical results of computation, but also on the time at which these results are produced [1]. This leads to the question what the implications of a real-time application — which does not meet the timing requirements of the environment — are. This should be part of the specification of a real-time system which consists of two possibilities:

1. Violation of a particular timing constraint leads to system failures. A real-time system with such a property is classified as a *hard real-time system*.

2. Failure of meeting a particular timing requirement can be tolerated and the real-time system continues its operation even though with reduced quality and degraded performance. A real-time system with such a property is classified as a *soft real-time system*.

First generation real-time applications were running on single processor environments since the problems to be solved were relatively simple. Nowadays applications like avionics, telecommunication, process control and distributed interactive simulation need real-time properties in a distributed environment [6]. Middlewares like *CORBA* [13] and *DCOM*[1] help to improve the flexibility, extensibility, maintainability and reuseability of distributed applications. But these middleware architectures cannot be used to build an avionics mission control application because they do not support real-time features. Therefore the *Object Management Group* develops a specification for CORBA with a real-time extension [15]. A first implementation of a Real-Time CORBA is *TAO*[2] from the *Center for Distributed Object Computing* at Washington University, but the development of real-time middleware is still in progress.

Many real-time applications run on embedded systems. However, CORBA is too complex to meet the exact code size and performance requirements for such applications. This scenario requires a cut-down version of CORBA, which has been specified by the *Object Management Group* [14] and is called *minimumCORBA*. We have implemented a prototype of Real-Time CORBA which based on the Real-Time CORBA and minimumCORBA specification, with the aim to minimize the memory footprint and to support real-time properties. In particular our version of Real-Time CORBA is suitable for embedded systems. Therefore our project is called *Real-Time CORBA for embedded Systems*[3] (ROFES).

Besides Ethernet, *ROFES* supports other network architectures. By supporting these new network architectures we hope to achieve better real-time characteristics. One of these network architectures is the *Scaleable Coherent Interface (SCI)*.

A real-time operating system provides ideal support for real-time applications. However, the cost aspect and the acceptance of operating systems like Windows NT, Sun Solaris and Linux have generated a need for real-time functionality in these operating systems. For instance Solaris

[1] http://www.microsoft.com/com/tech/dcom.asp
[2] http://www.cs.wustl.edu/˜schmidt/TAO.html
[3] http://www.lfbs.rwth-aachen.de/˜stefan/rofes

is compatible with the real-time extension POSIX 1003.1b, although it is not possible to develop a hard real-time application in such an environment. Many companies and universities are developing real-time extensions for Windows NT [20] [16] and Linux. A very interesting real-time extension for Linux is *Real-Time Linux* [2]. In Real-Time Linux a small hard-realtime kernel and standard Linux run on one or more processors, in order to build a system that can be used for applications like data acquisition, control and robotics while still serving as a standard Linux workstation. We plan to implement a version of Real-Time CORBA for embedded systems on Linux, Windows CE/NT, Real-Time Linux, LynxOS and the QNX Realtime Platform.

This article is organized as follows: Section 2 explains the new Real-Time CORBA specification. Before the actual implementation of ROFES is described in the succeeding section 4, the article presents the *Scalable Coherent Interface* in section 3, which is supported by ROFES. Section 5 presents and discusses our first results. Finally, some general assessments of the lessons learned are provided and some conclusions are drawn in section 6.

2. Real-Time CORBA

It is not possible to describe the whole Real-Time CORBA specification in this paper, but it explains the most important components. A detailed description of the Real-Time CORBA specification is given in [15] and [18].

A standard ORB handles all requests of clients in a unique way. This way of processing is of course not suitable for real-time ORBs. Generally, different clients which operate with different priorities send several requests to the server. A multithreaded server possesses a pool of threads that process the incoming requests. At which priority does the server process the individual requests? This question becomes even more complicated, if several threads operate in one client and have different priorities. In order to solve this problem, the OMG defines two models in the Real-Time CORBA specification. One model allows a server to dictate the priority at which an invocation made on a particular object will be executed. In such a model the priority is specified *a priori* by the server. The server encodes the priority of the object in its reference, which is then published to the clients. The OMG calls this model *server declared*. It is useful for certain real-time applications, but some applications require a dynamic priority specification. For such applications the OMG specifies a second model called *client propagated*. If the target object supports the client propagated model, the priority is carried with the invocation and the server processes the incoming request at the priority of the client thread, which originally invoked the operation.

Client and server can run on different operating systems and use different native thread priority schemes. For instance, Linux uses 100 and LynxOS 256 different priority classes. For the client propagated and the server declared priority model Real-Time CORBA needs a priority model independent from the operating system. The OMG defines a logical priority which has a system-wide uniform representation called *CORBA Priority*. A real-time ORB provides a priority mapping between CORBA priorities and native priorities for each supported platform. Furthermore a real-time ORB shall provide a mechanism to allow users to overwrite this default priority mapping.

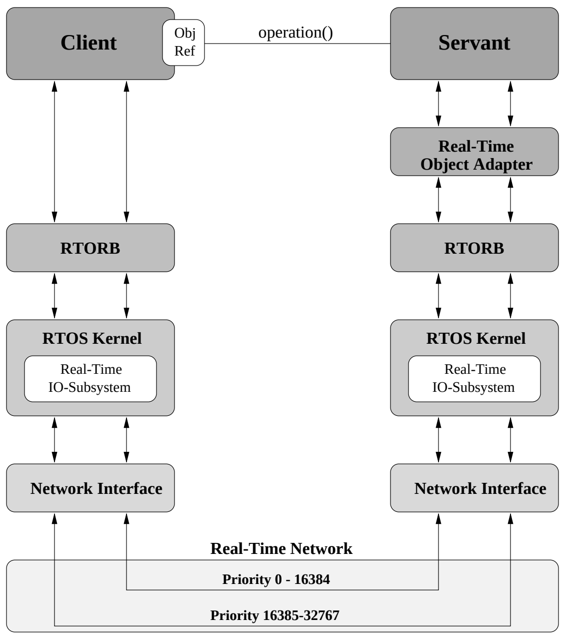

Figure 1. Priority Banded Connection

The original CORBA specification supports only implicit bindings. This model establishes resources on demand. For real-time applications with deterministic QoS *(Quality of Service)* requirements these implicit bindings are inadequate. Real-Time CORBA defines an explicit binding mechanism which improves the performance and predictability of the invocations. One of the explicit binding mechanisms is the *priority banded connection* model. This facility allows a client to communicate with the server via multiple transport connections — each dedicated for carrying invocations with different CORBA priority or range of priorities, as shown in Figure 1. A client establishes a priority banded connection by sending the *_bind_priority_band* request to the server which specifies the range of priorities the connection will be used for. This allows the server to allocate the necessary resources. The selection of the appropriate connection for each invocation is transparent to the application and is done by the ORB based on the value of the priority model (server declared or client propagated).

3. The Scalable Coherent Interface

3.1. Description of the Interconnect

Besides Ethernet *ROFES* supports the *Scaleable Coherent Interface (SCI)* [8]. SCI is a point-to-point, ring-based interconnect that can be configured in various switched-ring topologies such as rings and tori and falls into the class of the so-called *SANs (System Area Networks)* [12]: a high performance network for cluster systems. In distinction to other SANs (e.g. Myrinet or ServerNet) which only allow message-passing style communication, SCI offers transparent read and write remote memory access. Memory segments of each compute node can be mapped into the virtual address space of all cluster compute nodes and be used to assemble globally shared data structures. Accesses to segments of memory that are physically located on remote compute nodes are transparently mapped to the address range of the SCI adapter and served via a respective network transaction. Figure 2 illustrates the SCI-network. One vendor of SCI adapter cards is the Norwegian company *Dolphin Interconnect Solutions*[4]. Dolphin wants to place its adapter cards in the real-time market. Therefore, Dolphin is developing drivers for the real-time operating systems LynxOS and VxWorks. Dolphin's PMC-SCI adapter card provides a high reliable throughput, low latency, transparent memory-mapped link between PCI/PMC based workstations and embedded systems like VMEbus computers. Thus SCI can be used in an embedded system environment.

The basic performance characteristics of the SCI network was evaluated on our test platform[5]. The memory bandwidth amounts to 1.9 MByte/s for read and 83.6 MByte/s for write access. For a real-time system the average latency and its delay jitter[6] are substantially more interesting. These results are summarized in table 1. To increase remote memory access throughput, the Dolphin PCI-SCI adapter cards possess so-called stream buffers for reading and writing. Each stream buffer, which is 64 byte wide, can manage a separate pending transaction. Write stream buffers may delay the network transaction to allow to combine several write accesses into larger transaction as long as they are consecutive in memory. If a write access aligns to 64 byte, the write access does not delay in a stream buffer. Therefore, the average latency is smaller than for an unaligned write access. For read operations, the stream buffers can exploit prefetching, i.e. the whole 64-byte memory range is fetched on a read access not just the requested datum. The small average latency and delay jitter promise, that SCI is an ideal network for soft real-time systems. However, is SCI suitable for hard real-time applications?

[4]http://www.dolphinics.no

[5]The test platform is described in section 5.1.

[6]A delay jitter is the difference between the maximum and minimum values of the delay.

Table 1. Characteristics of the SCI-Network

message type	*operation*	*average latency*	*delay jitter*
char	write[a]	2.1 μs	6 μs
char	write[b]	2.1 μs	6 μs
char	read	4.6 μs	8 μs
int	write[a]	27.1 μs	13 μs
int	write[b]	2.1 μs	6 μs
int	read	4.6 μs	12 μs
double	write[a]	27.4 μs	12 μs
double	write[b]	2.5 μs	10 μs
double	read	5.0 μs	9 μs

[a]unaligned write operations

[b]64 byte aligned write operations

3.2. SCI as a real-time network

The real-time network is the core of a real-time cluster and has to provide the following services to the nodes in the cluster [11]:

- Reliable and temporally predictable message transmission with low latency and minimal latency jitter,
- support of fault-tolerance to handle replicated nodes and replicated communication channels,
- clock synchronization in the range of microseconds and
- membership service with low latency for detecting node failures.

This implicates that a hard real-time application cannot use TCP/IP over Ethernet because the communication over Ethernet is nondeterministic. However, the Ethernet network has a high acceptance in the industry and the costs are very low. So for cost reasons soft real-time applications often use the Ethernet technology.

But the question is if SCI is suitable for hard real-time applications. To answer this, the structure of a SCI interconnect must be examined (see also [17]). The simplest configuration of an SCI interconnect is a ring traversing all nodes. The ring is based on the architecture of a SCI adapter card illustrated in figure 3. Incoming packets to the SCI

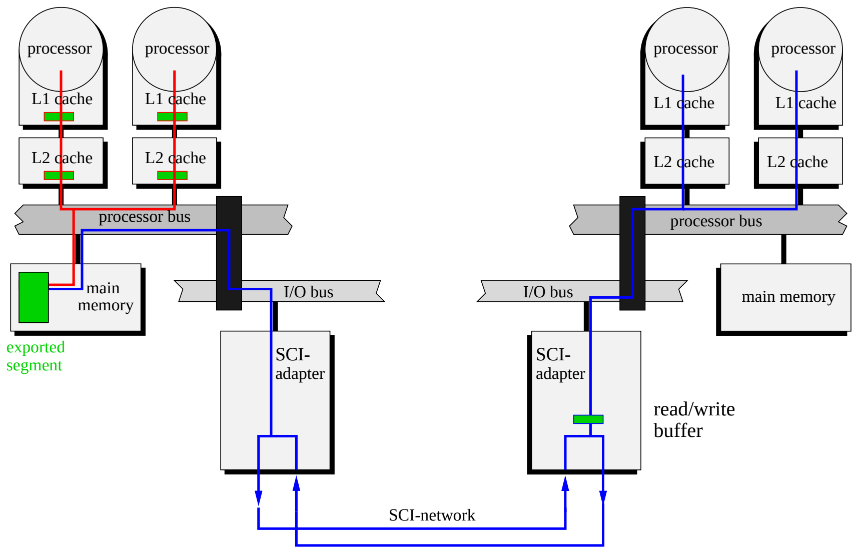

Figure 2. The system architecture of a SCI-Cluster

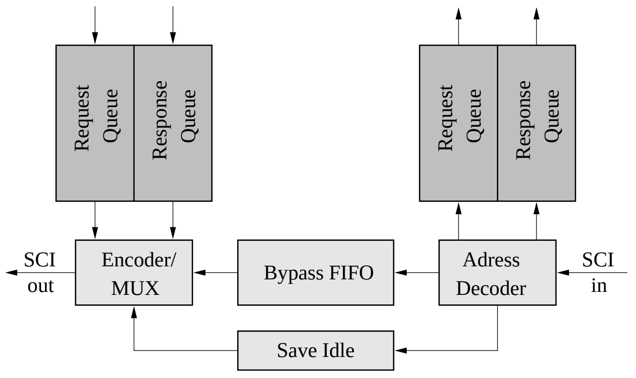

Figure 3. Model of a SCI Adapter Card

adapter card pass through an address decoder. If the packet is destined for the local node, the decoder places it into the request or response input queue. If the packet is destined for another downstream node, it is forwarded to the bypass FIFO. To output a packet, the SCI node must have sufficient free space in its bypass FIFO to hold all incoming packets. When there are no packets waiting in the output queue or there is insufficient free space in the bypass FIFO for the output queue data to be sent, data from the bypass FIFO is transmitted on the node's output link. If the bypass and output queue are empty, *idle* symbols are transmitted.

SCI does not know prioritized transactions. Thus it has a weakness when applied to real-time applications, for which prioritized transactions require priority-dependent minimal bandwith and maximum latency guarantees. Hence a priority inversion can occur in the FIFOs and queues of the SCI adapter. In [9] the *SCI/RT Working Group* defines a solution for this problem called *Scalable Coherent Interface for Real-Time Applications (SCI/RT)*. SCI/RT has the following features:

- a priority-based preemptive arbitration and queueing discipline protocol
- support for both priority-based shared memory and message passing architectures
- a standard global clock synchronization method
- a standard event notification method
- single-bit hard error detection and correction capability
- hardware subaction fault-retry protocol support

In order to realize these features all FIFOs and queues in figure 3 must be replaced by priority queues. Unfortunately a vendor of SCI/RT adapter cards does not exist.

TCP/IP over Ethernet has nearly the same problems. By sending messages over Ethernet, the driver builds a list of Ethernet-packets. The Ethernet adapter card gets the first element of the list via DMA-Transfer and sends it to the receiver. Therefore the packets are dispatched in FIFO order and a priority inversion can occur. In order to obtain a high bandwidth, the driver builds very large Ethernet-packets. In a SCI network the packets are built by the SCI adapter card. The SCI adapter card produces the packet substantially faster. This is one reason why SCI uses smaller packet sizes. Since SCI-packets are substantially smaller than Ethernet-packets, a priority inversion does not have such negative effects. Therefore SCI is more suitable for soft real-time applications than Ethernet.

4. Design of a SCI-based Real-Time CORBA

The SCI-based communication architecture is based on the *General Inter-ORB Protocol* (GIOP) 1.2. In order to obtain optimum performance it must be guaranteed that a process only writes to a remote SCI segment because read accesses are extremely expensive.

When for instance, a client creates a connection to a server, then the client and the server create a local SCI segment and map it into their address spaces. These local segments are the *receive buffers* of server and client and they expect messages from the other process in these buffers. Then they map the *receive buffers* of the other process and use them as their *send buffers*. Therefore the *send buffer* is a remote SCI segment (see figure 4). Now the client transmits a request message to the server by writing this message to its *send buffer*. The *send buffer* is the *receive buffer* of the server. Therefore the client writes the request message into the *receive buffer* of the server. Afterwards, the client has to signalize the server that there is a valid message in its *receive buffer*. In order to signalize this, the client sends a SCI interrupt to the server. This procedure wakes up the server, copies the message from the SCI segment to a local buffer and signalizes the client that the server got the message. Now the client can send other messages to the server, while the server can analyze the message and send an answer to the client's request message.

In *ROFES* the size of the *send* and *receive buffer* can be adjusted. Thereby the user can adapt *ROFES* to his individual requests. The default size of the *send* and *receive buffer* is 64 KByte.

For priority banded connections the server and the clients build several connections (see figure 4). The client chooses the connection according to the actual priority. If a message is larger than the buffer size, the message is divided into several smaller messages. This is specified in GIOP 1.2.

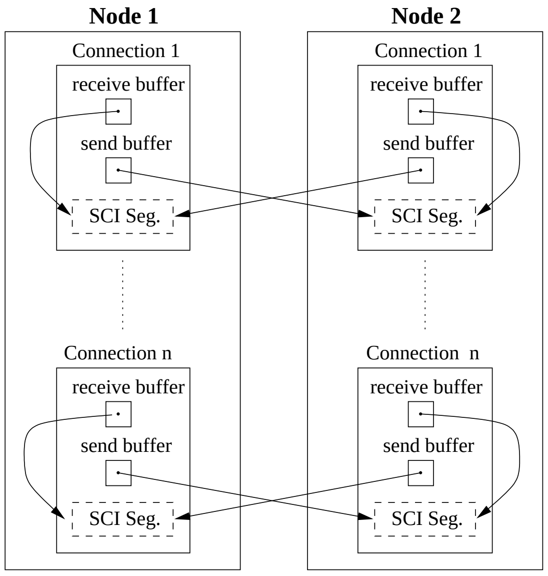

Figure 4. SCI-based CORBA Connection

5. Performance Evaluation

5.1. Performance Evaluation of a Priority Banded Connection

The benchmark in this section compares the performance of a normal and a priority banded connection. On the client side, a single high-priority thread and a variable number of low-priority threads run concurrently. The high-priority threads invoke several requests with variable argument sizes. The low-priority threads invoke permanent requests, which have no arguments. On the side of the server, a servant is created and configured to process the client requests at the same priority at which were originally invoked the operation on the client side. This means that the benchmark uses the client propagated priority model. The low-priority threads use the CORBA priorities 8192, 8446, 8704, 8960, 9216 and 9472, while the high-priority thread uses the CORBA priority 16384. The test platform consists of two 400 MHz Pentium II systems with 128 MBytes RAM. As operating system *LynxOS 3.0.1*[7] is used for these computers. The systems are connected with 100 Mbps Ethernet devices and Dolphin's PCI-SCI adapter cards based on LC2 SCI Link Controller and PSB-32 PCI-SCI Bridge. The test platform is not an embedded system. However the memory footprints of the benchmarks are very low, the results should also be meaningful to the results on an embedded system.

To get high bandwidth over Ethernet the TCP service layer buffers several outgoing messages internally and then passes these messages as one large message to the next lower layer for transmission. This mechanism – often called *Nagel* algorithm – increases the average latency and the jitter of a transmission. In order to achieve a jitter as small as possible, the *Nagel* algorithm is turned off in our CORBA implementation. This means that packets are always sent as soon as possible and no unnecessary delays are introduced.

The priority banded and the normal solution are benchmarked with different numbers of low-priority client threads. The experiment was repeated 100.000 times for each solution, each number of client threads and each argument size of the high-priority requests. Figure 5 shows the results of the normal solution using the TCP/IP protocol over a 100 Mbps Ethernet device. The figure shows clearly that the jitter of this solution becomes very large. This con-

[7]http://www.lynuxworks.com

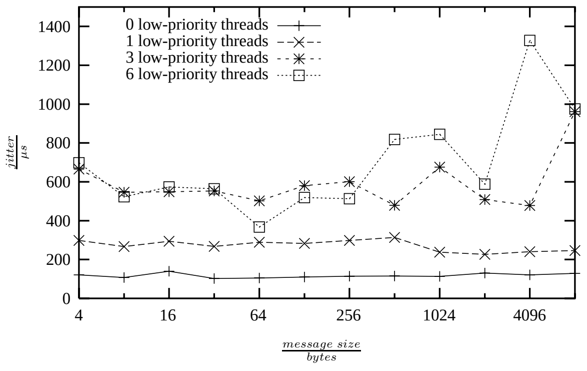

Figure 5. Jitter of a high-priority thread in the normal solution using Fast Ethernet

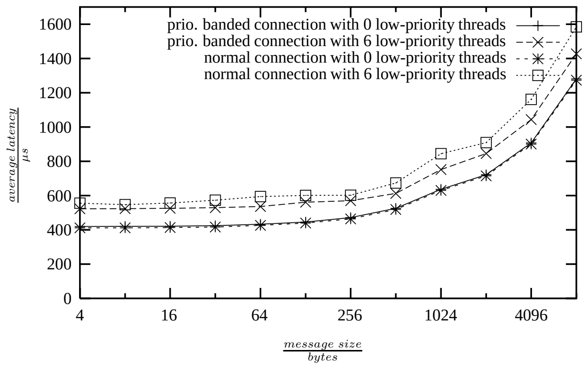

Figure 7. Average Latency of the high-priority thread using Fast Ethernet

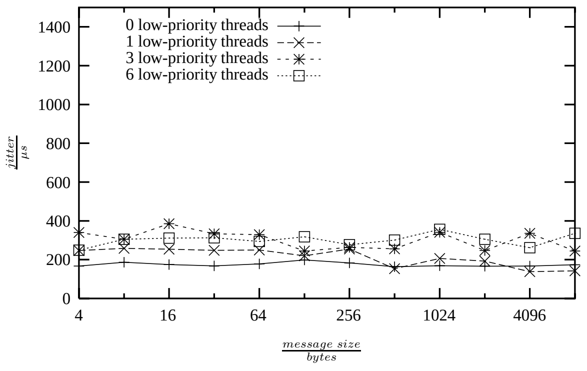

Figure 6. Jitter of a high-priority thread in the priority banded solution using Fast Ethernet

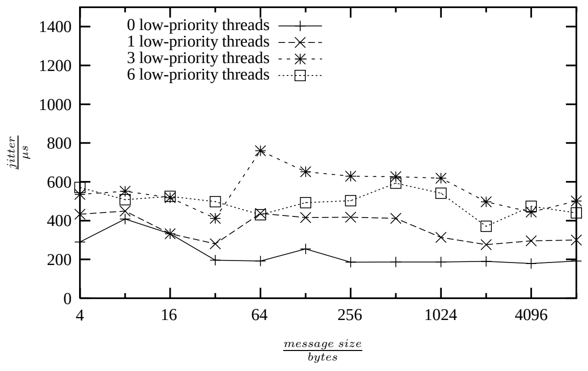

Figure 8. Jitter of high-priority thread in the normal solution using SCI

firms the assumption that a priority inversion occurs during the invocations. This problem is solved with a priority banded connection. Therefore an individual connection exists in this solution for each high-priority request. Figure 6 illustrates the results of the priority banded solution using the 100 Mbps Ethernet device and shows clearly that the jitter of this solution is definitely smaller. Additionally the average latency (see figure 7) of the priority banded solution is nearly equal to the average latency of the normal solution.

The results are nearly the same if the benchmark uses a SCI network. Figure 8 shows the results of the normal connection using the SCI network. In comparison to the solution using the Ethernet device (see figure 5), the jitter appears smaller. However, in real-time systems mainly the priority banded connection is used. If the priority banded connection uses the SCI interconnect, the results (see figure 9) are similar to the results of the priority banded connection using the Ethernet network (see figure 6). Figure 10 shows that for large messages the average latency of SCI-based solutions is clearly smaller than the average latency of the Ethernet-based solutions (see figure 7). But for small messages the average latency of SCI-based solutions is not smaller because the costs for signalizing incomming messages via SCI interrupts are nearly the same as in an Ethernet network. However in a SCI-based environment the average latency and the jitter are very small. This property is ideal for real-time systems.

5.2. Performance Evaluation with High Network Load

The benchmark in this section is based on the benchmark from section 5.1 and examines the performance of *ROFES* under high network load. On the client side, a single high-priority thread and six low-priority threads run concurrently. The high-priority and the low-priority threads invoke several requests with variable argument sizes. On the server side, a servant is created and configured to process the client requests at the same priority at which they were originally invoked the operation on the client side. This means that the

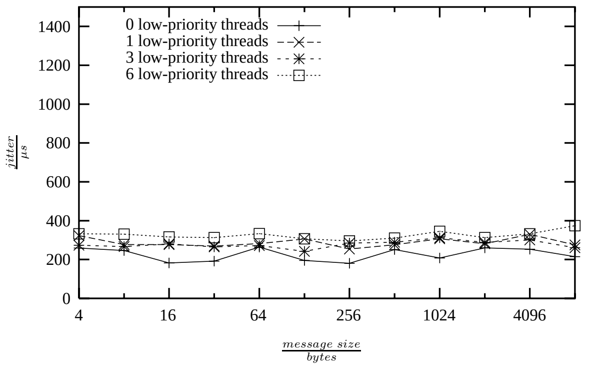

Figure 9. Jitter of high-priority thread in the priority banded solution using SCI

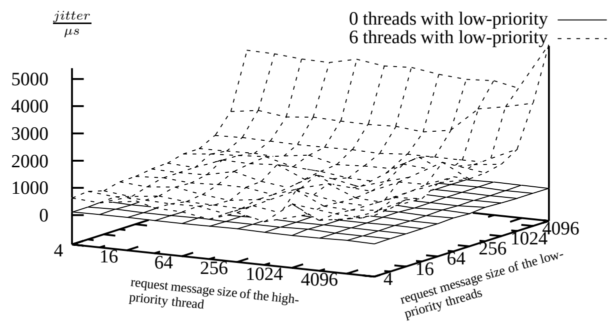

Figure 11. Jitter of a high-priority thread in the normal solution using Fast Ethernet

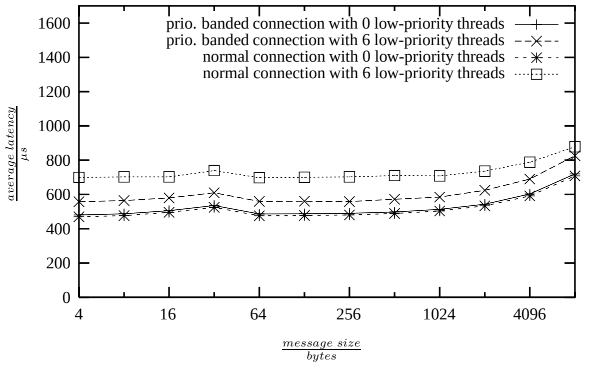

Figure 10. Average Latency of the high-priority thread using SCI

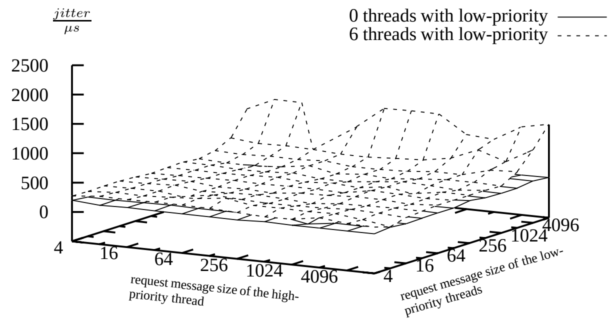

Figure 12. Jitter of a high-priority thread in the priority banded solution using Fast Ethernet

benchmark uses the client propagated priority model.

Figure 11 shows the results of the normal solution using the 100 Mbps Ethernet device. The delay jitter grows if the requests' message sizes of the low-priority threads become larger. Figure 12 shows that this problem is not completely solved by the priority banded connection. This acknowledges the assumptions described in section 3.2 that the probability of a priority inversion is very high. The results of the normal solution using the SCI interconnection (see figure 13) is clearly better than the results in the normal solution using Fast Ethernet (see figure 11). But figure 13 shows clearly that the jitter becomes worse by using several threads because in this solution the probability of a priority inversion is very high. Figure 14 proves that this problem does not occur when using the priority banded solution with the SCI interconnect. Hence the SCI interconnect is more suitable than Fast Ethernet for soft real-time systems.

6. Conclusions and Future Work

Real-Time CORBA helps to improve the flexibility, extensibility, maintainability and reuseability of distributed real-time applications. This article shows that *ROFES* possesses very good real-time characteristics. Additionally this article shows that SCI is suitable for soft real-time distributed applications.

In order to develop hard distributed real-time applications, ROFES needs a hard real-time network. For this reason ROFES will support other network types besides Ethernet and SCI. For instance *ROFES* will support *CAN* and *ATM* in the next few months. Subsequently, we will compare *SCI* and *CAN*, in order to determine whether *SCI* possesses as good real-time characteristics as *CAN*.

Real-Time Linux is a very popular real-time extension to *Linux*. Therfore in the near future *ROFES* will support *Real-*

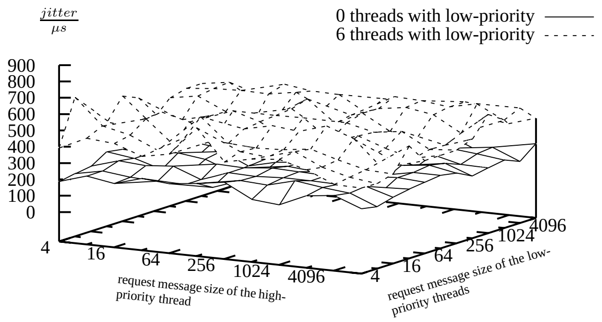

Figure 13. Jitter of high-priority thread normal solution using SCI

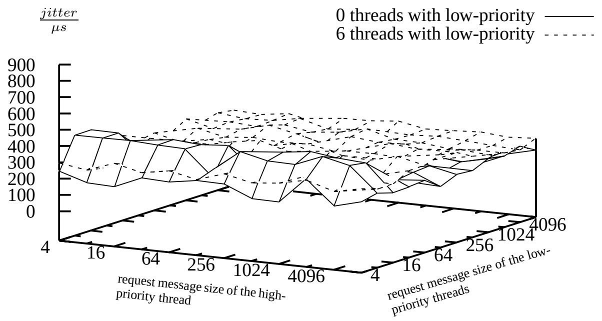

Figure 14. Jitter of high-priority thread in the priority banded solution using SCI

Time Linux. We are developing a time-triggered real-time protocol for Ethernet under *Real-Time Linux* that *ROFES* will support. We expect a smaller jitter by using this new protocol.

References

[1] K. Arvind, K. Ramamrithm, and J. A. Stankovic. A local area network architecture for communication in distributed real-time systems. Technical Report UM-CS-1991-004, Dept. of Computer Science, Univ. of Massachusetts at Amherts, 1991.

[2] M. Barabarnov. A linux-based real-time operating systems. Master's thesis, New Mexico Institution of Mining and Technology, 1997.

[3] Dolphin Interconnect Solutions. *PCI-SCI Cluster Adapter Specification*, 1996. White Paper.

[4] Dolphin Interconnect Solutions. *PCI-SCI Adapter Programming Specification*, 1997. Confidential Draft.

[5] B. P. Douglass. *Doing Hard Time – Developing Real-Time Systems with UML, Objects, Frameworks and Patterns*. Addison-Wesley, 1999.

[6] T. H. Harrison, D. L. Levine, and D. C. Schmidt. The design and performance of a real-time corba event service. In *Proceedings of OOPSLA '97*. ACM, 1997.

[7] M. Henning and S. Vinoski. *Advanced CORBA Programming with C++*. Addison-Wesley, 1999.

[8] IEEE. *ANSI/IEEE Std. 1596-1992, Scalable Coherent Interface (SCI)*, 1992.

[9] IEEE. *P1596.6, SCI/RT – Scalable Coherent Interface for Real-Time Applications*, 1992.

[10] D. James and D. Gustavson. Draft proposal for real-time transaction on sci. Draft Proposal, 1995.

[11] H. Kopetz. *Real-Time Systems – Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, 1997.

[12] S. S. Mukherjee and M. D. Hill. Making network interface less peripheral. *IEEE Computer*, 3(10):70–76, 1998.

[13] OMG Technical Document formal/98-07-01. *The Common Object Request Broker – Architecture and Specification*, 2.2 edition, 1998.

[14] OMG Technical Document orbos/98-08-04. *minimumCORBA – Joint Submission*, 1998.

[15] OMG Technical Document orbos/98-10-05. *Realtime CORBA – Joint Submission*, 1998.

[16] K. Ramamritham, C. Shen, G. Gonzalez, S. Sen, and S. Shirgurkar. Using windows nt for real-time applications: Experimental observartions and recommendation. In *Proceedings of the Fourth IEEE Real-Time Technology and Applications*, Denver, 1998.

[17] M. Sarwar, A. George, and D. Collins. Simulative reliability analysis of sci ring-based topologies. In *Proc. of IEEE Conference on Local Computer Networks (LCN)*, Tampa, 2000.

[18] D. C. Schmidt and F. Kuhns. An overview of the real-time corba specification. *IEEE Computer*, 33(6):56–63, 2000.

[19] J. Siegel. *CORBA 3 – Fundamentals and Programming, Second Edition*. OMG Press, 2000.

[20] T. Timmerman. Windows nt as real-time os. *Real-Time Magazine, 2Q97 Issue*, 1997.