Real-Time Systems, 5, 1–35 (1997)
© 1997 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

A Network Time Interface M-Module for Distributing GPS-Time over LANs*

ULRICH SCHMID, JOHANN KLASEK, THOMAS MANDL s@auto.tuwien.ac.at
Technische Universität Wien, Dept. of Automation, Treitlstraße 1, A-1040 Vienna, Austria.

HERBERT NACHTNEBEL, GERHARD CADEK, NIKOLAUS KERÖ keroe@iaee.tuwien.ac.at
Technische Universität Wien, Dept. of General Electrical Engineering and Electronics, Gußhausstraße 25–29, A-1040 Vienna, Austria.

Received November 25, 1995; Revised March 30, 1996

Editor: Wolfgang A. Halang

Abstract. This paper provides a comprehensive overview of our *Network Time Interface* (NTI) M-Module, which facilitates high-accuracy time distribution in LAN-based distributed real-time systems. Built around our custom UTCSU VLSI chip, it hosts all the hardware support required for interval-based external clock synchronization: A high-resolution state- and rate-adjustable clock, local accuracy intervals, interfaces to GPS receivers, and various timestamping features. Maximum network controller and CPU independence ensures that the available NTI prototype can be employed in virtually any COTS-based system with MA-Module interface. Our experimental evaluation shows that time distribution with μs-accuracy is possible even in Ethernet-based system architectures, provided that the available configuration parameters are suitably chosen to cope with the various hidden sources of timing uncertainty.

Keywords: Interval-based external clock synchronization, GPS time distribution, fault-tolerant distributed real-time systems, COTS, M-Modules, Ethernet, experimental evaluation.

1. Introduction

Designing distributed fault-tolerant real-time applications is considerably simplified when synchronized clocks are available. Temporally ordered events are in fact beneficial for a wide variety of tasks, ranging from correlating sensor data gathered at different nodes up to fully-fledged distributed algorithms, see (Liskov, 1993) for some examples. Providing mutually synchronized ("precise") local clocks is known as the *internal clock synchronization* problem, and numerous solutions have been worked out —at least in scientific research— under the term *fault-tolerant clock synchronization*, see (Ramanathan *et al.*, 1990*b*), (Simons *et al.*, 1990) for an overview and (Yang and Marsland, 1993) for a bibliography.

If synchronized clocks must also maintain a well-defined relation ("accuracy") to some external time standard like *Universal Time Coordinated* (UTC), then the

* The SynUTC-project (*http://www.auto.tuwien.ac.at/Projects/SynUTC/*) received funding from the Austrian Science Foundation (FWF) grant P10244-ÖMA, the OeNB "Jubiläumsfonds-Projekt" 6454, the BMfWV research contract Zl.601.577/2-IV/B/9/96, and the Austrian START programme Y41-MAT. The present work was also supported by the Austrian *Gesellschaft für Mikroelektronik* (GMe).

fault-tolerant external clock synchronization problem needs to be addressed. Appropriate solutions are particularly important for large-scale and wide-area distributed systems, since accuracy also secures precision among clusters that do not participate in a common internal synchronization algorithm. External synchronization did not receive much attention until recently, when highly accurate and cheap receivers for the *Global Positioning System* (GPS) became widespread; a quite comprehensive collection of related research can be found in (Schmid, 1997*c*).

The synchronization tightness achieved by any clock synchronization scheme depends primarily upon the underlying time distribution technique, i.e., the method for disseminating a node's local time to the other nodes in the system. The key parameter here is the uncertainty (= variability) ε of the end-to-end transmission delay. For typical LANs, ε lies in the ms-range, which makes it impossible to use data packets for disseminating time with high accuracy. Additional techniques are required here, which, however, should be compatible with existing network controller technology to be applicable in practice.

One of the achievements of our research project SynUTC (Schmid, 1994) is a suitable add-on hardware (Horauer *et al.*, 1998) that facilitates time distribution with μs-range accuracy in LAN-based distributed systems. Basically, it consists of hardware support for exact timestamping of clock synchronization data packets and a sophisticated local clock device. A prototype implementation of this *Network Time Interface* (NTI) is available (Mandl *et al.*, 1999), which can be used in conjunction with virtually any COTS CPU/network controller equipped with an MA-Module mezzanine interface. It was evaluated experimentally in an Ethernet-coupled distributed system made up of several VMEbus-CPUs running ISI's pSOS^{+m} multiprocessor real-time kernel. The appropriate evaluation results were first presented at the IFAC WRTP'99 (Schmid and Nachtnebel, 1999).

This paper unifies and extends our earlier work on the NTI, thereby providing a comprehensive overview of its architecture and features. It is organized as follows:

- Section 2 is devoted to the general system architecture and the resulting timestamping requirements.
- Section 3 contains the NTI's architecture, including a survey of the UTCSU VLSI chip in Subsection 3.1 and the hardware/software interface in Subsection 3.2. Subsection 3.3 shows how we incorporated the NTI into the pSOS^{+m} real-time kernel and reports on our experiences with COTS integration.
- Section 4 is devoted to the experimental evaluation of the NTI. Starting with an overview of the evaluation system's hard- and software architecture in Subsection 4.1, a reasonably complete discussion of our measurement results and findings is provided in Subsections 4.2 and 4.3.
- Section 5 links our experimental evaluation with the theoretical/algorithmic framework for interval-based clock synchronization. In Subsection 5.1, we show how our measurement results plug into a very simple time distribution scheme; Subsection 5.2 outlines the principles of more advanced algorithms.
- Section 6 relates our NTI to existing work on hardware-assisted clock synchronization.

Some conclusions in Section 7 eventually complete the paper.

2. System Architecture and Timestamping Principles

The SynUTC project aims at fault-tolerant external clock synchronization with μs-range precision/accuracy in LAN-based distributed systems. Figure 1 shows the basic architecture of a two-node system. Note carefully that there is no additional interconnection of the nodes apart from the packet-oriented data network.

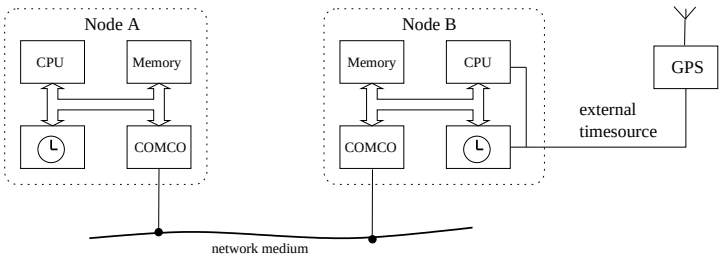


Figure 1. Basic architecture of a two-node distributed system for high-accuracy time distribution

Accordingly, each node must be equipped with

- a general purpose *CPU*, which can be the node's central processor or, preferably, a dedicated microprocessor or microcontroller that executes the software-part of the clock synchronization algorithm,
- a *communications coprocessor* (COMCO), which provides access to the network by reading/writing data packets from/to memory, independently of CPU operation,
- a *hardware clock*, namely, our UTCSU-ASIC, see Subsection 3.1.

For external synchronization purposes, at least one node must also be equipped with an external time source like a GPS satellite receiver, see e.g. (Dana, 1997) for an introduction.

To facilitate high-accuracy time distribution, this basic architecture has to be extended by a suitable mechanism for exact timestamping of *clock synchronization packets* (CSPs) at sending and receiving side. After all, the work of (Lundelius-Welch and Lynch, 1984) revealed that even n ideal clocks cannot be synchronized with a worst case precision less than $\varepsilon\,(1-1/n)$ in presence of a *transmission delay uncertainty* ε, which is defined as the variability of the difference between the real times of CSP timestamping at the peer nodes. Unfortunately, there are several steps involved in packet transmission/reception that could contribute to ε, cf. (Kopetz and Ochsenreiter, 1987):

1. Sender-CPU assembles the CSP
2. Sender-CPU signals sender-COMCO to take over for transmission
3. Sender-COMCO tries to acquire the network medium

4. Sender-COMCO reads CSP data from memory and pushes the resulting bit stream onto the medium
5. Receiver-COMCO pulls the bit stream from the medium and writes CSP data into memory
6. Receiver-COMCO notifies receiver-CPU of packet reception via interrupt
7. Receiver-CPU processes CSP

Purely software-based clock synchronization performs CSP timestamping upon transmission resp. reception in step 1 resp. 7, which means that ε incorporates the medium access uncertainty ($3 \rightarrow 4$), any variable network delay ($4 \rightarrow 5$), and the reception interrupt latency ($6 \rightarrow 7$). The first one can be quite large for any network utilizing a shared medium, and the last one is seriously impaired by code segments with interrupts disabled. Fortunately, in a LAN-based setting, we can safely neglect contributions from $4 \rightarrow 5$, since there are no intermediate gateway nodes and hence no (load- and hop-dependent) queueing delays. Therefore, the resulting transmission delay uncertainty ε emerges primarily from $1 \rightarrow 4$ resp. $5 \rightarrow 7$ at the sending resp. receiving node itself. Note that this statement remains true in a multi-hop network of the WAN-of-LAN type, if all intermediate nodes are also equipped with the timestamping mechanism described below, i.e., our NTI.

In an effort to reduce ε, clock synchronization hardware should thence be placed as close as possible to the network facilities. Ideally, a CSP should be timestamped at the sender resp. receiver exactly when, say, its first byte is pushed on resp. pulled from the medium. However, this needs support from the interior of the COMCO, which is only provided by a few research prototypes like (Horauer and Loy, 1998) or (Kopetz *et al.*, 1995). This explicit support of high-accuracy clock synchronization leads to an almost neglibile ε, cf. Section 6, which should decrease even further when transmission speeds increase. Unfortunately, if high-accuracy clock synchronization is to be built atop of commercially available network controller technology, a less tight method of coupling has to be considered.

Our NTI utilizes a refinement of the widely applicable DMA-based coupling method proposed in (Kopetz and Ochsenreiter, 1987) for this purpose. The key idea is to insert a timestamp on-the-fly into the memory holding a CSP in a way that minimizes the transmission delay uncertainty. More specifically, a modified address decoding logic for the memory is used, which

1. generates trigger signals that sample a transmit resp. receive timestamp into dedicated UTCSU-registers when a certain byte within the transmit resp. receive buffer for a CSP is read resp. written,
2. transparently maps the sampled transmit timestamp into some portion of the transmit buffer.

Note that this special functionality is only present when a transmit/receive buffer is accessed by the COMCO, whereas CPU-read/writes act as plain memory accesses.

Refering to the data transmission and reception sequence introduced above, CSP timestamping is thus moved to step 4 and 5, respectively. Hence, the only activities

that still contribute to ε are the time between fetching a byte from the transmit buffer and trying to deposit it in the receive buffer, and the bus arbitration necessary for COMCO memory writes upon CSP reception.

To further illustrate the resulting process of CSP timestamping, we briefly explain one possible scenario depicted in Figure 2; alternative ones may be found in (Schossmaier and Schmid, 1995).

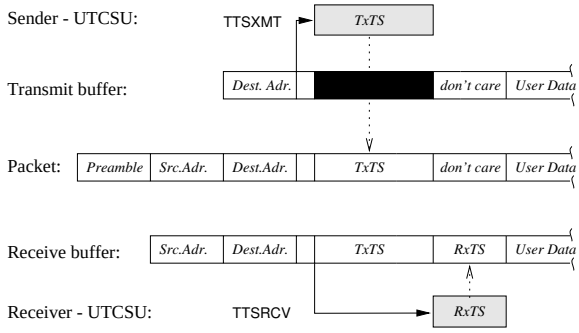


Figure 2. Hardware-supported packet timestamping at sending and receiving node

Whenever the COMCO fetches data from the transmit buffer holding the CSP for transmission, it has to read across the particular address that causes the decoding logic to generate the trigger signal `TTSXMT`. Upon occurrence of this signal, the UTCSU puts the *transmit timestamp* (TxTS) into a dedicated sample register, which is transparently mapped into a certain succeeding portion of the transmit buffer and hence automatically inserted into the outgoing packet. Note that the trigger address and the mapping address may be different. By the same token, when the COMCO at the receiving side writes a certain portion of the receive buffer, the trigger signal `TTSRCV` is generated by the decoding logic, which causes the UTCSU to sample the *receive timestamp* (RxTS) into a dedicated register. Subsequently, the timestamp can be saved in an unused portion of the receive buffer upon reception notification or by a similar transparent mapping technique.

The proposed approach works for any COMCO that can access CSP data in external memory. Suitable chip-sets are available for several networks, ranging from fieldbusses like Profibus over Ethernet up to ATM networks. Two limiting factors must be considered, however:

(1) COMCOs with large on-chip storage (FIFOs) in the transmission path cannot be used. As exemplified in Subsection 4.2, if an entire packet could await (re-)transmission in the on-chip FIFO, ε can become as large as the maximum packet transmission time.

(2) Determining ε for a particular COMCO is usually impossible without experimental evaluation. As reported in Subsection 4.2, theoretical "data sheet"

knowledge might prove insufficient to fully assess the hidden architectural intricacies of a COMCO and to properly select suitable values for the NTI's transmit/receive timestamp trigger positions.

3. NTI Hardware and Software Architecture

Our *Network Time Interface* (NTI) provides all the hardware support required for high-accuracy clock synchronization according to the previous exposition. Aiming at COTS-based system architectures and maximum CPU and COMCO independence, we decided to use a suitable mezzanine bus interface to add the required functionality to existing CPU boards. Guided by issues like simplicity, robustness, size, etc., the NTI prototype was eventually designed as an MA-Module. Figure 3 shows the final result of our development efforts, which are documented in a number of papers and technical reports (Schossmaier and Schmid, 1995), (Horauer *et al.*, 1996), (Horauer *et al.*, 1998), (Nachtnebel *et al.*, 1998), and (Mandl *et al.*, 1999).

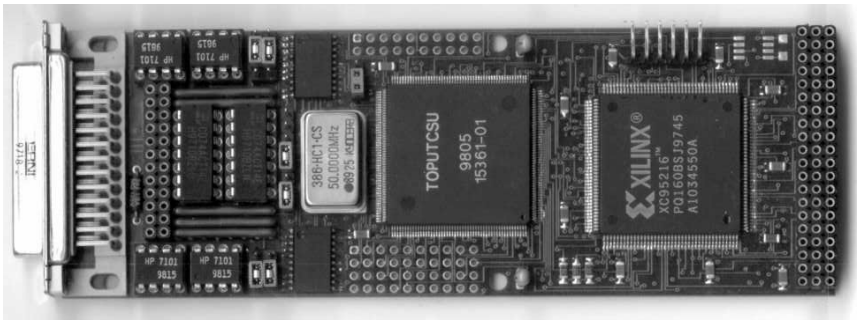


Figure 3. Snapshot of the NTI MA-Module

M-Modules (MUMM, 1996) are an open, simple and robust mezzanine bus interface primarily designed for VME carrier boards, which are commonly used in Europe. *MA-Modules* are enhanced M-Modules, providing a 32-bit data bus instead of the 16-bit one of the original M-Modules. The address space consists of 256 bytes *I/O-space* accessible via the standard M-Module interface, and up to 16 MB of *memory-space* addressed by multiplexing the MA-Module data bus. The asynchronous bus interface requires the module to generate only an acknowledgement signal for terminating a bus cycle, thus minimizing the on-board control logic. Further signals in the M-Module interface comprise a single vectorized interrupt line and two additional DMA control lines. The unit construction design of the 146 x 53 mm MA-Modules provides a peripheral 25-pin D-sub *front-panel connector* (FPC), a 24-pin peripheral *carrier plug connector* (CPC) and a 60-pin *MA-Interface Connector* plugging into the carrier board, and finally a 2 x 10-pin *intermodule port connector* (IPC).

Figure 4 shows the major components on-board the NTI, which can be accessed from any COTS CPU/COMCO with MA-interface via ordinary memory and memory-mapped registers.

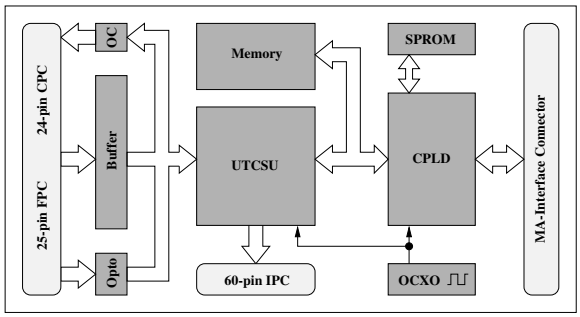


Figure 4. Block diagram of the NTI MA-Module

Accordingly, the NTI hosts the following major components:

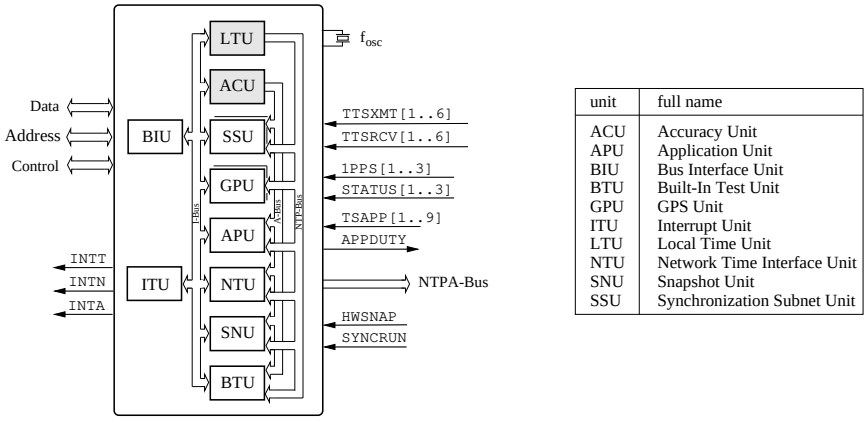
- The *UTCSU-ASIC* surveyed in Subsection 3.1 below contains most of the dedicated hardware support for clock synchronization. It is clocked by an on-board temperature-compensated (TCXO) or ovenized (OCXO) quartz oscillator; alternatively, an external frequency source like the 10 MHz output of a high-end GPS receiver can be used.
- The *memory* serves as control and data interface between the CPU and the COMCO, providing the special functionality for COMCO accesses outlined in Section 2. It consists of up to four 64k x 16-bit SRAM chips and supports byte, word, and longword read/write accesses. The memory map of the current version of the NTI —designed for Intel's i82596 CA Ethernet coprocessor— can be found in Subsection 3.2.
- Any decoding and glue logic of the NTI is assembled in a single, in-circuit programmable *complex programmable logic device* (CPLD) designed using VHDL (Nachtnebel *et al.*, 1998). It adapts the UTCSU and the memory to the MA-interface, provides the timestamp triggering and mapping functionality, forwards interrupt requests from the UTCSU to the carrier-board, generates the acknowledgement signal terminating a bus cycle, etc.

All application-related I/O-pins of the UTCSU as well as all interfaces to GPS receivers are routed to the M-Module's 25-pin D-sub FPC. In addition, all receive and transmit timestamp signals are made available to the carrier board via the 24-pin CPC. Finally, an extended 60-pin IPC exports the UTCSU's internal time information ("NTPA-bus") for future expansion modules, and facilitates the connection of modularized GPS receivers. Of course, high-speed opto-couplers or buffers are provided for all inputs to ensure a decoupled and reliable interface.

3.1. UTCSU-ASIC

In this subsection, we provide a succinct overview of the wealth of functionality of our custom *Universal Time Coordinated Synchronization Unit* (UTCSU), which contains most of the clock synchronization hardware support provided by the NTI. Considerable effort has been spent on its development, which is documented in a number of reports and papers (Schossmaier and Schmid, 1995), (Schossmaier and Loy, 1996), (Loy, 1996), and (Schossmaier *et al.*, 1997).

The UTCSU-ASIC has been manufactured using a 0.7 μm digital CMOS technology (Atmel-ES2). The design complexity is about 80,000 gates, which resulted in a die size of 100 mm^2 that was packed into a high-density 208-pin PQFP case. Due to its flexible bus interface, featuring dynamic bus sizing and little/big endian byte ordering, it can be used in conjunction with virtually any 8-, 16- and 32-bit CPU. Figure 5 gives an overview of the major functional blocks inside the UTCSU.



unit	full name
ACU	Accuracy Unit
APU	Application Unit
BIU	Bus Interface Unit
BTU	Built-In Test Unit
GPU	GPS Unit
ITU	Interrupt Unit
LTU	Local Time Unit
NTU	Network Time Interface Unit
SNU	Snapshot Unit
SSU	Synchronization Subnet Unit

Figure 5. Major building blocks and signals of the UTCSU-ASIC

The centrepiece of the UTCSU is the *Local Time Unit* (LTU), primarily hosting a local clock that maintains a fixed point representation of the current time with a 32-bit integer part and a 24-bit fractional part, i.e., a 56-bit NTP-time (Mills, 1991). Clock time can be read atomically as a 32-bit *timestamp* with a resolution of $2^{-24} \approx$ 60 ns that wraps around every 256 s, and a 32-bit *macrostamp* containing the remaining 24 most-significant bits along with an 8-bit checksum protecting the entire time information.

The local clock of the UTCSU can be paced with any oscillator frequency $f_{osc} \in 1 \ldots 25$ MHz, is fine-grained rate adjustable in steps of about 10 ns/s, and supports adjustment via continuous amortization as well as optional leap second corrections in hardware. Those outstanding[1] features are primarily a consequence of our novel *adder-based clock* (ABC) design, which uses a large (91-bit) high-speed adder instead of a simple counter for summing up the elapsed time between consecutive

oscillator ticks. Of course, a proper augend value (in multiples of 2^{-51} s ≈ 0.44 fs) must be loaded to achieve the desired rate of progress of the ABC.

To support interval-based clock synchronization (see Section 5), our UTCSU contains two additional adder-based "clocks" in the *Accuracy Unit* (ACU) that are also driven by the oscillator frequency f_{osc}. They are responsible for holding and automatically deteriorating the 16-bit *accuracies* α^- and α^+ to account for the maximum oscillator drift. Both can be (re)initialized atomically in conjunction with the local clock in the LTU. In addition, some extra logic suppresses a wrap-around of α^- and α^+ and zero-masks potentially negative accuracies during continuous amortization.

A number of external events, supplied to the UTCSU via polarity programmable input lines, can be *time+accuracy-stamped* with local time and accuracy (i.e., α^- and α^+); the instantaneous values are atomically sampled into dedicated registers upon the appropriate input transition. Optionally, an interrupt can be triggered on such an event as well. Due to the asynchronous nature of these inputs, internal synchronizer stages are utilized that introduce a timing uncertainty of at most $1/f_{osc}$. Note that a one- resp. two-stage synchronizer is employed, depending on the status of the UTCSUs `RELIABLE`-pin; the recovery time for metastability phenomenons is therefore $1/f_{osc}$ resp. $2/f_{osc}$, resulting in a reasonably small probability of failure, see (Loy, 1996) for more information.

Three different functional blocks in the UTCSU utilize time+accuracy-stamping:

1. The `TTSXMT` and `TTSRCV` trigger signals generated by the decoding logic at CSP transmission/reception (as explained in Section 2) sample the current local time+accuracy into dedicated UTCSU-registers in the *Synchronization Subnet Unit* (SSU). Six independent SSUs are provided to support redundant communication architectures or gateway nodes.
2. Three independent *GPS Units* (GPUs) are provided for timestamping a *1 pulse per second* (`1PPS`) signal —indicating the exact beginning of a second— from up to three GPS receivers. Note that this simple interface, augmented by an optional `STATUS`-input, is sufficient for connecting GPS timing receivers, since the additional and less time critical information is usually provided via a serial interface and handled off-chip the UTCSU.
3. Nine independent application time+accuracy-stamping inputs are provided by the *Application Unit* (APU). Additional application-related features can be realized off-chip by tapping the 48-bit wide multiplexed *NTPA-Bus*, which exports the entire local time and accuracy information at full speed.

The above timestamping features are complemented by 48-bit *duty timers* accommodated in several functional blocks of the UTCSU. Duty timers are required for executing the protocol for CSP exchange governed by the clock synchronization algorithm, controlling continuous amortization, inserting/deleting leap seconds, and generating application-related events. Whenever an armed duty timer goes off due to the fact that local time reaches the programmed one, an interrupt is raised. Moreover, the APU's duty timer can be used to generate a pulse on the dedicated `APPDUTY`-output as well.

Obviously, there are many different interrupt sources on-chip the UTCSU, which can be controlled on an individual basis. They are statically mapped onto three dedicated UTCSU interrupt outputs `INTN` (network-related), `INTT` (timer-related) and `INTA` (application-related). Since M-Modules provide only a single interrupt line for signaling a vectorized interrupt, the NTI is responsible for further mapping `INTN`, `INTT`, and `INTA` onto that single interrupt line and generating the appropriate interrupt vector.

Last but not least, the UTCSU is equipped with features for test and debugging purposes, provided by the *Built-In Test Unit* (BTU) and *Snapshot Unit* (SNU). They include calculation of checksums, blocksums and signatures for local time, snapshots of certain registers to facilitate experimental evaluation of precision and accuracy, and a synchronous (re)start feature. Those provisions are particularly useful for fault-tolerant applications based on self-checking and/or redundant units.

3.2. NTI Hardware/Software Interface

In this subsection, we briefly survey how the NTI can be accessed by a node's CPU and COMCO and, hence, by the clock synchronization algorithm and the network device driver. A comprehensive description of all those low-level programming issues can be found in (Mandl *et al.*, 1999).

Any access to UTCSU-registers and NTI memory is performed by addressing the M-Modules memory-space shown in Figure 6.

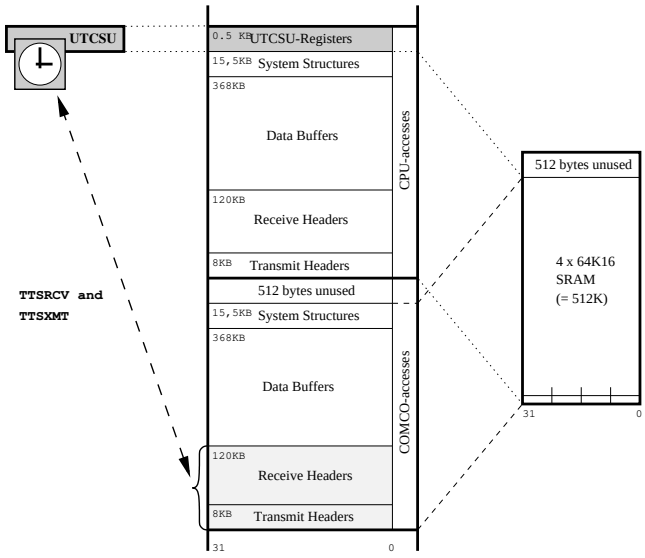


Figure 6. Layout of the memory-space of the NTI MA-Module

Since only COMCO reads/writes trigger timestamping functionalities, the NTI must be able to distinguish CPU and COMCO accesses. This is accomplished by mapping two 512 KB address regions onto the same 512 KB of physical NTI memory. On top of the memory map is the 512 KB memory address region for CPU-accesses, which includes a 512-byte segment containing the UTCSU registers. It is decoded without special functionality. The bottom 512 KB region is dedicated to COMCO-accesses and involves timestamp triggering and mapping functionality.

Both regions are divided into (the same) four sections: The *System Structures* section holds the command interface and system data structures required by the COMCO, the *Data Buffers* are available for packet data. Timestamp triggering and mapping functionality applies only to certain addresses in the *Receive Headers* and *Transmit Headers* sections, which hold packet-specific control and routing information (like source & destination address and type field) for received and transmitted CSPs, respectively.

The CPLD is currently programmed to support Intel's 82596CA Ethernet coprocessor with 64-byte receive and 128-byte transmit headers. Figure 7 outlines the (software-programmable) offsets within each of the headers that have associated special functionality.

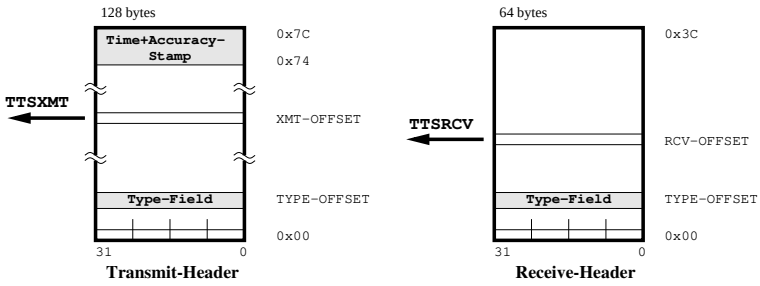


Figure 7. Layout of receive and transmit headers for COMCO accesses

As soon as the COMCO reads offset `XMT-OFFSET` within a transmit header upon transmission of a CSP, the timestamp trigger signal `TTSXMT` is issued to the UTCSU. Since the UTCSU-registers holding the sampled time+accuracystamp are transparently mapped into the (fixed) offsets 0x74–0x7F in the transmit header, it is automatically inserted into the outgoing data packet.

Similarly, as soon as the COMCO writes offset `RCV-OFFSET` within a receive header upon reception of a CSP, the timestamp trigger signal `TTSRCV` is generated. Moreover, the base address of the accessed receive header is stored in a dedicated NTI-register to facilitate saving the receive timestamp in an *interrupt service routine* (ISR), as explained below. To reduce the resulting interrupt load, however, any special receive processing functionality is only active when a CSP drops in. The data packet's type field, written to the receive header at offset `TYPE-OFFSET`, is used to distinguish CSPs from ordinary data packets.

In addition to the UTCSU-registers and the shared memory, there are also a few dedicated NTI-registers with 16-bit each. They are accessible in the M-Module I/O-space according to the memory map in Figure 8.

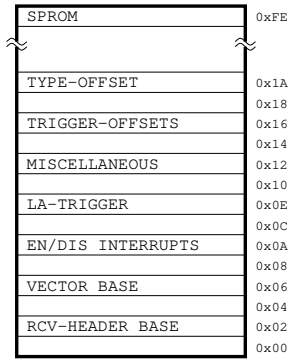


Figure 8. Layout of the I/O-space of the NTI MA-Module

The `RCV-HEADER BASE` register is required for correctly assigning receive timestamps to data packets: After the UTCSU has sampled a receive time+accuracy-stamp, it must be moved to an unused portion of the appropriate CSP before the next one drops in. This can be done in an ISR activated by a `TTSRCV` interrupt, which, however, cannot reliably[2] compute the address of the receive header associated with the sampled timestamp without hardware assistance. Therefore, the NTI latches this address into `RCV-HEADER BASE` upon the occurrence of the `TTSRCV`-signal.

Two NTI registers control interrupt generation: The `VECTOR BASE` register can be used to setup the interrupt vector generated upon an UTCSU interrupt. Note that the final vector also includes the state of the three UTCSU interrupt pins `INTT`, `INTN`, and `INTA`. Accesses to Register `EN/DIS INTERRUPTS` (re-)enable/disable the interrupt logic of the NTI.

Reading or writing register `LA-TRIGGER` generates a pulse on a dedicated NTI output, which is useful for test and debugging purposes. The `MISCELLANEOUS` register allows to tie a few distinguished UTCSU pins to a specific level, thereby switching on/off additional synchronizer stages, for example.

The register `TRIGGER-OFFSETS` allows to setup the `XMT-OFFSET` and `RCV-OFFSET`, which determine where timestamp triggering occurs in a transmit and receive header, respectively. Moreover, the `TYPE-OFFSET` register should be preloaded with the offset of the data packet type field in the receive header, as explained earlier.

Finally, the `SPROM` register on top of the I/O-space provides access to a serial PROM, which contains the M-Module's identification data according to (MUMM, 1996).

3.3. NTI Device-Driver for pSOS^{+m}

In this subsection, we sketch how the NTI was eventually incorporated into the state-of-the-art[3] industrial multiprocessing/multitasking real-time kernel pSOS^{+m} (Integrated Systems, Inc.).

The features of the NTI are made available to the clock synchronization algorithm and the application tasks by means of two layers of driver software written in C. The lower-level one is the *NTI-Handler* (Schmid and Mandl, 1999), which is responsible for initialization and configuration of the NTI, the M-Module carrier-board and the low-level interrupt handling. The current version supports both AcQ i6360 and MEN A203 passive VMEbus carrier-boards in conjunction with Motorola's MVME-162 CPU (M68040 CPU + Intel i82596CA Ethernet coprocessor). The upper-level software layer is provided by the *i82596 NTI-Driver* (Richter *et al.*, 1999), which actually integrates the NTI and the i82596 COMCO into pSOS^{+m}. Note that a transition to a different target system hardware only requires re-development of (parts of) the NTI-Driver + NTI-Handler and perhaps some modification of the NTI-CPLD.

Figure 9 outlines the complete software structure of a node, including the underlying hardware.

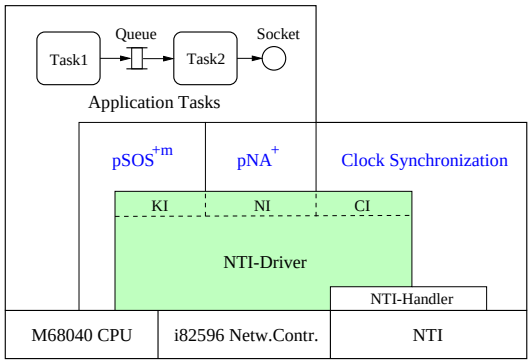


Figure 9. Architecture of a pSOS^{+m} node using the NTI-Driver

It is apparent that the NTI-Driver actually multiplexes three different interfaces to the COMCO and the NTI:

1. *Kernel Interface* (KI): pSOS^{+m} supports multiprocessing by means of remote objects (tasks, queues, semaphores, etc.), which are implemented atop of RPCs. To keep the kernel reasonably independent of the underlying network, a user-supplied KI is required that maps a simple message-passing interface to the particular COMCO.

2. *Network Interface* (NI): In addition to kernel services, application tasks can use TCP/IP sockets for communication with remote sites if the additional software

component pNA$^+$ is present. Like the pSOS^{+m} kernel, pNA$^+$ is kept hardware-independent by means of a user-supplied NI, which is similar to the KI but plugs into a different message-passing interface.

3. *Clock Interface* (CI): The third component that requires network services is the clock synchronization algorithm. Again, a simple message-passing interface CI is sufficient here. Note that it is the only one that relies upon the timestamping feature of the NTI.

Viewed at the level of application tasks, the NTI-Driver and the atop running clock synchronization algorithm simply add synchronized clocks to a standard pSOS^{+m}/pNA$^+$-environment. Apart from the created CPU and network load, the process of clock synchronization is in fact totally transparent to the application. Therefore, complex distributed timing problems can be solved easily by means of the various timestamping and duty timer functionalities of the UTCSU's APU, recall Subsection 3.1.

Of course, simplicity of usage does not say anything about the effort for developing the underlying system and, in particular, system integration. In fact, our experiences with integrating a custom piece of hard- and software, namely, the NTI and its driver software, into a COTS-based target system were definitely sobering. We can really only advice to carefully consider articles like *"COTS Integration: Plug and Pray?"* (Boehm and Abts, 1999) before deciding on how to do prototype development.

In our case, we decided to develop the NTI for a COTS-based system architecture not least because of the perspective of reduced design complexity and hence development time: Since both CPU and COMCO could be borrowed from COTS components, there was no need to accomodate them on-board the NTI. Those savings, however, finally proved quite expensive due to a number of unanticipated issues that almost caused the project to fail at all:

1. When encountering some misbehavior of the NTI prototype on a COTS M-Module carrier board, obviously the NTI was blamed for it. It took some time to recognize that the real cause might be the carrier board as well: Both the i6360 and the A203 had at least two serious bugs, ranging from ringing at critical signals over specification violations up to tricky signal race conditions.

2. Finding a bug on a COTS module is one thing; convincing the manufacturer of its existence and pressing him to fix it is a completely different matter. In fact, if custom hardware like the NTI is involved, suppliers usually divert the problem to these components. Hence, one is forced to develop a handy hardware + software system that clearly reproduces the bug, and ship it. Even if this can be managed, however, it depends upon the severity of the error —and the importance of the customer— whether the manufacturer is willing to provide a bug fix in due time.

3. Even the best documentation of a COTS product usually lacks an in-depth description of certain non-standard features. For example, when the CSP memory was moved from the on-board RAM of the MVME-162 to the NTI, the

NTI-Driver showed completely irregular behavior. Our first guess was that the i82596 on the MVME might be unable to access memory on the VMEbus, contradicting both hints in the user manual and pre-purchase statements of the supplier. Several weeks after having issued the problem to the customer support, this guess was confirmed by Motorola. Accidentally, however, a clue to an i82596 test that allowed to specify the RAM address was found in the MVME diagnostics utilities manual. Trying it out with the NTI-RAM, it worked fine! (The problem was eventually tracked down to the fact that i82596 locked access cycles had been enabled, which are not supported on the VMEbus.)

Therefore, at the end, we are no longer convinced that it was indeed a good idea to build an NTI without CPU and COMCO as the initial prototype. It may well be the case that, with respect to the overall development cost, improved control of the behavior of an NTI with on-board CPU and COMCO would have brought in the additional development time.

4. Evaluation

One of our major motivations for taking the trouble of developing a professional NTI prototype was the prospect of experimental evaluation. And indeed, backed up by our measurement results, we are now in the position to set realistic figures against the often questionable claims given in literature now and then, cf. Section 6.

The basic functionality of the evaluation system employed for our experiments is quite simple: It just measures and statistically analyzes the transmission delays arising in CSP exchanges between peer nodes. Figure 10 explains the quantities of interest in such a roundtrip between a *client node* p and *server node* $q \neq p$.

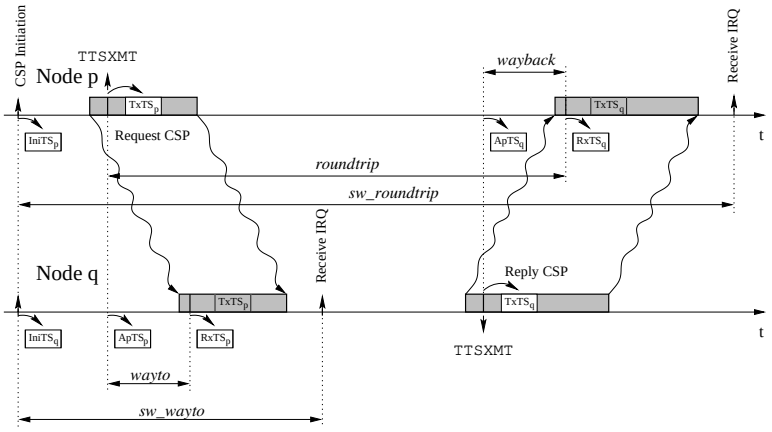


Figure 10. Transmission delays involved in a single CSP exchange

Triggered by a suitable CSP initiation event, any client node p broadcasts a "request" CSP to any server node q. Upon reception, node q immediately sends back a "reply" CSP to node p that eventually completes the roundtrip.

Both histograms and standard statistical parameters like averages, standard deviation and minima/maxima are computed for any of the primary[4] quantities of interest given in Table 1 and any pair of peer nodes. In addition, 99%- and 95%-minima/maxima, that is, bounds like $\min_{99}$ securing that at most 0.5% of the outcomes are less than $\min_{99}$, are provided for easy characterization of long tails of rare events.

Table 1. Quantities gathered by the evaluation system for hardware timestamping (HW) and pure software timestamping (SW)

Quantity	**Timestamping**	**Meaning**
wayto	HW	one-way transmission delay client to server (request CSP)
wayback	HW	one-way transmission delay server to client (reply CSP)
roundtrip	HW	roundtrip delay
sw_wayto	SW	one-way transmission delay client to server (request CSP)
sw_roundtrip	SW	roundtrip delay

Whereas the evaluation system's output is quite straightforward, this is definitely not true for its "input". In fact, the various system and design parameters that potentially affect transmission delays turned out to be a challenge to both devising the experiments and designing the evaluation system. Table 2 lists the key parameters explored more or less systematically during our experimental evaluation, along with a rule-of-thumb characterization of their actual effect on the hardware timestamping capabilities of the NTI. Note that one of the most crucial parameters, namely, NTI bus speed, was simply overlooked when planning our experiments and had to be included during evaluation, see Subsection 4.3.

Table 2. Key parameters potentially affecting transmission delays and their actual effect

Parameter	**Effect**
TxTS and RxTS timestamp trigger offsets	Subsection 4.2
NTI bus speed	Subsection 4.3
Interfering NTI-accesses	Subsection 4.3
Number n of nodes	irrelevant (for $n \geq 3$)
Network load L	irrelevant
Network segment lengths	irrelevant
Simultaneous vs. staggered CSP initiation	irrelevant
CSP broadcast vs. multiple unicasts	irrelevant
Size of request/reply CSP	irrelevant
CPU and interrupt load	irrelevant

4.1. Evaluation System Architecture

Aiming at the support of existing technology, it was only natural to use[5] COTS components for building up the evaluation system as well. In this subsection, we

briefly survey its hard- and software architecture, which is admittedly unlikely to be chosen in practice. It is nevertheless most appropriate for evaluation, however, since it constitutes something like a "worst case environment".

Our evaluation system consists of multiple nodes comprising a Motorola MVME-162 CPU and a AcQ i6360 or, alternatively, a MEN A203 VMEbus carrier-board hosting the NTI MA-Module, which are plugged into a dedicated A32/D32 VMEbus backplane. All nodes are interconnected via the CPU's 10 Mbit/s Ethernet port using thin-wire technology. Figure 11 outlines the basic hardware architecture.

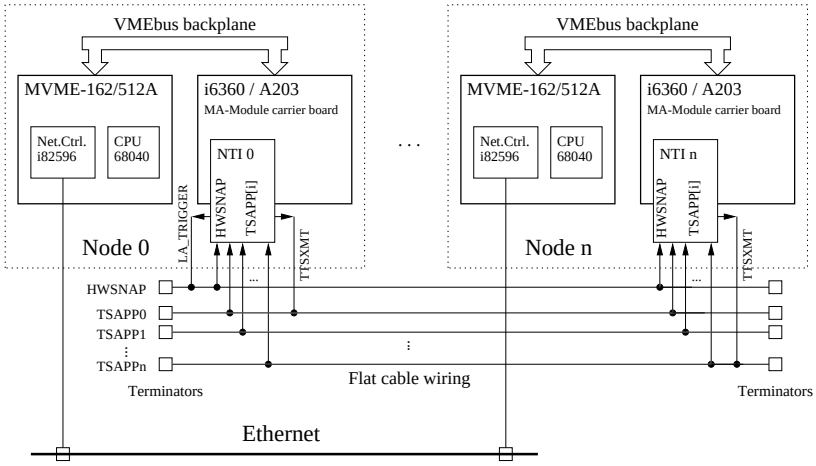


Figure 11. Hardware architecture of the evaluation system (two nodes shown only)

Dedicated flat cable wiring —the only add-on to Figure 1 required for experimental evaluation— is used for measuring the one-way transmission delay *wayto* and *wayback* between any two nodes in the system, cf. Figure 10. More specifically, the `TTSXMT`-signal triggering the CSP transmit timestamp on node i's NTI is fed to the application timestamp input `TSAPP`i on each NTI in the system. This way, any node that receives a CSP from node i just has to compute the —locally available— timestamp-difference $\mathrm{RxTS}_i - \mathrm{ApTS}_i$ to obtain the packet's transmission delay.

To facilitate coordinated activity of all client nodes, we also implemented a simultaneous interrupt that initiates CSP transmission. Optionally, a node-specific delay may be interposed to achieve deterministic or random staggering as well. For this purpose, the `HWSNAP`-timestamp inputs of all NTIs are tied together and driven by the `LA_TRIGGER`-output of NTI 0, which hence acts as a "master node".

The development of the evaluation system's software was considerably simplified by utilizing the NTI-Driver introduced in Subsection 3.3. In fact, thanks to the powerful pSOS^{+m}-features, the multitasking system shown in Figure 12 was developed without much difficulty.

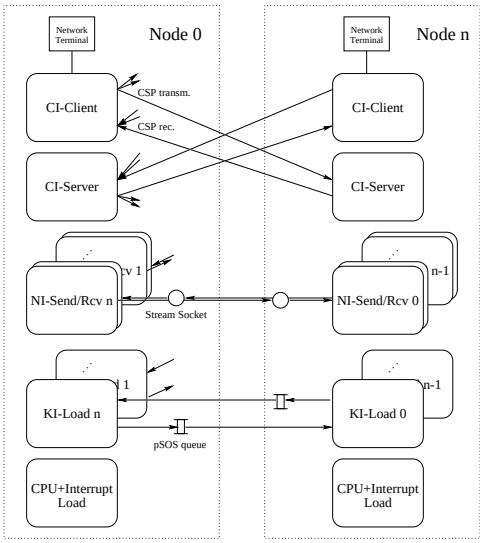


Figure 12. Software architecture of the evaluation system (two nodes shown only)

It consists of a number of concurrently executing tasks (the same at each node), which can be parameterized on-line by means of a custom configuration dialogue.

- The *CI-Client* is the major task of the evaluation system. It periodically broadcasts CSPs to the *CI-server* at any other node and collects the reply CSPs according to Figure 10. After a specific number of rounds has been reached, the CI-client computes the transmission delay statistics of Table 1 and prints them on the network terminal. Among the many configurable parameters are the number of rounds, broadcast period, deterministic/random staggering, CSP size, TxTS and RxTS trigger offsets, etc., cf. Table 2.
- Any pair of *NI-Send* and *NI-Receive* tasks at a remote node exchange data via a TCP/IP stream socket, thereby generating network traffic of configurable load and type.
- Similarly, any pair of *KI-Load* tasks at different nodes use two global pSOS^{+m} queues to exchange data over this IPC mechanism.
- Finally, there is a task responsible for generating a configurable CPU and/or interrupt load with or without accesses to the NTI.

4.2. Exploratory Measurements

In this and the following subsection, we provide a reasonably complete discussion of the conclusions drawn from the wealth of measurement data gathered by our evaluation system. Like our experiments, we also start our presentation with more or less "exploratory" measurements.

It had been clear right from the beginning that the COMCO's large FIFOs used for tolerating varying bus latencies when accessing memory would impair ε. The i82596 provides a 64-byte *Tx-FIFO* and a 128-byte *Rx-FIFO*, which are filled from/emptied to memory by two on-chip DMA channels. Elaborate prefetching policies ensure that the data consumed/produced by the serial side at the constant rate of 10 Mbit/s can be handled even when the bus access grant is delayed. Given the NTI's flexibility w.r.t. programming the TxTS and RxTS timestamp offsets, however, it was to be hoped that a gross impairment could be circumvented. Fortunately, this proved true at the end, although it turned out that "theoretical" knowledge from the —comprehensive and quite detailed— i82596 users manual was not sufficient to avoid certain tricky pitfalls revealed by experimental evaluation.

For example, the i82596 manual says that the serial side initiates transmission when Tx-DMA has loaded the first two lwords (=8 bytes, destination address and type field) from CSP data. Therefore, we concluded that transmission is in progress, i.e., not deferred due to carrier sense or collision resolution, when the succeeding lwords are eventually fetched. Since CSP data is preceeded by 4 additional lwords containing control information for the i82596, this led to the decision to trigger TxTS at offset 28 [bytes] in the CSP buffer. Experimental evaluation, however, revealed an unacceptable $\varepsilon \approx 1.2$ ms in this case, which is the duration of a maximally-sized Ethernet-packet. Figure 13 outlines the reason for this unpleasant effect.

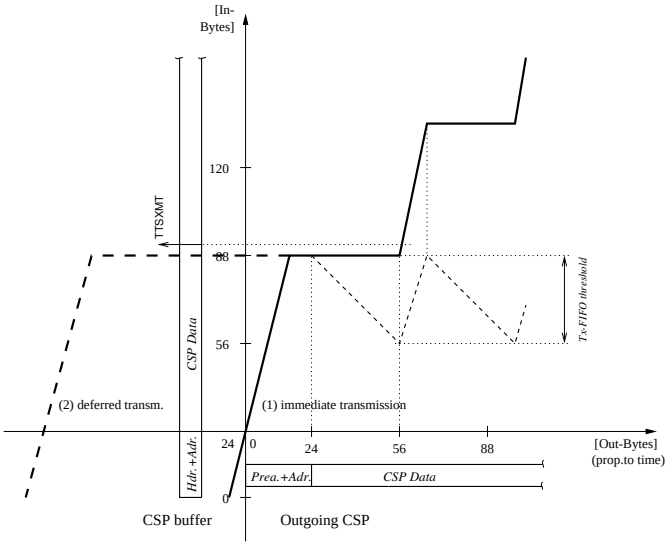


Figure 13. Timing relation of accesses to the memory holding a CSP vs. outgoing bytes on the network

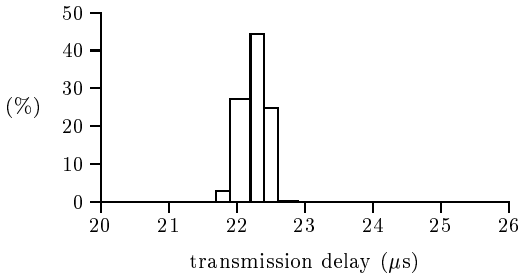
The shown curves relate the instant of accessing a certain offset in the CSP buffer with the outgoing transmission at the serial side. Since the latter is performed at

the constant rate of 10 Mbit/s, the x-axis can be interpreted as a time axis. The time origin is start of channel activity, which coincides with reading offset 24, as said before. The fat solid curve (1) represents the case of sensing an idle channel.

What was not anticipated correctly is the fact that the Tx-FIFO is also completely filled when transmission is deferred, as shown by the fat dashed curve (2). By choosing the TxTS trigger offset 28 as above, the packet could be delayed by the maximum channel access time *after its TxTS was drawn*! Hence, the TxTS trigger offset must be moved beyond offset 24+64=88, so that the 64 byte Tx-FIFO can be completely filled without triggering `TTSXMT`. TxTS is eventually drawn during the next Tx-FIFO fill, which takes place after 32 bytes have been read off the Tx-FIFO and sent to the channel, cf. the dashed line in the figure above.

Using the "search mode" of our evaluation system, which automatically scans all TxTS and RxTS trigger offsets within certain limits, we convinced ourselves that any TxTS offset > 92 indeed performs equally well. Note that the effect of varying the RxTS trigger offset is negligible, at least if it remains close to the beginning of the receive header.

Having settled the problem of proper TxTS selection, we started exploring the "best case" scenario. A two-node configuration was used for this purpose, where one node acted as client and the other as server only. Obviously, no NI- and KI-load was generated on the network and no additional CPU and interrupt load was put on the nodes itself. Figure 14 shows the appropriate transmission delays for the A203 carrier board. Although the resulting ε of about 1 μs is not representative for the general case discussed in Subsection 4.3, it nevertheless reveals how the NTI would perform in a collision-free Ethernet, cf. Section 6.

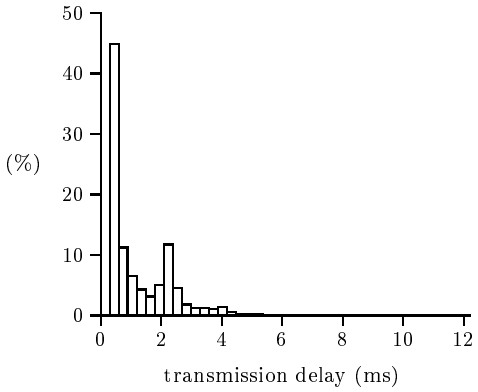


Minimum	21.7 μs
99%-minimum	21.7 μs
95%-minimum	21.7 μs
95%-maximum	22.6 μs
99%-maximum	22.9 μs
Maximum	22.9 μs
Average	22.3 μs
Std.Dev.	0.3 μs

Figure 14. Histogram of "best case" transmission delays (hardware timestamping, A203 carrier board)

The question of how pure software-based timestamping would perform in our framework can be considered as the other extreme of achieveable performance. Figure 15 shows a typical sample of the one-way transmission delay *sw_wayto* for a 4-node system with substantial network and CPU load; as in real clock synchronization, we employed simultaneous CSP initiation at all client nodes. Obviously, the resulting ε in the few 10 ms-range (note that we do not extend the x-axis of our histograms to cover the "Maximum" value in order to save space) prohibits

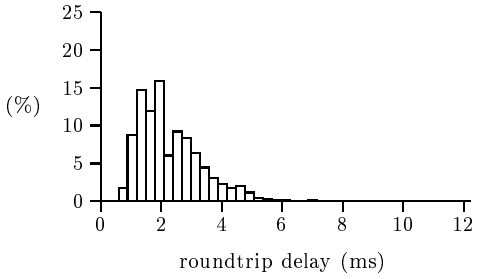
high-accuracy clock synchronization under any reasonable operating conditions. Moreover, the particular shape of the above histogram depends heavily upon most of the system parameters given in Table 2, in particular, network and CPU load.



Minimum	0.30 ms
99%-minimum	0.30 ms
95%-minimum	0.30 ms
95%-maximum	4.17 ms
99%-maximum	9.24 ms
Maximum	37.00 ms
Average	1.35 ms
Std.Dev.	1.63 ms

Figure 15. Histogram of typical transmission delays (software timestamping) in a substantially loaded 4-node system (A203 carrier board)

For the sake of completeness, we also provide the statistics of the overall roundtrip delay *sw_roundtrip*, for the same system setup as before. Figure 16 shows a typical example, which again varies heavily under different operating conditions.



Minimum	0.60 ms
99%-minimum	0.60 ms
95%-minimum	0.89 ms
95%-maximum	5.07 ms
99%-maximum	10.13 ms
Maximum	37.46 ms
Average	2.39 ms
Std.Dev.	1.64 ms

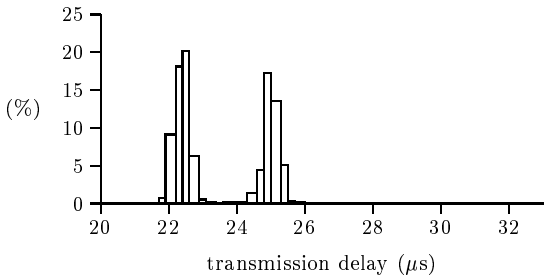
Figure 16. Histogram of typical roundtrip delays (software timestamping) in a substantially loaded 4-node system (A203 carrier board)

Comparison with Figure 15 reveals that the major contribution to excessive roundtrip delays comes from *sw_wayto*. This is primarily due to the peak load caused by simultaneous CSP initiation at all client nodes, hence vanishes if staggering is used.

4.3. Realistic Operating Conditions

This final subsection is devoted to the discussion of the most important results of our experimental evaluation, namely, the performance of the NTI's hardware

timestamping under realistic operating conditions. Figure 17 shows the one-way transmission delays of a 4-node system with substantial network and CPU load. Note that the resulting histograms, unlike Figure 15, are similar for *wayto* and *wayback* and essentially insensitive to all system parameters listed in Table 2 (apart from the fact that absolute delay values may be shifted and that the peaks may have different heights).



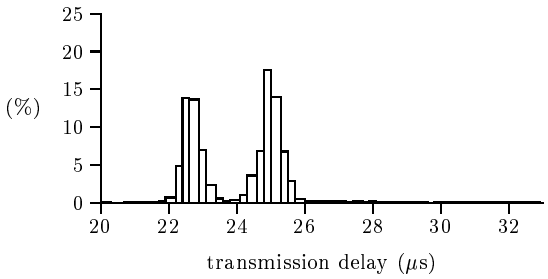
Minimum	20.4 μs
99%-minimum	21.7 μs
95%-minimum	21.9 μs
95%-maximum	25.5 μs
99%-maximum	31.0 μs
Maximum	36.8 μs
Average	23.6 μs
Std.Dev.	1.6 μs

Figure 17. Histogram of transmission delays (hardware timestamping) in a substantially loaded 4-node system (A203 carrier board)

The histogram in Figure 17 reveals three peculiarities, which call for explanation:

(A) A long tail of excessively large transmission delays (up to 36.8 μs)
(B) Two conspicuous peaks around 22.5 μs and 25 μs
(C) A short tail of small transmission delays (down to 20.4 μs)

As far as (A) is concerned, we soon recognized that the length of the right-hand side tail is different for the two available carrier-boards A203 and i6360, which are functionally equivalent but have different bus speeds. Figure 18 shows the corresponding statistics for the i6360 carrier board.



Minimum	19.2 μs
99%-minimum	21.7 μs
95%-minimum	22.2 μs
95%-maximum	26.2 μs
99%-maximum	31.9 μs
Maximum	42.4 μs
Average	24.1 μs
Std.Dev.	1.5 μs

Figure 18. Histogram of transmission delays (hardware timestamping) in a substantially loaded 4-node system (i6360 carrier board)

Comparing the above statistics, it is apparent that the ratio of the tail length of about 12 μs for the A203 vs. 18 μs for the i6360 matches the bus speed ratio of

about 600 ns/lword vs. 900 ns/lword, suggesting an interfering bus activity of at most 20 lword accesses.

This is eventually explained by the fact that the Tx-FIFO fill of a deferred transmit command can interfere with CSP reception. More specifically, upon reception, the Rx-FIFO is filled with incoming data, which are not transfered to memory by Rx-DMA until the Rx-FIFO threshold of 64 bytes has been reached. Now, if it happens that a transmit command is issued to the i82596 at a time that causes the Tx-FIFO fill to start right before the Rx-threshold is reached, Rx-DMA can be delayed (at most) for the time t_{88} required to fill 88 bytes, i.e., 22 lwords, into Tx-FIFO. Note that this is obviously a rare event, since Tx-DMA must start in the time window $[t_{Rx} - t_{88}, t_{Rx}]$ to be interfering at all. However, our experiments revealed that simultaneous CSP initiation increases this probability considerably.

Unfortunately, since the CPU issuing the transmit command does usually not know about ongoing CSP receptions, there is no easy way of avoiding[6] this pitfall. Hence, the only remedy is increasing the bus speed, which both decreases the maximum disturbance of ε and the probability of its occurrence. For example, using the maximum speed of the M-Module interface (60 ns/lword and even below), one could achieve a quite reasonable $t_{88} \leq 1.3$ μs.

It must be stressed here, however, that the interfering Tx-FIFO fill is not the only cause for (A). Actually, the latter just hides the same phenomenon that leads to the left-hand side tail (C) as well. Therefore, a remaining short tail of a few μs cannot be completely avoided.

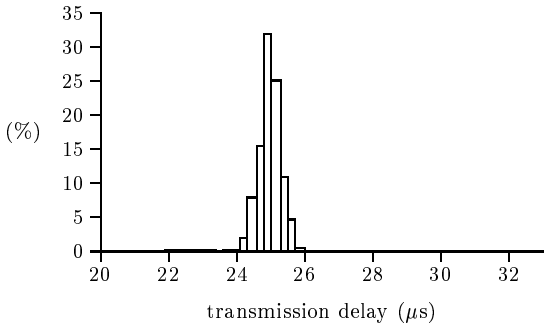
Next, we turn our attention to (B), the conspicuous peaks in the histograms of Figure 17 and 18. Being the same for both the A203 and the i6360, they must be caused by i82596-internal uncertainties and/or MVME-162-internal bus arbitration delays that do not depend upon VMEbus speed. Our measurements finally disclosed an i82596-internal uncertainty as the primary reason: Apparently, the first peak stems from CSP transmissions that find the channel idle, as is always the case in the "best-case" Figure 14, whereas the second peak is caused by CSP transmissions that find the channel busy and must wait until it becomes idle again. In fact, the i82596 seems to initiate the second Tx-DMA (which eventually draws the transmit timestamp according to Figure 13) about 2.5 μs later in case of a deferred transmission. Therefore, (B) is a "residual" of the major problem ruled out by properly selecting the TxTS trigger offset in Subsection 4.2.

What primarily backs up this claim is the different height of the peaks, i.e., their different probability masses, for certain system parameter settings. The clue to our appropriate experiments was laid by the —at first sight contradictory— symmetry of the peaks in Figure 17, which turned out to be largely independent of the network load. However, given an average network load of, say, 10 %, our explanation would rather suggest a probability > 0.5 of finding the channel idle, and hence a dominating first peak.

However, it turned out that this symmetry is in fact a consequence of simultaneous CSP initiation in conjunction with a symmetric system configuration: Two of the four nodes of the evaluation system that provided Figures 17 and 18 were equipped with an A203, whereas the remaining ones used an i6360. As CSP initiation occurs

at the same time at all nodes, the COMCOs of the two nodes with the (much faster) A203 start transmission first — after all, there is some device driver code with NTI-accesses that must be executed to start transmission of a CSP. A race for the channel occurs, which is won by any of the two A203-nodes with probability about 0.5 (the respective loosing node's transmission is deferred), and this ultimatively explains the peaks' symmetry.

The situation is slightly different for the two slower i6360-nodes, since they will usually find the channel busy by A203 CSPs. Indeed, if there was no additional load by NI- and KI-packets, Figure 18 would rather look like Figure 19. The additional NI- and KI-load, however, introduces some random staggering due to increased CPU load and evolving transmit queues, and increases the probability that all nodes will find the channel busy as well. Consequently, an i6360-node could also find the channel idle upon its CSP transmission attempt, which ultimatively explains why Figure 18 shows a first peak as well.



Minimum	21.2 μs
99%-minimum	22.4 μs
95%-minimum	24.1 μs
95%-maximum	25.7 μs
99%-maximum	26.0 μs
Maximum	40.3 μs
Average	24.9 μs
Std.Dev.	0.5 μs

Figure 19. Histogram of transmission delays (hardware timestamping) in a 4-node system without additional load (i6360 carrier board)

Finally, we turn our attention to (C), the short left-hand side tail of small transmission delays. By comparing the appropriate tail lengths of about 2 μs resp. 3 μs[7] in Figure 17 resp. 18, it is apparent that their ratio again matches the bus speed ratio of A203 resp. i6360. The effect was eventually tracked down to interfering NTI-accesses of the CPU, which could delay Tx-DMA and hence produce a slightly later TxTS. Note that Rx-DMA experiences the same interference, leading to slightly larger transmission delays. This effect, however, is completely hidden by the primary effect causing (A).

We must add, however, that we encountered some rare events with excessively small transmission delays as well: Less than, say, one CSP out of 100,000 received at an i6360-node experienced a transmission delay of about 1 μs only, which differs from the nominal 22.5 μs by more than 20 μs. Our experiments revealed that this phenomenon must be caused by the receiving node, i.e., that RxTS is drawn much earlier than usual here, since a delayed TxTS would affect all recipients simultaneously due to CSP broadcasting. Moreover, it seems as if the problem occurs in conjunction with back-to-back packet reception.

We do not have a sound explanation for this effect, but only a hypothesis: The low bus speed of the i6360 might sometimes prohibit the completion of a received packet's final Rx-DMA by the time the next packet drops in, namely, when there is an interfering Tx-FIFO fill according to (A). If this happens, the first Rx-DMA of a succeeding CSP could be joined with the deferred Rx-DMA, which means that the former is initiated before the Rx-FIFO threshold (64 bytes) is reached. The CSP's Rx-DMA could hence commence almost at the beginning of packet reception, whereas it normally starts when receiving the 64th byte. What does not fit into this simple explanation, however, is the observed pre-dating of only 20 μs, which suggests additional influences. Unfortunately, lacking information about the i82596-internal architecture prohibits further exploration of this problem.

To sum up our evaluation of the hardware timestamping capabilities of the NTI, we obtain an improvement of at least three orders of magnitude over $\varepsilon^{SW} = \max^{SW} - \min^{SW} > 10$ ms (cf. Figure 15) measured for pure software-based timestamping: For the A203, Figure 17 reveals $\varepsilon^{HW}_{95} = \max^{HW}_{95} - \min^{HW}_{95} \approx 4$ μs for 95% of all CSP transmissions, and incorporating the remaining 5% as well leads to a final $\varepsilon^{HW} = \max^{HW} - \min^{HW} \approx 17$ μs. However, the unexplained effect mentioned above reminds that rare events with excessive transmission delays could go unnoticed easily, despite of thorough experimental evaluation. Fortunately, such events can be viewed as transient faults and eventually masked out by a suitable fault-tolerant clock synchronization algorithm at a higher level, see Subsection 5.2.

5. Interval-based Clock Synchronization

A unique feature of the NTI is its support of the *interval-based paradigm* originally developed in (Marzullo, 1984) and (Lamport, 1987). Real-time t, that is, GPS time or UTC, is not just represented by a single time-dependent clock value $C(t)$ here, but rather by an *accuracy interval* $\boldsymbol{C}(t)$ that must satisfy $t \in \boldsymbol{C}(t)$. Interval-based clock synchronization thus assumes that each node p is equipped with a local *interval clock* $\boldsymbol{C}_p$ that continuously displays p's instantaneous accuracy interval $\boldsymbol{C}_p(t) = [C_p(t) - \alpha_p^-(t), C_p(t) + \alpha_p^+(t)]$, as shown in Figure 20. Naturally, $C_p(t)$ is just the UTCSU's local clock value and $\boldsymbol{\alpha}_p(t) = [-\alpha_p^-(t), \alpha_p^+(t)]$ its negative & positive accuracy, recall Subsection 3.1.

An *interval-based (external) clock synchronization algorithm* is in charge of maintaining any node's $\boldsymbol{C}_p$ such that the following can be guaranteed:

(P) *Precision requirement*: There is some *precision* $\pi_{\max} \geq 0$ such that $|C_p(t) - C_q(t)| \leq \pi_{\max}$ for all nodes p, q that are non-faulty up to real-time t.

(A) *Accuracy requirement*: The interval of accuracies $\boldsymbol{\alpha}_p(t)$ is such that $-\alpha_p^+(t) \leq C_p(t) - t \leq \alpha_p^-(t)$ for all nodes p that are non-faulty up to real-time t.

Note that (A) can be used to specify both external and internal synchronization, simply by requesting $\alpha_p^+(t)$, $\alpha_p^-(t)$ to be less than a fixed accuracy $\alpha_{\max}$ and linearly bounded w.r.t. t, respectively.

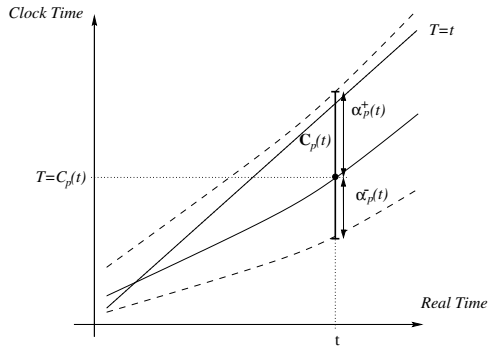


Figure 20. Basic issues of interval clocks and accuracy intervals

In the course of our SynUTC project, we developed a powerful and reasonably complete theoretical framework for fault-tolerant interval-based clock synchronization (Schmid, 1994), (Schmid and Schossmaier, 1997*a*), (Schmid and Schossmaier, 1997*b*) (Schossmaier, 1997) as well as a few particular algorithms (Schmid, 1997*b*), (Schmid, 1997*a*). In Subsection 5.2, we will very briefly sketch some basic principles; a comprehensive collection of our results can be found in (Schossmaier, 1998).

The major advantage of interval-based clock synchronization is its ability to provide each node with a local on-line bound on the own clock's deviation from real-time. A clock synchronization application, like OSF DCE (OSF, 1992), for example, can hence judge whether the instantaneous accuracy is sufficient for a certain goal, a feature that is particularly interesting for multi-clustered[8] applications. The price to be paid for this additional information, however, are explicit bounds on certain system parameters like transmission delay uncertainty — and this is where the results of our experimental evaluation come into play.

5.1. A Simple Time Distribution Algorithm

In this subsection, we show how the evaluation results of Section 4 plug into our interval-based framework (Schmid and Schossmaier, 1997*a*). Fortunately, it is sufficient to consider a very simple (non-fault-tolerant) algorithm for external clock synchronization for this purpose: In a system consisting of a single node g equipped with a GPS receiver and one or more ordinary nodes, let the interval clock of node g be continuously locked to GPS time. This is easily accomplished by adjusting $\boldsymbol{C}_g(t)$ according to the difference GPS time vs. sampled UTCSU-timestamp of every `1PPS`-pulse from the GPS receiver. Then, the following trivial algorithm can be used for time distribution:

(G) *GPS node g*: Periodically, at $C_g(t) = kP$, $k \geq 1$, node g uni- or broadcasts a CSP containing $\boldsymbol{C}_g(t)$ (= NTI g's transmit time+accuracystamp) to all other nodes in the system.

(O) *Ordinary node p*: If a CSP from node g (containing $\boldsymbol{C}_{gp} = [T_{gp} \pm \boldsymbol{\alpha}_{gp}]$) arrives at local time T_p^g (= the timestamp-part of NTI p's receive time+accuracystamp $\boldsymbol{C}_p^g$), compute

$$\boldsymbol{I}_p^g = \boldsymbol{C}_{gp} + [\delta_{gp} \pm \varepsilon_{gp}] + \boldsymbol{u} + \overline{\boldsymbol{G}} + T^R - T_p^g + (T^R - T_p^g)\boldsymbol{\rho}_p$$

and setup the UTCSU clock correction duty timer for time $T^R = kP + \Delta$, $\Delta > 0$ sufficiently large, to initiate local clock correction towards $\boldsymbol{I}_p^g$.

Note that $\Delta > 0$ secures that message reception and computation of the above interval are completed before clock correction takes place.

As explained in detail in (Schmid and Schossmaier, 1997*a*), the apparently complicated formula above expresses a few basic operations only: First, the received accuracy interval must be enlarged by ε_{gp} to account for the variable transmission delay $\delta'_{gp} \in [\delta_{gp} \pm \varepsilon_{gp}]$ (*delay compensation*). Second, when shifting the resulting interval from T_p^g to resynchronization time T^R, a sufficient enlargement ("deterioration") of the shifted interval is required to compensate the non-zero drift $\rho'_p \in \boldsymbol{\rho}_p$ of the local clock $C_p(t)$. Note that this *drift compensation* is performed continuously by the UTCSU in hardware during the whole round as well, cf. Figure 20. Finally, to cope with clock granularity resp. rate adjustment uncertainty, the intervals $\overline{\boldsymbol{G}} = [-G, 0]$ resp. $\boldsymbol{u} = [-u, u]$ are incorporated; using an oscillator with $f_o = 10$ MHz as in the evaluation setup, $u = 1/f_o = 100$ ns and $G = 2^{-23} \approx 120$ ns.

Clearly, $\boldsymbol{I}_p^g$ also gives node p's interval of accuracies $\boldsymbol{\alpha}_p^0$ immediately after resynchronization. Assuming identical $\boldsymbol{\rho}_p$ and ε_{gp} for any p for simplicity, one obtains $\boldsymbol{\alpha}_p^0 \subseteq \boldsymbol{\alpha}^0 = \boldsymbol{\alpha}_g + \varepsilon + \boldsymbol{u} + \overline{\boldsymbol{G}} + \Delta\boldsymbol{\rho}$. Moreover, the interval of accuracies immediately before the next resynchronization is bounded by $\boldsymbol{\alpha}^{\max} = \boldsymbol{\alpha}^0 + P\boldsymbol{\rho} + \boldsymbol{u} + \boldsymbol{G}$ with $\boldsymbol{G} = [0, G]$, so that the worst case precision of any two nodes in the system evaluates to $\pi = |\boldsymbol{\alpha}^{\max}|$.

Assuming an OCXO with $\boldsymbol{\rho}_p = [-10^{-7}, 10^{-7}]$, $\Delta = 100$ ms, $\boldsymbol{\alpha}_g = [-370\text{ns}, 370\text{ns}]$ and choosing $\delta_{gp} = \delta = 23.5$ μs, $\varepsilon = [-3\mu\text{s}, 14\mu\text{s}]$ according to Figure 17, the worst case accuracy and precision listed in Table 3 can be guaranteed for an NTI on a A203 carrier board.

Table 3. Worst case accuracy and precision of the simple time distribution algorithm in our evaluation system (A203 carrier board)

Period P [s]	$\boldsymbol{\alpha}^0$ [μs, μs]	$\boldsymbol{\alpha}^{\max}$ [μs, μs]	**Prec.** π [μs]
10	[-3.6, 14.5]	[-4.7, 15.7]	20.4
50	[-3.6, 14.5]	[-8.7, 19.7]	28.4
100	[-3.6, 14.5]	[-13.7, 24.7]	38.4

Note finally that the above choice of ε was guided by securing correctness for any CSP, which requires incorporating the rare events where the transmission delay is extremely large. The small standard deviation in Figure 17 shows, however, that the average accuracy/precision is about 10 times better than the worst case.

Even more, when advanced (namely, fault-tolerant) versions of the above algorithm are employed, the long tails of ε can safely be cut and $\delta = 24.2$ μs along with $\varepsilon_{95} = [-2\mu\text{s}, 2\mu\text{s}]$ be used instead.

5.2. Advanced Interval-Based Clock Synchronization

In this subsection, we will very briefly outline the basic principles of more advanced interval-based clock synchronization algorithms.

All interval-based clock synchronization algorithms developed and analyzed so far are in fact instances of a generic algorithm introduced in (Schmid and Schossmaier, 1997*a*). The latter relies upon the same round-based structure as traditional clock synchronization algorithms and consist of the following steps executed by each node p periodically:

1. When $C_p(t) = kP$, $k \geq 1$ denoting the current round, a CSP containing p's local interval clock $\boldsymbol{C}_p(t)$ is broadcast to each node in the system.
2. Upon reception of a CSP, the received accuracy interval is preprocessed to make it "compatible" with the accuracy intervals of the other nodes received during the same round.
3. When $C_p(t) = kP + \Delta$ for some suitable $\Delta > 0$, an interval-valued *convergence function* is applied to the set of preprocessed intervals to compute a new value for $\boldsymbol{C}_p(t)$; it is put into effect either instantaneously or by means of continuous amortization.
4. During the remainder of the round, p's free-running local interval clock keeps track of local time and accuracy.

Two basic operations —already introduced in Subsection 5.1— are required in Step 2, where the exchanged intervals are made compatible with each other while preserving the inclusion of real-time: First, *delay compensation* is applied to the interval received in a CSP to account for the effects of transmitting an accuracy interval over a network. To account for the maximum transmission delay uncertainty ε, the received interval must be enlarged appropriately. Second, *drift compensation* is used to shift the resulting interval to some common point in real-time by means of the local clock $C_p(t)$. Since clocks usually have a non-zero drift, a sufficient enlargement ("deterioration") of the interval is required here.

In Step 3, a suitable convergence function is applied to the set of preprocessed accuracy intervals. It is in charge of providing a new (smaller) accuracy interval for the local interval clock that guarantees conditions (P) and (A) of Section 5, despite of possibly faulty input intervals. Note that it is solely the convergence function that determines both performance and fault-tolerance degree of our interval-based clock synchronization algorithm.

Fault-tolerant external clock synchronization can be achieved by employing a convergence function based upon *interval-based clock validation* (Schmid, 1994) in the generic algorithm above: A —highly accurate but possibly faulty— accuracy interval provided by an external time source like a GPS receiver is only considered if it is

consistent with a certain —less accurate but reliable— *validation interval* derived from the interval clock readings of the ordinary nodes. Actually, a convergence function suitable for internal interval-based clock synchronization, like (Schmid, 1997*b*) or (Schmid, 1997*a*), is used for computing the validation interval. Additional measures for increasing the fault-tolerance degree of external time sources, like multiple CSPs per round or multiple GPS receivers, can of course be incorporated here as well.

The interval-based approach outlined above proved advantageous in a slightly different context as well, namely, for *clock rate synchronization*. Although accuracy intervals, which are maintained dynamically by the above clock state algorithm, are quite small on the average, it is apparent from Table 3 that our ambitious goal of a worst case precision/accuracy in the μs-range would require frequent resynchronizations. This is a consequence of the relatively large clock drift bound $\boldsymbol{\rho}$, which must be chosen conservatively enough to fully cover initial oscillator frequency offset, temperature drift, etc. Alternatively, however, clock state algorithms could use dynamic clock drift bounds provided by the interval-based clock rate synchronization algorithm introduced and analyzed in (Schossmaier, 1997), (Schossmaier, 1998). Responsible for synchronizing the speeds $dC(t)/dt$ of all non-faulty clocks, it effectively reduces the maximum drift (and hence its bounds) without necessitating highly accurate and stable oscillators at each node.

6. Relation to Other Work

Since the transmission delay uncertainty ε is one of the key factors that determine the worst-case synchronization tightness π of any clock synchronization scheme, the following classification of communication subsystems is commonly employed in the literature:

(I) If the interconnected nodes are only a few 10 meters apart, a dedicated and usually fully connected clocking "network" with small and constant propagation delays is affordable. This setting allows the construction of phase-locked-loop clocks with clock voting for increased fault-tolerance, which can provide a synchronization tightness down to the ns-range, see (Ramanathan *et al.*, 1990*b*) for an overview.

(II) Nodes within a few 100 meters of each other are usually interconnected by a packet-oriented data network, where sending data packets is the only means for distributing (time) information. Almost any work on clock synchronization addresses this type of systems, preferably for fully connected point-to-point networks, see (Ramanathan *et al.*, 1990*b*), (Simons *et al.*, 1990) for an overview and (Yang and Marsland, 1993) for a bibliography. Purely software-based clock synchronization typically achieves a synchronization tightness in the ms-range here, which can be brought down to μs with moderate hardware support.

(III) World-wide distributed systems connected via long haul networks constitute an entirely different class of systems. In fact, they have to cope with end-

to-end transmission delays that are potentially unbounded and highly variable due to queueing delays and failures at intermediate gateway nodes. The most prominent external clock synchronization scheme for this setting is undoubtly the *Network Time Protocol* (NTP) designed for disseminating UTC throughout the Internet, see (Mills, 1991), (Mills, 1995) for details. Although deterministic guarantees cannot be given here, there are reports like (Troxel, 1994) that state maximum UTC deviations in the 10 ms-range under "reasonable" conditions.

Despite of the large body of research in clock synchronization, however, there are only a few papers that deal with high-accuracy clock synchronization: The pioneering paper (Kopetz and Ochsenreiter, 1987) describes a *Clock Synchronization Unit* (CSU) chip, which accomplishes ε in the 10 μs-range in a collision-free Ethernet. In (Kopetz *et al.*, 1995), a downsized successor of the CSU targeted to the TTP fieldbus for automotive applications is outlined (see below). No implementation details are available for the hardware-assisted clock synchronization scheme (Ramanathan *et al.*, 1990*a*), which targets ε in the 100 μs-range for not necessarily fully connected point-to-point networks.

An even better ε in the 1 μs-range and below can be achieved with almost any hardware support if the network controller provides the required transmit and receive timestamp trigger signals directly. Although this cannot be expected from COTS COMCOs, there are appropriate research prototypes for TTP (Kopetz *et al.*, 1995) and LON (Horauer and Loy, 1998). For example, $\varepsilon \approx 1.9$ μs is claimed for the (collision-free) TTP running at a speed of 100 Kbit/s.

The NTI evaluated in this paper also leans on the general hardware architecture and the DMA-based method of packet timestamping proposed in (Kopetz and Ochsenreiter, 1987). Still, several "uncertainty-saving" engineering improvements have been added that guarantee ε in the 10 μs-range even for non-collision-free Ethernet. The apparent similarities between the CSU and our UTCSU, however, are only implied by the general requirements put on any hardware support for clock synchronization. In fact, our UTCSU differs from the CSU described in (Ochsenreiter, 1987) in many important ways:

- The UTCSU fully supports interval-based clock synchronization by hardware.
- Our approach aims at a worst case precision/accuracy in the μs-range, which demands considerably smaller clock granularity and fine-grained clock adjustment capabilities.
- We employ fundamentally different implementations of the vital functional units on-chip the UTCSU. The strikingly elegant and simple adder-based clock design, for example, surpasses any existing approach we are aware of. This is also true for the unwieldy clock device of (Kopetz *et al.*, 1995), which may be viewed as a concatenation of an adder-based clock and a counter, cf. (Schossmaier and Schmid, 1995).
- The UTCSU provides features like hardware support for continuous amortization and leap second insertion/deletion, which are not found in alternative approaches.

- Last but not least, the tremendous advances in VLSI technology allowed us to overcome the CSU's obvious design limitations. Our UTCSU provides dozens of wide internal registers (56+8-bit NTP-time & 16+16-bit accuracy) and many additional units like application timestamping features and interfaces to GPS receivers, which would have been impossible to accommodate in 1987.

A completely different approach to high-accuracy clock synchronization that does not need (much) hardware support but still achieves π in the 10 μs-range is the remarkable *a posteriori agreement technique* used in (Veríssimo *et al.*, 1997). Basically, it exploits the simultaneity of reception in broadcast networks for ruling out the medium access uncertainty. However, unlike the NTI, it is only applicable to shared channels with hardware broadcasting capabilities and generates considerable network and CPU load.

Relating our NTI to purely hardware-based approaches, in particular, phase-locked-loop clocks, is more difficult due to different system assumptions. In particular, PLL-clocks provide a superior accuracy/precision down to the ns-range, but require a dedicated and fully connected clocking network. Hence, solutions like (Ramanathan *et al.*, 1990*b*) cannot be used in a distributed systems like ours, where nodes are interconnected by a standard data network only.

Last but not least, in view of the negligible costs of GPS receivers, it is tempting to solve the clock synchronization problem simply by equipping each node with a modular GPS receiver. For example, (Halang and Wannemacher, 1997) proposes a real-time systems architecture that simply phase-locks an internal clock to the `1PPS`-output of a GPS timing receiver. However, although such solutions provides an excellent accuracy/precision in the 100 ns-range, it is not feasible when stringent fault-tolerance requirements are to be met.

First of all, it is generally arguable to make a pivotal distributed service like clock synchronization completely dependent upon a system with single points of failures, as present in the GPS control segment (Dana, 1997). Moreover, although GPS receivers provide reliable information most of the time, it is nevertheless true that erroneous output may occasionally occur: We conducted a 2-month continuous experimental evaluation (Höchtl and Schmid, 1997) of the output of six different GPS receivers, which confirmed that it is risky to always trust the output of a GPS receiver. In (Geier *et al.*, 1995), it was also noted that a prototype TDMA communications system at Motorola eventually broke down due to a certain GPS failure.

Apart from fault-tolerance considerations, there are also practical problems with this approach. First of all, one has to consider the effort of accommodating and connecting the "forest" of antennas required for a, say, distributed factory automation system with 100 nodes. After all, any GPS antenna needs full view of the sky and is quite sensitive to multipath reception caused by buildings and other obstacles. Techniques for antenna multiplexing might be used to reduce the number of antennas required, but such techniques reduce signal quality and can hence serve a limited number of receivers per antenna only (and introduce a single point of failure as well). Last but not least, the large time-to-fix of GPS receivers implies that it

may take 30 seconds or more until correct timing information is available. This in turn implies a large node join delay in case of re-integrating a newly powered-up or failed node.

The NTI in conjunction with interval-based clock validation provides a way to escape from the abovementioned problems, by simultaneously increasing the fault-tolerance degree and decreasing the number of GPS receivers required in the system. Since the existing data network is also used for time distribution here, additional cabling is not required. The only price to be paid, however, is decreased precision/accuracy, which is hopefully acceptable for typical applications.

7. Conclusions

In this paper, we presented a comprehensive overview of architecture and features of our *Network Time Interface* (NTI) MA-Module. The NTI supports μs-accuracy time distribution in LAN-based distributed systems and can hence be used to add fault-tolerant synchronized clocks to state-of-the-art real-time systems technology. Maximum network controller and CPU independence ensures that the NTI can be employed in virtually any COTS-based system with M-Module mezzanine interface.

The results of our thorough experimental evaluation revealed a worst case transmission delay uncertainty $\varepsilon \approx 10$ μs even in "bad" system architectures, incorporating Ethernet, slow buses, and COTS network controllers with large FIFOs. Proper choice of TxTS triggering and increasing bus speed have been identified as key issues in reducing ε, which can be brought down to the μs-range if the full speed of the NTI's M-Module interface is exploited. This shows an improvement of at least three orders of magnitude over $\varepsilon^{SW} > 10$ ms measured for pure software-based timestamping.

A simple time distribution algorithm was used to link our evaluation results with interval-based clock synchronization. Although this algorithm does not incorporate any advanced features, it still provides a worst case accuracy/precision in the 10 μs range (average case 1 μs) with negligible system overhead. More advanced algorithms are available, which use multiple GPS nodes and elaborate clock validation techniques for improving accuracy/precision and fault-tolerance degree. A forthcoming paper will be devoted to an in-depth experimental evaluation of such algorithms.

Notes

1. 60 ns granularity, 25 MHz maximum oscillator frequency, and 10 ns/s minimal rate adjustment might be considered an overkill for μs-range worst-case precision/accuracy. However, our extensive formal analysis of interval-based clock synchronization revealed that clock granularity and, in particular, rate adjustment uncertainty considerably impair achievable worst-case precision and accuracy, cf. Section 5.1.

2. There are alternatives, which, however, do not work in general. For example, one might try to move the timestamp in the packet reception ISR, where the base address of the receive buffer is of course known. Unfortunately, this might be too late for avoiding a timestamp loss in case of back-to-back CSPs. Equally inappropriate are schemes that try to exploit a sequential order of received packets, since there might be CSPs that trigger a timestamp but are eventually discarded, due to an incorrect CRC, for example.
3. We are of course aware of the fact that customary kernels are ill-suited for building up well-founded distributed real-time systems. Still, we prefer to gradually introduce sound concepts into existing systems, rather than to persuade industry to discard familiar (= ready-to-use but insufficient) technology in favor of a novel (= immature but sufficient) one.
4. A few "derived" quantities like *wayto* + *wayback* and *wayto* - *wayback* are also computed by our evaluation system.
5. Actually, VMEbus equipment from another research project was re-used for this purpose.
6. It should be mentioned that the idea of triggering RxTS by the COMCO's reception interrupt `INT`, which is the method used in all other existing approaches (see Section 6) would perform even worse here. In fact, there are technical problems that rule out this method for the i82596 at all: Apart from the difficulty of tapping the interrupt line on a COTS module like the MVME-162, `INT` signals various conditions other than packet reception as well.
7. Note that the impairment due to (C) can be as much as 5 μs in case of the i6360 carrier board, since its internal architecture occasionally causes excessive NTI access delays: An on-board M68360 processor is used to forward NTI interrupts to the VMEbus; its local bus activities can defer external VMEbus accesses by a few μs.
8. Accuracy can be used to secure precision among clusters that do not participate in a common clock synchronization algorithm: If $\boldsymbol{C}_p$, $\boldsymbol{C}_q$ at nodes p, q located in different clusters are both non-faulty, inclusion of t implies that their clock values $C_p(t)$, $C_q(t)$ cannot be further apart than $-\left(\alpha_p^-(t) + \alpha_q^+(t)\right) \leq C_q(t) - C_p(t) \leq \alpha_q^-(t) + \alpha_p^+(t)$.

References

Boehm, Barry and Chris Abts (1999). COTS integration: Plug and pray?. *IEEE Computer* **32**(1), 135–138.

Dana, Peter H. (1997). Global Positioning System (GPS) time dissemination for real-time applications. *Real-Time Systems* **12**(1), 9–40.

Geier, G. Jeffrey, T. Michael King, Howard L. Kennedy, Russel D. Thomas and Brett R. McNamara (1995). Prediction of the time accuracy and integrity of GPS timing. In: *Proceedings of the 49th IEEE International Frequency Control Symposium*. San Francisco. pp. 266–274.

Halang, Wolfgang A. and Markus Wannemacher (1997). High accuracy concurrent event processing in hard real-time systems. *J. Real-Time Systems* **12**(1), 77–94.

Höchtl, Dieter and Ulrich Schmid (1997). Long-term evaluation of GPS timing receiver failures. In: *Proceedings of the 29th IEEE Precise Time and Time Interval Systems and Application Meeting (PTTI'97)*. Long Beach, California. pp. 165–180.

Horauer, Martin and Dietmar Loy (1998). Hardware-unterstützte Uhrensynchronisation in Verteilten Systemen. In: *Proceedings AUSTROCHIP'98*. Wiener Neustadt, Austria. pp. 67–72. (ISBN 3-901578-03-X, in German).

Horauer, Martin, Dietmar Loy and Ulrich Schmid (1996). NTI functional and architectural specification. Technical Report 183/1-69. Department of Automation, Technische Universität Wien.

Horauer, Martin, Ulrich Schmid and Klaus Schossmaier (1998). NTI: A Network Time Interface M-Module for high-accuracy clock synchronization. In: *Proceedings 6th International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS'98)*. Orlando, Florida. pp. 1067–1076.

Kopetz, Hermann and Wilhelm Ochsenreiter (1987). Clock synchronization in distributed real-time systems. *IEEE Transactions on Computers* **C-36**(8), 933–939.

Kopetz, Hermann, Andreas Krüger, Dietmar Millinger and Anton Schedl (1995). A synchronization strategy for a time-triggered multicluster real-time system. In: *Proceedings Reliable Distributed Systems (RDS'95)*. Bad Neuenahr, Germany.

Lamport, Leslie (1987). Synchronizing time servers. Technical Report 18. Digital System Research Center.

Liskov, Barbara (1993). Practical uses of synchronized clocks in distributed systems. *Distributed Computing* **6**, 211–219.

Loy, Dietmar (1996). GPS-Linked High Accuracy NTP Time Processor for Distributed Fault-Tolerant Real-Time Systems. Dissertation. Technische Universität Wien. Faculty of Electrical Engineering.

Lundelius-Welch, Jennifer and Nancy A. Lynch (1984). An upper and lower bound for clock synchronization. *Information and Control* **62**, 190–204.

Mandl, Thomas, Herbert Nachtnebel and Ulrich Schmid (1999). Network Time Interface user manual. Technical Report 183/1-87. Department of Automation, Technische Universität Wien. (in German).

Marzullo, Keith A. (1984). Maintaining the Time in a Distributed System: An Example of a Loosely-Coupled Distributed Service. PhD dissertation. Stanford University. Department of Electrical Engineering.

Mills, David L. (1991). Internet time synchronization: The network time protocol. *IEEE Transactions on Communications* **39**(10), 1482–1493.

Mills, David L. (1995). Improved algorithms for synchronizing computer network clocks. *IEEE Transactions on Networks* pp. 245–254.

MUMM (1996). *ANSI/VITA 12-1996, M-Module Specification*. Manufacturers and Users of M-Modules e.V.

Nachtnebel, Herbert, Nikolaus Kerö, Gerhard R. Cadek, Thomas Mandl and Ulrich Schmid (1998). Rapid Prototyping mit programmierbarer Logik: Ein Fallbeispiel. In: *Proceedings AUSTROCHIP'98*. Wiener Neustadt, Austria. pp. 99–104. (ISBN 3-901578-03-X, in German).

Ochsenreiter, Wilhelm (1987). Fehlertolerante Uhrensynchronisation in verteilten Realzeitsystemen. Dissertation. Technische Universität Wien. Faculty of Technical and Natural Sciences. (in German).

OSF (1992). *Introduction to OSF DCE*. Prentice Hall. Englewood Cliffs, NJ.

Ramanathan, Parameswaran, Dilip D. Kandlur and Kang G. Shin (1990*a*). Hardware-assisted software clock synchronization for homogeneous distributed systems. *IEEE Transactions on Computers* **39**(4), 514–524.

Ramanathan, Parameswaran, Kang G. Shin and Ricky W. Butler (1990*b*). Fault-tolerant clock synchronization in distributed systems. *IEEE Computer* **23**(10), 33–42.

Richter, Gerda, Michael Schmidt and Ulrich Schmid (1999). i82596 NTI Device-Driver software documentation. Technical Report 183/1-90. Department of Automation, TU Vienna.

Schmid, Ulrich (1994). Synchronized UTC for distributed real-time systems. In: *Proceedings 19th IFAC/IFIP Workshop on Real-Time Programming (WRTP'94)*. Lake Reichenau, Germany. pp. 101–107.

Schmid, Ulrich (1997*a*). Interval-based clock synchronization with optimal precision. Technical Report 183/1-78. Department of Automation, Technische Universität Wien. (submitted to Information and Computation).

Schmid, Ulrich (1997*b*). Orthogonal accuracy clock synchronization. Technical Report 183/1-77. Department of Automation, Technische Universität Wien. (submitted to Chicago Journal of Theoretical Computer Science).

Schmid, Ulrich and Herbert Nachtnebel (1999). Experimental evaluation of high-accuracy time distribution in a COTS-based Ethernet LAN. In: *Proceedings 24th IFAC/IFIP Workshop on Real-Time Programming (WRTP'99)*. Schloß Dagstuhl, Germany. pp. 59–68.

Schmid, Ulrich and Klaus Schossmaier (1997*a*). Interval-based clock synchronization. *J. Real-Time Systems* **12**(2), 173–228.

Schmid, Ulrich and Klaus Schossmaier (1997*b*). Interval-based clock synchronization revisited. Technical Report 183/1-80. Technische Universität Wien. Department of Automation.

Schmid, Ulrich and Thomas Mandl (1999). Implementation of the NTI Device-Handler. Technical Report 183/1-86. Department of Automation, TU Vienna.

Schmid, Ulrich, Ed.) (1997*c*). *Special Issue on The Challenge of Global Time in Large-Scale Distributed Real-Time Systems*. J. Real-Time Systems 12(1–3).

Schossmaier, Klaus (1997). An interval-based framework for clock rate synchronization algorithms. In: *Proceedings 16th ACM Symposium on Principles of Distributed Computing*. St. Barbara, USA. pp. 169–178.

Schossmaier, Klaus (1998). Interval-based Clock State and Rate Synchronization. Dissertation. Technische Universität Wien. Faculty of Technical and Natural Sciences.

Schossmaier, Klaus and Dietmar Loy (1996). An ASIC supporting external clock synchronization for distributed real-time systems. In: *Proceedings of the 8th Euromicro Workshop on Real-Time Systems*. L'Aquila, Italy. pp. 277–282.

Schossmaier, Klaus and Ulrich Schmid (1995). UTCSU functional specification. Technical Report 183/1-56. Technische Universität Wien. Department of Automation.

Schossmaier, Klaus, Ulrich Schmid, Martin Horauer and Dietmar Loy (1997). Specification and implementation of the Universal Time Coordinated Synchronization Unit (UTCSU). *J. Real-Time Systems* **12**(3), 295–327.

Simons, Barbara, Jennifer Lundelius-Welch and Nancy Lynch (1990). An overview of clock synchronization. In: *Fault-Tolerant Distributed Computing* (Barbara Simons and A. Spector, Eds.). Springer Verlag. pp. 84–96. (Lecture Notes on Computer Science 448).

Troxel, G. D. (1994). Time Surveying: Clock Synchronization over Packet Networks. PhD thesis. Department of Electrical Engineering and Computer Science, Massachusetts Institut of Technology.

Veríssimo, Paulo, Luís Rodrigues and Antonio Casimiro (1997). CesiumSpray: a precise and accurate global clock service for large-scale systems. *J. Real-Time Systems* **12**(3), 243–294.

Yang, Z. and T. A. Marsland (1993). Annotated bibliography on global states and times in distributed systems. *ACM SIGOPS Operating Systems Review* pp. 55–72.