EXPERIMENTAL EVALUATION OF HIGH-ACCURACY TIME DISTRIBUTION IN A COTS-BASED ETHERNET LAN

Ulrich Schmid [*,1] **Herbert Nachtnebel** [**,2]

[*] *Technische Universität Wien, Department of Automation, Treitlstraße 1, A-1040 Vienna, Austria. Email: s@auto.tuwien.ac.at*
[**] *Technische Universität Wien, Department of General Electrical Engineering and Electronics, Gußhausstraße 25–29, A-1040 Vienna, Austria. Email: nachtneb@iaee.tuwien.ac.at*

Abstract: This paper reports on the experimental evaluation of the *Network Time Interface* (NTI) M-Module developed in the SynUTC-project at TU Vienna. Designed for COTS-based distributed systems, the NTI provides all hardware-related features for high-accuracy interval-based external clock synchronization, like a high-resolution rate- and state-adjustable clock, local accuracy intervals, interfaces to GPS receivers, and various timestamping features. The evaluation results show that the NTI enables GPS time distribution with μs-accuracy even in Ethernet-based systems. However, the available configuration parameters must be carefully chosen —and experimentally verified— in order to cope with the various hidden sources of timing uncertainty.

Keywords: GPS time distribution, Ethernet, experimental evaluation, interval-based external clock synchronization, COTS hardware support, M-Modules

1. INTRODUCTION

It is well-known from (Lundelius-Welch and Lynch, 1984) that the synchronization tightness achieved by any clock synchronization scheme depends primarily on the uncertainty (= variability) ε in the end-to-end transmission delay. For typical LANs, ε lies in the ms-range, which makes it impossible to use a simple packet data exchange to disseminate time with high accuracy. Additional techniques are required for this purpose, which, however, should be compatible with existing network controller technology to be useful in practice.

One of the achievements of the authors' research project SynUTC (Schmid, 1994) is a suitable add-on hardware (Horauer *et al.*, 1998) for high-accuracy time distribution in COTS-based distributed systems. A prototype implementation of this *Network Time Interface* (NTI) is available as an industry-standard M-Module (Mandl *et al.*, 1999). It was thoroughly evaluated recently in an Ethernet-coupled distributed system made up of several VMEbus-CPUs running ISI's multiprocessor real-time kernel pSOS^{+m}.

This paper reports on the major [3] results and insights gathered during this evaluation. It is organized as follows: After a brief overview of ar-

[1] The SynUTC-project received support 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. For further reference see *http://www.auto.tuwien.ac.at/Projects/SynUTC/*

[2] Supported by the Austrian *Gesellschaft für Mikroelektronik* (GMe).

[3] A forthcoming technical report (Schmid *et al.*, 1999) will contain all the details.

chitecture and features of the NTI in Section 2, the hard- and software of the evaluation system is outlined in Section 3. Section 4 presents two issues that were found to have a major impact upon ε, along with supporting measurement data. In Section 5, it is shown how the obtained results plug into a very simple interval-based time distribution algorithm. A relation to existing work in Section 6 and some conclusions in Section 7 eventually complete the paper.

2. NTI FEATURES AND ARCHITECTURE

The *Network Time Interface* (NTI) (Horauer *et al.*, 1998) has been designed for high-accuracy fault-tolerant external clock synchronization in LAN-based[4] distributed systems, cf. Section 6. Apart from advanced clock synchronization algorithms, this goal primarily requires hardware support for exact timestamping of data packets and a sophisticated rate and state adjustable clock at each node. Fig. 1 shows the basic architecture of a 2-node system.

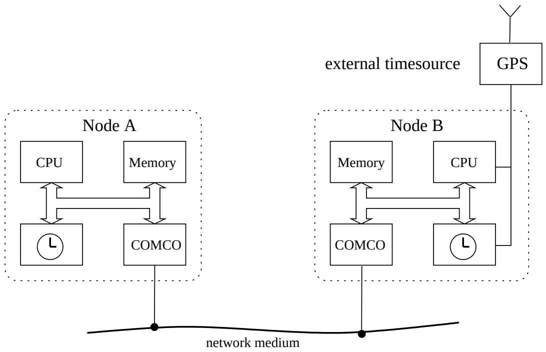


Fig. 1. *Basic architecture of a 2-node system*

Accordingly, each node must be equipped with a clock device (the UTCSU, see below), a CPU responsible for executing the clock synchronization algorithm, and a *Communications Coprocessor* (COMCO) providing access to the network by reading/writing data packets from/to (shared) memory. At least one node must also be connected to a GPS timing receiver (Höchtl and Schmid, 1997).

In purely software-based clock synchronization, timestamping of *Clock Synchronization Packets* (CSPs) at the sending resp. receiving side is done by reading the clock when assembling the CSP for transmission resp. in the packet reception interrupt service routine. However, as explained in detail in (Horauer *et al.*, 1998), this implies that the transmission delay uncertainty ε includes both the network channel access uncertainty and the reception interrupt latency, which can be quite large. To get rid of them, a refinement of the DMA-based coupling method originally proposed in (Kopetz and Ochsenreiter, 1987) can be used. The key idea is to insert a timestamp on-the-fly into the memory holding a CSP in a way that minimizes ε, as outlined in Fig. 2.

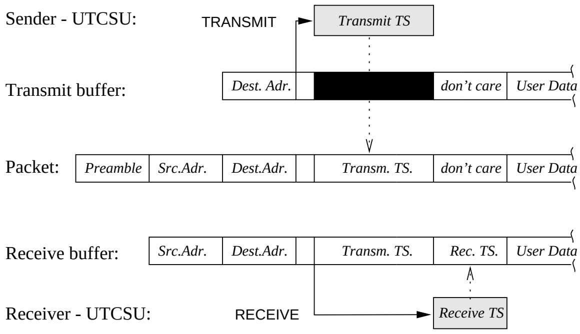


Fig. 2. *Packet timestamping*

Whenever the COMCO fetches data from the transmit buffer holding the CSP, it has to read across the particular offset that causes a special decoding logic to generate the trigger signal `TRANSMIT`. Upon its occurrence, the UTCSU puts a *transmit timestamp* (TxTS) into a dedicated sample register, which is transparently mapped into a certain portion of the transmit buffer and hence automatically inserted into the outgoing packet. Note that the trigger address and the mapping address are configurable parameters that may be different. By the same token, when the receiving COMCO writes a certain offset in the receive buffer, the trigger signal `RECEIVE` is generated by the decoding logic, which causes the UTCSU to sample the *receive timestamp* (RxTS) into a dedicated register. This timestamp is saved subsequently in an unused portion of the receive buffer.

The proposed approach works for any COMCO that accesses CSP data in memory. Suitable chipsets are available for a wide variety of networks, ranging from fieldbusses like Profibus over Ethernet up to ATM networks. It must be stressed, however, that it is usually impossible to decide *a priori* whether a particular COMCO is suitable for a certain application. In fact, the many hidden architectural intricacies make it very difficult to accurately assess ε without actual measurements.

The *Network Time Interface* (NTI), see (Mandl *et al.*, 1999) for the comprehensive user manual, provides all the hardware support required for high-accuracy clock synchronization on a single-height

[4] Note that the approach can also be adopted to more general topologies commonly known as WANs-of-LANs, provided that all gateway nodes are also equipped with the NTI, cf. (Schmid, 1994) and (Schossmaier and Schmid, 1995).

(146 x 53 mm) MA-Module[5]. Fig. 3 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.

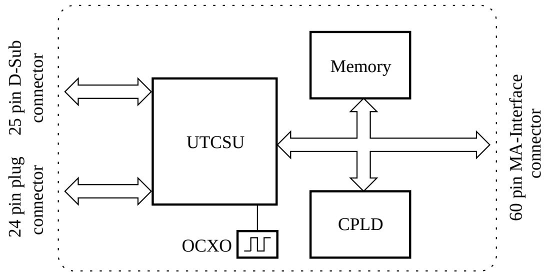


Fig. 3. *NTI block diagram*

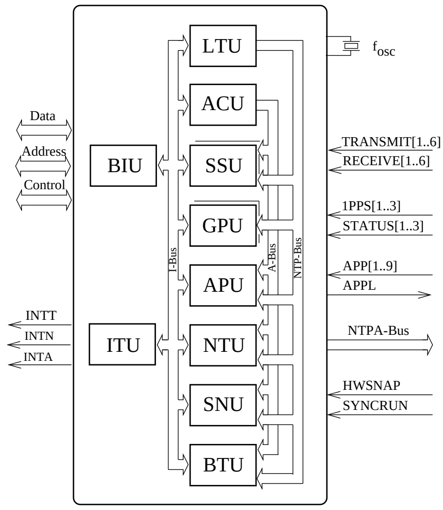
All required decoding and glue logic of the NTI is assembled in a single, in-circuit programmable *complex programmable logic device* (CPLD) designed using VHDL (Nachtnebel *et al.*, 1998). It adapts the UTCSU and the memory to the MA-Module interface, forwards interrupt requests from the UTCSU to the carrier-board, generates the acknowledgement signal terminating a bus cycle, etc.

The *memory* serves as control and data interface between the CPU and the COMCO, providing the special functionality for COMCO accesses as outlined before. It consists of up to four 64k x 16 bit SRAM chips and supports byte, word, and longword read/write accesses.

The UTCSU-ASIC (*Universal Time Coordinated Synchronization Unit*) described in (Schossmaier *et al.*, 1997), (Loy, 1996) contains most of the dedicated hardware support for clock synchronization. Manufactured as an ASIC in 0.7 μm CMOS technology, the UTCSU accommodates about $80,000$ gates on a 100 mm^2 die packed into a 208-pin PQFP case. Fig. 4 gives an overview of the major functional blocks.

The centrepiece of the UTCSU is a *local clock unit* (LTU) utilizing a 56 bit NTP-time format. Clock time can be read atomically as a 32 bit *timestamp* with resolution $2^{-24} \approx 60$ ns and a 32 bit *macrostamp* containing the remaining 24 most-significant bits + an 8 bit checksum protecting the entire time information. The LTU employs an unconventional *adder-based* clock design, which uses a 91-bit high-speed adder instead of a simple counter for summing up the elapsed time between succeeding oscillator ticks. Owing to this, the UTCSU can be paced by a quartz oscillator

[5] *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 and enhanced addressing capabilities.



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

Fig. 4. *Interior of the UTCSU*

of arbitrary frequency $f_{osc} \in 1 \ldots 20$ MHz; alternatively, an external frequency source like the 10 MHz output of a high-end GPS receiver can be used. Moreover, the local clock is fine-grained rate adjustable in steps of about 10 ns/s and supports state adjustment via continuous amortization as well as (optional) leap second corrections in hardware.

To support interval-based clock synchronization (see Section 5), the UTCSU contains two more adder-based "clocks" in the 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^+, thereby maintaining a bound on the local clock's instantaneous deviation w.r.t. real-time.

A number of external events, supplied to the UTCSU via buffered/opto-decoupled, polarity programmable input lines, can be *time/accuracy-stamped* with local time and accuracy upon the appropriate input transition. Optionally, an interrupt can be raised on such an event as well. Three different functional blocks in the UTCSU utilize this feature: First, trigger signals generated by the decoding logic at CSP transmission and reception sample the current local time/accuracy into dedicated UTCSU registers in one of the six available SSUs. Second, three independent GPUs are provided for timestamping the *one pulse per second* (1pps) signal —indicating the exact beginning of a UTC second— from up to three GPS receivers. Finally, nine independent application time/accuracy-stamping inputs are provided by the APU.

Those timestamping features are complemented by several 48 bit programmable *duty timers*: Whenever local time reaches the programmed time of an armed duty timer, an internal or external signal changes state and an optional interrupt is raised. Duty timers are required for triggering activity of the clock synchronization algorithm, for controlling continuous amortization, inserting/deleting leap seconds, and generating application-related events.

All UTCSU application time/accuracy-stamp inputs and duty timer outputs as well as all interfaces to GPS receivers are available via the NTI M-Module's front-panel 25-pin D-sub connector. In addition, all receive and transmit time/accuracy-stamp signals are fed to the carrier board via the 24-pin plug connector. High-speed opto-couplers or transceivers are provided for all inputs to ensure a decoupled and reliable interface.

3. EVALUATION SYSTEM

Aiming at the support of existing technology, it was only natural to use COTS components for building up the evaluation system as well. Each node thus consists of a Motorola MVME-162 CPU (M68040 CPU + Intel i82596CA Ethernet-controller) 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 [6] backplane. All nodes are interconnected via the CPU's Ethernet port using thin-wire technology. Fig. 5 outlines the basic hardware architecture.

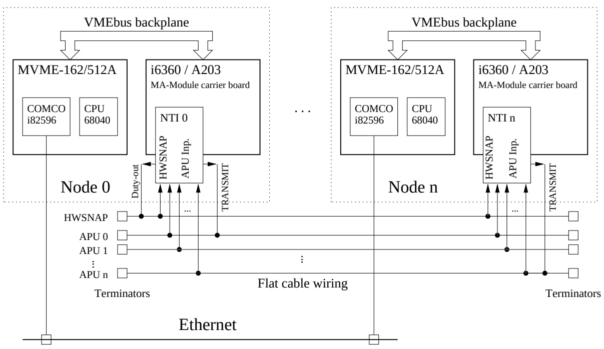


Fig. 5. *Evaluation system hardware architecture*

[6] Actually, VMEbus equipment from another research project was re-used for this purpose. Whereas the resulting architecture is unlikely to be chosen in practice, it nevertheless constitutes an suitable "worst case environment" for evaluation.

Dedicated flat cable wiring —the only add-on to Fig. 1 required for evaluation purposes— is used for measuring the (one-way) transmission delay between any two nodes in the system. More specifically, the `TRANSMIT`-signal (recall Section 2) triggering the transmit timestamp on node i's NTI is exported and fed to the application timestamp input A_i on each NTI in the system. This way, any node that receives a CSP from node i just has to compute the difference of the —locally available— timestamps RxTS − A_iTS to obtain the actual transmission delay. In addition, the `HWSNAP`-timestamp inputs of all NTIs are also tied together and driven by a duty-timer output of NTI 0. This enables the generation of simultaneous interrupts at all nodes, a feature required for initiating coordinated activity.

The evaluation system's software consists of two layers. The lower-level one is provided by the *i82596 NTI-driver* (Richter *et al.*, 1999), which integrates the NTI and the i82596 Ethernet-COMCO into ISI's pSOS^{+m} multiprocessor real-time kernel.

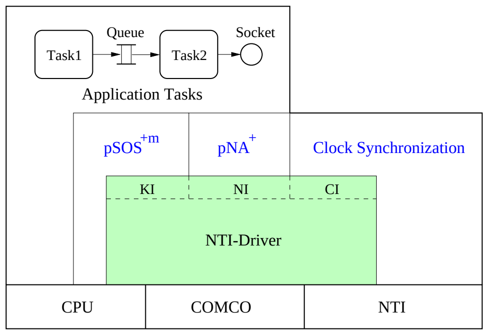


Fig. 6. *A pSOS^{+m} node using the NTI-driver*

From Fig. 6, it is apparent that the NTI-driver multiplexes three different interfaces to the i82596 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 particular communications 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; it is the only one that uses the time-stamping feature of the NTI.

Since the primary intention of the NTI-driver was transparent integration of clock synchronization into pSOS^{+m}, it also simplified the development of the evaluation system's software considerably. In fact, using the powerful features of pSOS^{+m}, the multitasking system shown in Fig. 7 was developed without much difficulty.

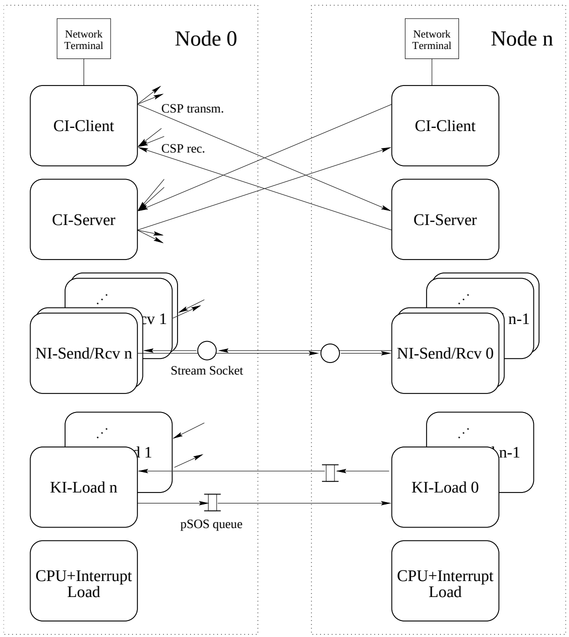


Fig. 7. *Evaluation system software architecture*

It consists of a number of concurrently executing tasks (the same at each node), which can be parameterized online 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 messages. After a specific number of rounds has been reached, the CI-client computes all transmission delay statistics and prints them on the network terminal. Among the configurable parameters are e.g. the number of rounds, broadcast period, CSP size, TxTS and RxTS trigger offsets, etc.
- 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 optional accesses to NTI memory.

Relying upon the widespread belief that using COTS components makes development easy, 6 months were projected for crafting the above evaluation system. In reality, however, about two years were spent on putting the system into full operation, and certain problems almost caused the project to fail at all. In fact, given the particular pitfalls listed below, *"COTS Integration: Plug and Pray?"* (Boehm and Abts, 1999) would have been a more appropriate belief for planning the development work:

(1) When encountering a certain misbehavior of the NTI prototype on a COTS M-Module carrier board, the NTI was obviously 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 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 to the manufacturer. Even if this can be managed, however, it depends upon the severity of the error —and the importance of the customer— whether and when a bug fix is provided.

(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. The 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 issuing the problem to the customer support, this guess was confirmed by Motorola. Accidentally, however, a clue to an i82596 test that allows 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.)

In view of this experience, we are not sure whether it was indeed a good idea to design the NTI prototype without an on-board CPU and COMCO. W.r.t. overall development costs, it might well have been the case that improved control over all components had amply brought in the increased design complexity.

4. MEASUREMENT RESULTS

Given the wealth of measurement data gathered during the various experiments, only the two most interesting findings can be presented in this paper. Further details can be found in (Schmid *et al.*, 1999).

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 bytes *Tx-FIFO* and a 128 bytes *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 (recall Section 2), however, it was to be hoped that a gross impairment could be circumvented. Fortunately, this proved to be true at the end, although it turned out that "theoretical" knowledge from the —comprehensive and quite detailed— i82596 user 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 length field) from CSP data. Therefore, it was 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.

Fig. 8 outlines the reason for this unpleasant effect. 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 a 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 channel idle.

What was not anticipated correctly is the fact that the Tx-FIFO is also completely filled when transmission is deferred, as shown in 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 has to be

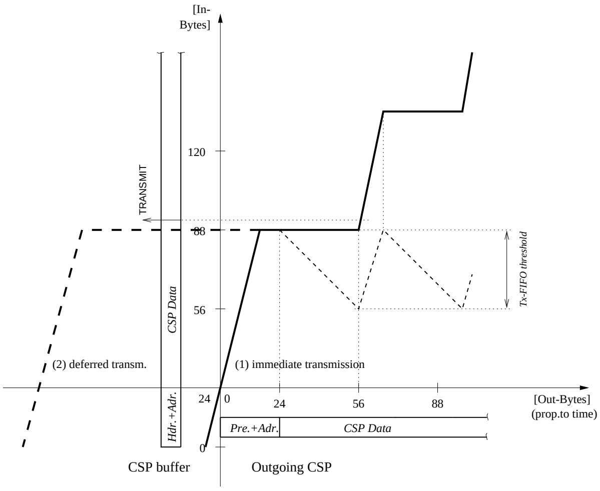


Fig. 8. *Buffer accesses during CSP transmission*

moved beyond offset 24+64=88, so that the 64 byte Tx-FIFO can be completely filled without triggering TRANSMIT. 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. The appropriate experiments confirmed that any TxTS offset > 92 performs almost equally well.

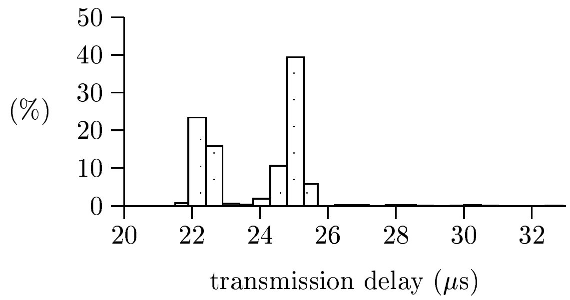


Fig. 9. *Histogram (rel. frequency) of transmission delays for MEN A203 carrier board*

Nevertheless, the resulting histograms like the one in Fig. 9 showed a disturbing long tail of rare excessive transmission delays, which needed explanation. It was soon recognized that the length of the tail is different for the two carrier-boards available, which are functionally equivalent but have different bus speeds. In fact, the ratio of the encountered maximum tail excess of about 12 μs for the A203 vs. 20 μs for the i6360 was found to match 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, some experiments revealed that coordinated activity (simultaneous CSP sends at all nodes) can increase this probability considerably at particular nodes, like at the one that produced Fig. 9.

Unfortunately, since the CPU issuing the transmit command does usually not know about ongoing CSP receptions, there is no easy way of avoiding[7] 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 NTI's M-Module interface (100 ns/lword), one could achieve a quite reasonable $t_{88} \leq 2.2\ \mu$s.

Fig. 9 finally reveals a remaining worst case timing uncertainty $\varepsilon \approx 5\ \mu$s (with a standard deviation of about 1.5 μs), which is almost the same for the A203 and the i6360. Therefore, it must be caused by bus arbitration delays and/or by i82596-internal uncertainties that do not depend upon VMEbus speed. A possible explanation of the former are MVME-162 CPU-activities that delay granting the local bus to the on-board i82596; the latter uncertainty might be caused by a different transmission initiation at the serial side in case of channel idle resp. channel busy. Unfortunately, those effects cannot be distinguished easily by means of the present evaluation setup, since MVME-162 CPUs with different local bus speeds are not available.

Anyway — comparing the above result to $\varepsilon >$ 10 ms measured for pure software-based timestamping reveals an improvement of a factor of 1000, which seems to be remarkably enough.

5. TIME DISTRIBUTION ALGORITHM

A unique feature of the NTI/UTCSU is its support of the interval-based paradigm (Marzullo, 1984), where real-time t (e.g. GPS time or UTC) is represented by an *accuracy interval* $\boldsymbol{A}(t)$ satisfying $t \in \boldsymbol{A}(t)$. Referring to Section 2, a node p's accuracy interval $\boldsymbol{A}_p(t) = [C_p(t) - \alpha_p^-(t), C_p(t) + \alpha_p^+(t)]$ is formed by adding its UTCSU's clock value $C_p(t)$ and its interval of accuracies $\boldsymbol{\alpha}_p(t) = [-\alpha_p^-(t), \alpha_p^+(t)]$. Exploiting this on-line bound on the local clock's deviation from real-time, a clock synchronization application can 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 the experimental evaluation come into play.

During the last few years of working on the SynUTC project, a thorough theoretical framework for fault-tolerant interval-based clock synchronization was established, see (Schmid, 1994), (Schmid and Schossmaier, 1997), (Schossmaier, 1997), (Schmid, 1997), etc.; a comprehensive collection of those results can also be found in (Schossmaier, 1998). To link this research with the NTI and the evaluation results, it is sufficient to consider a very simple (non-fault-tolerant) algorithm for external synchronization: In a system consisting of a single node g equipped with a GPS receiver and one or more ordinary nodes, let the UTCSU-clock of node g be continuously locked[9] to GPS time. 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 $\boldsymbol{A}_g(t)$ (= NTI g's transmit time/accuracystamp) to all other nodes in the system.

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

$$\begin{aligned}\boldsymbol{I}_p^g = \boldsymbol{A}_{gp} + [\delta_{gp} \pm \boldsymbol{\varepsilon}_{gp}] + \boldsymbol{u} + \overline{\boldsymbol{G}} \\ + T^R - T_p^g + (T^R - T_p^g)\boldsymbol{\rho}_p\end{aligned}$$

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 $\Delta > 0$ must secure that message

[7] It should be mentioned that the idea of triggering RxTS by the reception interrupt signal `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.

[8] Accuracy can be used to secure precision among clusters that do not participate in a common clock synchronization algorithm: If $\boldsymbol{A}_p$, $\boldsymbol{A}_q$ 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 $-\left(\alpha_p^-(t) + \alpha_q^+(t)\right) \leq C_q(t) - C_p(t) \leq \alpha_q^-(t) + \alpha_p^+(t)$.

[9] This is easily accomplished by adjusting node g's UTCSU-clock according to the difference GPS time vs. sampled 1pps UTCSU-timestamp upon every 1pps-pulse from the GPS receiver.

reception and computation of the above interval are completed before clock correction takes place.

As explained in detail in (Schmid and Schossmaier, 1997), 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. Finally, to cope with clock granularity resp. rate adjustment uncertainty, the intervals $\overline{\boldsymbol{G}} = [-G, 0]$ resp. $\boldsymbol{u} = [-u, u]$ are provided; 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 $\boldsymbol{\alpha}_p$ immediately after resynchronization. Assuming identical $\boldsymbol{\rho}_p$ and ε_{gp} for any p for simplicity, one obtains $\boldsymbol{\alpha}_p \subseteq \boldsymbol{\alpha} = \boldsymbol{\alpha}_g + \varepsilon + \Delta\boldsymbol{\rho} + \boldsymbol{u} + \overline{\boldsymbol{G}}$. Moreover, the interval of accuracies immediately before the next resynchronization is bounded by $\boldsymbol{\alpha}' = \boldsymbol{\alpha} + P\boldsymbol{\rho}$, so that the worst case precision of any two nodes in the system evaluates to $\pi = |\boldsymbol{\alpha}'|$.

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 $\delta_{gp} = \delta = 22.5$ μs, $\boldsymbol{\varepsilon} = [-3\mu\text{s}, 12\mu\text{s}]$ according to the A203's evaluation results in Section 4, the following worst case accuracy and precision can be guaranteed in the evaluated setting:

P [s]	$\boldsymbol{\alpha}$ [μs,μs]	$\boldsymbol{\alpha}'$ [μs,μs]	π [μs]
10	[-3.6,12.5]	[-4.6,13.5]	18.1
50	[-3.6,12.5]	[-8.6,17.5]	26.1
100	[-3.6,12.5]	[-13.6,22.5]	36.1

The above choice of $\boldsymbol{\varepsilon}$ was guided by securing correctness for any CSP, which requires incorporating the rare cases where the transmission delay is extremely large. The small standard deviation shows, however, that the average accuracy/precision is about 10 times better than the worst case. When more advanced (namely, fault-tolerant) versions of the above algorithm are employed, the long tail of ε can be cut. Note that there are several directions for such an improvement, ranging from multiple CSPs over multiple GPS receivers up to advanced clock validation algorithms (Schmid, 1994).

6. RELATION TO OTHER WORK

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

(I) Nodes within a few 10 meters of each other can be connected by a dedicated clocking "network". Phase-locked-loop clocks with clock voting for increased fault-tolerance can be used here to provide π in the 10 ns-range (Ramanathan *et al.*, 1990*b*).

(II) Nodes within a few 100 meters of each other are usually interconnected by a packet-oriented communications subsystem. Sending data packets is the only means for exchanging (time) information here. Purely software-based solutions typically achieve π in the ms-range, which can be brought down to μs with moderate hardware support.

(III) World-wide distributed systems connected via long haul networks exhibit highly variable and even potentially unbounded end-to-end transmission delays. This effectively rules out any bounded π. Nevertheless, an accuracy in the 10 ms-range can be achieved by means of the prominent *Network Time Protocol* NTP (Mills, 1995) under "reasonable" conditions, see (Troxel, 1994).

There is a large body of research in clock synchronization, which primarily addresses type (II) systems. However, only a few papers deal with high-accuracy clock synchronization, i.e., hardware support for decreasing ε.

In their pioneering paper (Kopetz and Ochsenreiter, 1987), the authors describe a *Clock Synchronization Unit* (CSU) chip that ensures ε in the few 10 μs-range if used in a collision-free Ethernet. A downsized successor of the CSU targeted to the TTP fieldbus for automotive applications is briefly outlined in (Kopetz *et al.*, 1995) (see below). No implementation details are available for the hardware-assisted clock synchronization scheme (Ramanathan *et al.*, 1990*a*), which claims ε in the 100 μs-range for not necessarily fully connected point-to-point networks.

An even better ε in the 1 μs-range can be achieved with 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) fieldbusses. 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 standard (= non-collision-free) Ethernet. Note also that the NTI's UTCSU-ASIC does not have much in common with the original CSU chip and its successor: The UTCSU not only relies upon the interval-based paradigm but employs fundamentally different building blocks internally as well, like the high-resolution adder-based clocks.

A completely different approach 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 channel access uncertainty. However, unlike the NTI, it is only applicable to networks with hardware broadcasting capabilities and generates considerable network and CPU load.

Finally, 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. However, apart from fault-tolerance considerations (Höchtl and Schmid, 1997), there are 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. In addition, 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 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. This is basically done without additional (cabling) costs, since it uses the existing data network only. The price to be paid is additional hardware support and decreased precision/accuracy, which is hopefully acceptable for typical applications.

7. CONCLUSIONS

In this paper, some results and insights gained during the experimental evaluation of the NTI M-Module supporting the distribution of GPS time over LANs have been presented. As opposed to the typical worst case transmission delay uncertainty $\varepsilon \approx 10$ ms of purely software-based solutions, the NTI provides $\varepsilon \approx 10$ 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 1 μs-range if the full speed of the NTI's M-Module interface is exploited.

A simple time distribution algorithm has been sketched to link the evaluation results with interval-based clock synchronization. Although it does not incorporate any advanced feature, it still provides an accuracy/precision in the 10 μs range 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 those algorithms.

ACKNOWLEDGEMENTS

This work could not have been done without vital support from all the other members of the NTI development team, namely, Thomas Mandl, Michael Schmidt, Gerhard R. Cadek and Nikolaus Kerö. The invaluable assistance of Johann Klasek is also gratefully acknowledged.

8. REFERENCES

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

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

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

Horauer, Martin, Ulrich Schmid and Klaus Schossmaier (1998). NTI: A Network Time Interface M-Module for high-accuracy clock synchronization. In: *Proc. 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: *Proc. Reliable Distributed Systems (RDS'95).* Bad Neuenahr, Germany.

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, TU Vienna.

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. (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: *Proc. Austrochip 1998.* Wiener Neustadt, Austria. pp. 99–104. (ISBN 3-901578-03-X).

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: *Proc. IFAC Workshop on Real-Time Programming (WRTP'94).* Lake Reichenau, Germany. pp. 101–107.

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

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

Schmid, Ulrich, Johann Klasek, Thomas Mandl, Herbert Nachtnebel, Gerhard R. Cadek and Nikolaus Kerö (1999). A Network Time Interface M-Module for distributing GPS-time over LANs. Technical report. Technische Universität Wien. Department of Automation. (forthcoming).

Schossmaier, Klaus (1997). An interval-based framework for clock rate synchronization algorithms. In: *Proc. 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 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.

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.