How to Distribute GPS-Time over COTS-based LANs

Ulrich Schmid
Department of Automation
Technische Universität Wien
Treitlstraße 1, A-1040 Vienna, Austria
tel ++43-1-58801-18325 fax ++43-1-58801-18391
e-mail s@auto.tuwien.ac.at

Martin Horauer
Institute of Computer Technology
Technische Universität Wien
Gußhausstraße 25–27
A-1040 Vienna, Austria
tel ++43-1-58801-38416
fax ++43-1-58801-38499
e-mail horauer@ict.tuwien.ac.at

Nikolaus Kerö
Department of Applied Electronics and Quantum Electronics
Technische Universität Wien
Gußhausstraße 25–27
A-1040 Vienna, Austria
tel ++43-1-58801-35956
fax ++43-1-58801-35999
e-mail keroe@iaee.tuwien.ac.at

Abstract

This paper shows how to distribute GPS-time with μs-accuracy and below even in Ethernet-based distributed systems. Our SynUTC-approach[1] *is based upon a simple network controller-level hardware for timestamping data packets as they leave and arrive at a node, which comes in two flavours: The memory-based timestamping method exploited by our* Network Time Interface (NTI) *M-Module timestamps data packets as the network controller accesses them in memory. This technique can be used for virtually any type of network and network controllers. For 10 Mb/s Ethernet, for example, our experimental evaluation revealed a time distribution accuracy down to the μs-range. Still, memory-based timestamping requires network controllers that cannot store entire packets on-chip, and the available configuration parameters must be carefully chosen in order to cope with the various hidden sources of timing uncertainty. To escape from those limitations, we recently developed a novel MII-based timestamping method that can be used in conjunction with almost any modern 10/100 Mb/s Ethernet chipset. The timestamping hardware sits at the standard MII-interface between network controller and transceiver here, and will allow a time distribution accuracy well below the μs-range.*

[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, the *Gesellschaft für Mikroelektronik* (GMe), and the START programme Y41-MAT. See *http://www.auto.tuwien.ac.at/Projects/SynUTC/* for further information.

1 INTRODUCTION

Next-generation distributed real-time applications will certainly be equipped with accurately synchronized clocks at every node. In fact, synchronous data acquisition and simultaneous triggering of actuators at several nodes is impossible without such a feature. Given the advances in GPS technology, it is no longer an issue to supply highly accurate time information to dedicated computing nodes. However, accurately and reliably disseminating GPS-time to *all* nodes in a distributed system is still a challenge.

As more and more distributed real-time systems are now being built atop COTS-type LANs, the most desirable solution would be using the data network for time distribution, as done e.g. in NTP [Mil91]. However, [LWL84] revealed that the worst-case synchronization tightness achieved by any clock synchronization scheme depends on the worst-case uncertainty (= maximum variability, jitter) 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 μs-accuracy. Additional techniques are required for this purpose, which, however, should be compatible with existing network controller technology to be useful in practice.

Part of our research project SynUTC [Sch94] is devoted to this problem. Among its results is our *Network Time Interface* (NTI) [SKM+00] add-on hardware, which is available as an industry-standard M-Module [MUM96]. The NTI has been built around our custom UTCSU-ASIC [SSHL97], which contains most of the hardware support required for interval-based external clock synchronization in fault-tolerant distributed systems: A high-resolution state- and rate-adjustable clock, local accuracy intervals, interfaces to GPS receivers, and various timestamping features.

In [SN99] and [SKM+00], we showed that a worst-case ε in the 10 μs-range can be achieved when using the NTI in a 10 Mb/s Ethernet-coupled distributed system made up of Motorola MVME-162 CPUs. Although ε can be brought down to the few μs-range in more suitable system architectures, it became nevertheless clear that memory-based timestamping in conjunction with existing network interfaces is limited both with respect to applicability and achievable time distribution accuracy.

In this paper, we present and analyze a complete algorithm for distributing GPS-time with a few μs-range accuracy in our NTI-based testbed. In addition, we introduce a novel MII-based timestamping method, which will be employed in a second generation 10/100 Mb/s Ethernet-NTI that is currently under development. The presented material is organized as follows: After a brief overview of the NTI and its pivotal UTCSU in Section 2, we introduce our evaluation system's hardware and software architecture in Section 3. Section 4 is devoted to the particular time distribution algorithm used. A brief survey of our novel MII-based timestamping method in Section 5 and some conclusions in Section 6 eventually complete the paper.

2 NTI FEATURES AND ARCHITECTURE

The *Network Time Interface* (NTI) [SKM+00] has been designed for high-accuracy fault-tolerant external clock synchronization in LAN-based[2] distributed systems. Apart from advanced clock synchronization algorithms, this goal primarily requires hardware

[2]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. [Sch94] and [SS95].

support for exact timestamping of data packets and a sophisticated rate and state adjustable clock at each node. Figure 1 shows the basic architecture of a 2-node system.

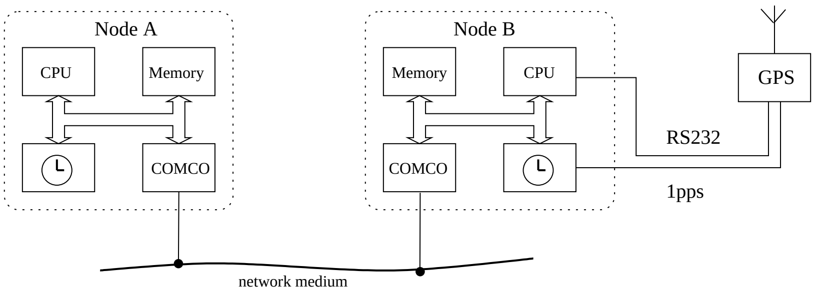


Figure 1: *Basic architecture of a 2-node system*

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.

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 COMCO's packet reception interrupt service routine. However, as explained in detail in [SKM+00], 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 those, a refinement of the DMA-based coupling method originally proposed in [KO87] 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 Figure 2.

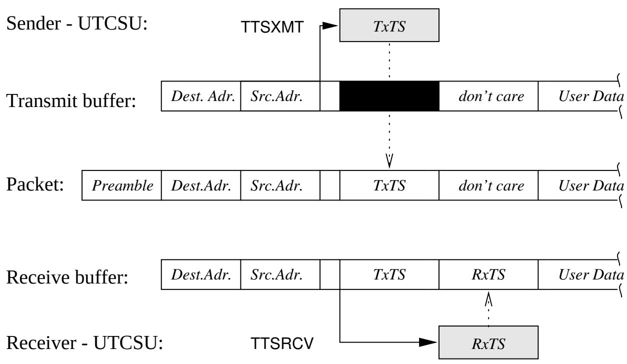


Figure 2: *Packet timestamping*

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 `TTSXMT`. 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. By the same token, when the receiving COMCO writes a certain offset in 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. This timestamp is saved subsequently in an unused portion of the receive buffer.

This approach works for any COMCO that accesses CSP data in memory. Suitable chip-sets are available for a wide variety of networks. It must be stressed, however, that COMCOs with a transmit-FIFO that can hold an entire packet defeat memory-based timestamping. Moreover, determining ε for a particular COMCO usually requires experimental evaluation.

The NTI provides the above memory-based timestamping facilities —as well as all other hardware support required for high-accuracy clock synchronization— on a single-height (146 x 53 mm) MA-Module[3]. Figure 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; details can be found in the comprehensive user manual [MNS99].

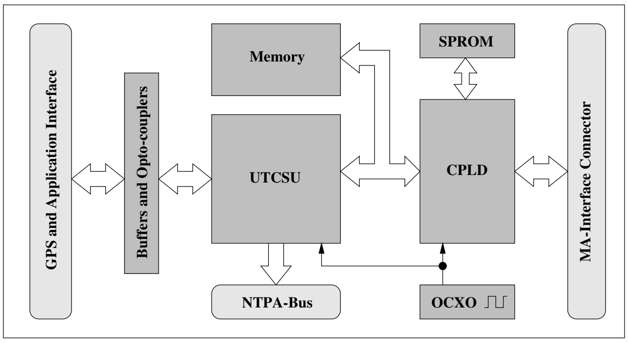


Figure 3: *NTI block diagram*

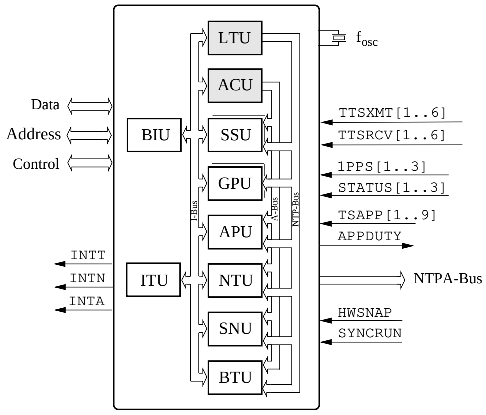
All required decoding and glue logic of the NTI is assembled in a single, in-circuit programmable *complex programmable logic device* (CPLD) designed using VHDL. It adapts the UTCSU and the memory to the MA-Module interface, forwards interrupt requests from the UTCSU to the carrier-board, generates the 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 [SSHL97] and [Loy96] contains most of the dedicated hardware support for clock synchroniza-

[3] *M-Modules* [MUM96] implement an open, simple, and robust mezzanine bus interface primarily designed for VME carrier boards. *MA-Modules* are enhanced M-Modules, providing a 32 bit data bus and enhanced addressing capabilities.

tion. 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 MQFP case. Figure 4 gives an overview of the major functional blocks.



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 4: *Interior of the UTCSU*

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 a resolution of $2^{-24} \approx 60$ 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 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 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 leap second corrections in hardware.

To support interval-based clock synchronization (see Section 4), 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 second— from up to three GPS timing 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 application-related I/O-pins of the UTCSU as well as all interfaces to GPS receivers are routed to the GPS and application interface. In addition, the UTCSU's internal time information ("NTPA-bus") is exported for future expansion modules. Of course, high-speed opto-couplers or buffers 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 our evaluation testbed as well. As in [SN99] and [SKM+00], we employ[4] VMEbus-based nodes consisting of a Motorola MVME-162 CPU and an AcQ i6360 or MEN A203 passive carrier-board hosting the NTI MA-Module. All nodes are interconnected via the CPU's Ethernet port using 10 Mb/s thin-wire technology. Figure 5 outlines the basic hardware architecture.

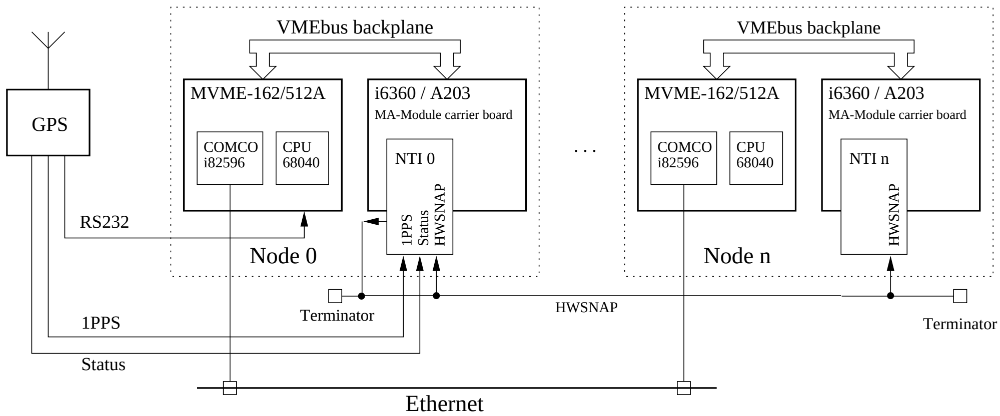


Figure 5: *Evaluation system hardware architecture*

A dedicated wire —the only add-on to Figure 1 required for evaluation purposes— is provided for simultaneous sampling of the current time at all nodes. It connects the `HWSNAP`-timestamp inputs of all NTIs and is driven by an output of a dedicated master node (NTI 0).

The evaluation system's software consists of several layers shown in Figure 6. Two layers of driver software written in C make the NTI's features available to the clock synchronization algorithm: The lower-level *NTI-Handler* [SM99] is responsible for initialization/configuration of the NTI and the M-Module carrier-board as well as any

[4] 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 a suitable "worst case environment" for evaluation.

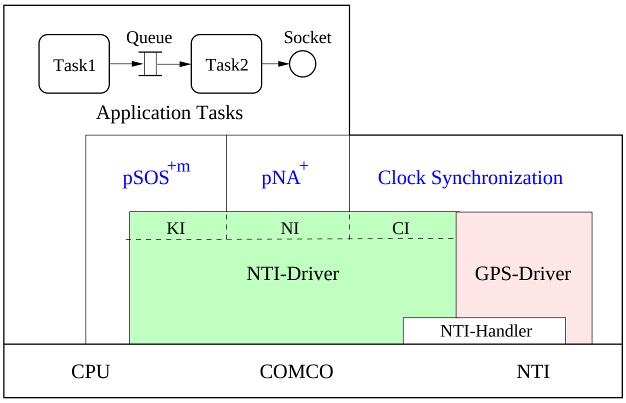


Figure 6: *Architecture of a pSOS^{+m} node using the NTI- and GPS-Driver*

low-level interrupt handling. The upper-level driver layer consists of the *NTI-Driver* and the *GPS Device-Driver*.

The *NTI-Driver* [RSS99] allows the pSOS^{+m} kernel and other software components to communicate via the MVME-162's i82596 Ethernet-COMCO with or without data packet timestamping. It actually multiplexes three different interfaces to the COMCO:

1. *Kernel Interface* (KI): pSOS^{+m} supports multiprocessing by means of remote objects (tasks, queues, semaphores, etc.), which are implemented atop of *remote procedure calls* (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.

The *GPS Device-Driver* [US99] is responsible for interfacing one or more GPS timing receiver(s) to a pSOS^{+m}-based target system equipped with at least one NTI. As shown in Figure 5, it supports GPS timing receivers that provide an RS232 serial interface, a 1 pps output and an optional digital status signal. The 1 pps signal announces when GPS-time/UTC advances to the next second, the status signal indicates a valid/non-valid output. The RS232 data, which is usually supplied several 10 ms after the 1 pps, reveals the point-in-time of the last 1 pps transition, usually along with additional status information. The GPS-Driver assumes that the RS232 interface is connected to a standard pSOS^{+m} serial channel, whereas the 1 pps and status output are fed to one of the three GPUs of the NTI's UTCSU, recall Section 2.

The GPS Device-Driver provides the clock synchronization with an easy to use *application programmable interface* (API) through which a GPS receiver can be controlled and time information retrieved. After initialization of the NTI M-Module, the RS232 interface and the GPS receiver, the driver processes the 1 pps transitions and RS232 messages and calls a user-supplied callback function that periodically delivers a submicrosecond-accurate timestamp of the last 1 pps transition. Note carefully that this accuracy can only be achieved due to the hardware timestamping feature of the GPU. Special attention is paid to fault recognition (lost or wrong 1 pps pulses, lost or wrong RS232 messages) and error handling, since wrong 1 pps $\leftrightarrow$ RS232 message assignments would result in a 1 second-range error.

Viewed at the level of application tasks, the NTI/GPS-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. Still, complex distributed timing problems can now be solved by means of the various timestamping and duty timer functionalities of the UTCSU's APU.

4 TIME DISTRIBUTION ALGORITHM

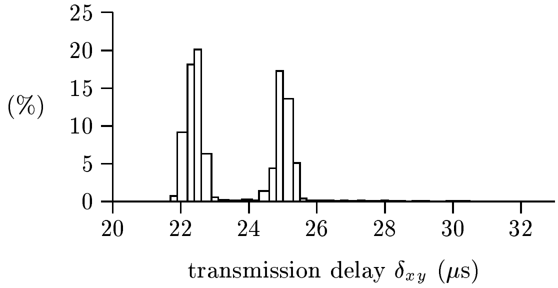
During the last few years of working on the project SynUTC, we established a reasonably complete framework for fault-tolerant interval-based clock synchronization [Sch94], [SS97], [Sch97c], [Sch97b], [Sch97a], [SW99] etc. Real-time t (= GPS-time or UTC) is not just represented by an ordinary clock $C_x(t)$ at node x here, but rather by an *interval clock* $\boldsymbol{C}_x(t) = [C_x(t) \pm \boldsymbol{\alpha}_x(t)]$ made up of the UTCSU's clock value $C_x(t)$ and its interval of accuracies $\boldsymbol{\alpha}_x(t) = [-\alpha_x^-(t), \alpha_x^+(t)]$. An interval-based clock synchronization algorithm is responsible for maintaining any $\boldsymbol{C}_x(t)$ such that *accurateness* w.r.t. real-time, i.e., $t \in \boldsymbol{C}_x(t)$, as well as mutual *precision*, i.e., $|C_x(t) - C_y(t)| \leq \pi_{\max}$, can be guaranteed.

The major advantage of the interval-based approach is a locally available on-line bound on the local clock's deviation from real-time: Since $t \in \boldsymbol{C}_x(t)$ just means $t \in [C_x(t) - \alpha_x^-(t), C_x(t) + \alpha_x^+(t)]$, an application can judge whether the instantaneous accuracy is sufficient for a certain goal. This feature is particularly interesting for multi-clustered applications, since accuracy w.r.t. real-time implies a certain precision even for nodes that do not participate in a common clock synchronization algorithm: If nodes x, y in different clusters are both non-faulty, inclusion of t implies that their intervals must be overlapping. Hence, clock values $C_x(t)$, $C_y(t)$ cannot be further apart than $-\big(\alpha_x^-(t) + \alpha_y^+(t)\big) \leq C_y(t) - C_x(t) \leq \alpha_y^-(t) + \alpha_x^+(t)$.

The price to be paid for this accuracy information, however, are explicit bounds on certain system parameters like transmission delay uncertainty ε — and this is where the results of an experimental evaluation like [SN99] and [SKM$^+$00] come into play. Figure 7 shows the transmission delay uncertainty ε observed for the NTI in the evaluation system of Figure 5. Thanks to the hardware timestamping capabilities of the NTI, the primary transmission delay characteristics[5] are independent of the network load. It is apparent, however, that excessively long transmission delays occur now and then. This forced us to choose a conservative bound for ε in the simple algorithm of [SKM$^+$00], which thus suffers from a relatively poor worst case accuracy.

In this paper, we show how to improve accuracy by means of a slightly more ad-

[5]Note that the second peak is caused by packets that find the Ethernet busy upon their arrival.



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 7: *Histogram of transmission delays (A203 carrier board)*

vanced algorithm using multiple-source time distribution combined with round-trip-based transmission delay measurement: Consider a system consisting of $m \geq 1$ *P-nodes* $p_1, \ldots, p_m$ equipped with a GPS receiver, and one or more *S-nodes* that are to be synchronized with GPS-time. Assume that:

- node p's GPS receiver has accuracy $\boldsymbol{\alpha}_p$,
- each UTCSU is driven by an oscillator with frequency f_o, resulting in a clock granularity $\overline{\boldsymbol{G}} = [-G, 0]$ with $G = 2^{-\lceil \log_2(f_o) \rceil}$ and a rate adjustment uncertainty $\boldsymbol{u} = [-u, u]$ with $u = 1/f_o$ (see [SS97, Assum. 2]),
- any node x's UTCSU is set up properly to ensure correct deterioration of the local interval of accuracies $\boldsymbol{\alpha}_x$ when the maximum local oscillator drift is within $\boldsymbol{\rho}_x = [-\rho_x^-, \rho_x^+]$ (see below) and discrete rate adjustment with $\boldsymbol{u}$ applies (see [SS97, Assum. 3]),
- the transmission delay from node x to y is $\delta'_{xy} \in [\overline{\delta}_{xy} \pm \varepsilon_{xy}]$, with unknown —but constant or slowly varying— expectation $\overline{\delta}_{xy} = \overline{\delta}_{yx} + d_{yx}$, known unsymmetry d_{yx}, and known maximum uncertainty $\boldsymbol{\varepsilon}_{xy} = [-\varepsilon_{xy}^-, \varepsilon_{xy}^+]$,
- at most $f \geq 0$ of the $m \geq 2f + 1$ P-nodes may deliver wrong time information to an S-node due to GPS failures, P-node failures, excessive transmission delays, etc.

Table 1 shows the particular parameter settings for our evaluation system when all nodes are equipped with an A203 carrier board.

Name	Value	Meaning
f_o	10 MHz	UTCSU oscillator frequency
u	100 ns	rate adjustment uncertainty
G	$2^{-23} \approx 120$ ns	clock granularity
$\boldsymbol{\rho}_x$	$\boldsymbol{\rho} = [-10^{-7}, 10^{-7}]$	maximum oscillator drift
$\boldsymbol{\alpha}_p$	$\boldsymbol{\alpha}_0 = [-150\text{ns}, 150\text{ns}]$	accuracy Motorola VP-Oncore GPS receiver [HS97]
$\boldsymbol{\varepsilon}_{xy}$	$\boldsymbol{\varepsilon} = [-3\mu\text{s}, 3\mu\text{s}]$	transmission delay uncertainty (see below)
d_{xy}	d=0	average unsymmetry
Δ	300 ms	maximum transmission delay
Γ_s	$\Gamma = 100$ ms	maximum computation time S-nodes
Γ_p	$\Gamma_0 = 10$ ms	maximum computation time P-nodes

Table 1: *Parameter settings for our evaluation system (identical nodes with A203 carrier board)*

Figure 8: *Time distribution algorithm: P-nodes*

(L) *Lock local clock to GPS-time*

Upon every 1pps transition occurring at GPS-time t_i with $C_p(t_i) = T_i$,

1. compute

$$\begin{aligned} \boldsymbol{I}_p &= [t_i \pm \boldsymbol{\alpha}_p] + \boldsymbol{u} + \overline{\boldsymbol{G}} \\ &\quad + T^P - T_i + (T^P - T_i)\boldsymbol{\rho}_p \end{aligned}$$

for $T^P = T_i + \Gamma_p$, where $\Gamma_p > 0$ compensates the execution time of this step,

2. SetupClockCorrection(T^P, $\boldsymbol{I}_p$).

(R) *Receive round-trip CSP from S-nodes*

Upon reception, at local time T_p^s, of a CSP containing the local time T_{sp} of transmission at S-node s, save (s, T_{sp}, T_p^s).

(D) *Distribute GPS-time to S-nodes*

Periodically, when $C_p(t) = kP$ in round k,

1. BroadcastCSP(S, $\forall s : (s, T_{sp}, T_p^s)$)
2. $k := k + 1$.

Figure 9: *Time distribution algorithm: S-nodes*

(S) *Send round-trip CSP to P-nodes*

Periodically, when $C_s(t) = kP - \Delta$ in round k, BroadcastCSP(P, $\emptyset$).

$\Delta > 0$ must allow reception at any P-node before step (D), and should ensure that there is no step (L) during this time.

(G) *Get GPS-time from P-nodes*

Upon reception, at local time T_s^p, of a CSP from node p containing $\boldsymbol{C}_{ps} = [T_{ps} \pm \boldsymbol{\alpha}_{ps}]$ and (s, T_{sp}, T_p^s) in round k,

1. compute $\Delta T_s = T_s^p - T_{sp}$, $\Delta T_p = T_{ps} - T_p^s$ and the current transmission delay estimator

$$\delta_{ps} = \frac{\Delta T_s - \Delta T_p - d_{ps}}{2} \quad (1)$$

and update the corresponding running averages $\overline{\Delta T}_s$, $\overline{\Delta T}_p$, $\overline{\delta}_{ps}$.

2. compute

$$\begin{aligned} \boldsymbol{I}_s^p &= \boldsymbol{C}_{ps} + \boldsymbol{u} + \overline{\boldsymbol{G}} \\ &\quad + [\overline{\delta}_{ps} \pm \varepsilon_{ps}] + \overline{\Delta T}_s \boldsymbol{\rho}_s + \overline{\Delta T}_p \overline{\boldsymbol{\rho}}_p \\ &\quad + T^R - T_s^p + (T^R - T_s^p)\boldsymbol{\rho}_s, \quad (2) \end{aligned}$$

where $\overline{\boldsymbol{\rho}}_p = [-\rho_p^+, \rho_p^-]$ and $T^R = kP + \Delta + \Gamma_s$. $\Delta > 0$ and $\Gamma_s > 0$ must allow message reception and completion of all computations before step (C).3.

(C) *Compute the final correction interval*

At time $C_s(t) = kP + \Delta$ in round k,

1. initiate local clock reading delivering $\boldsymbol{C}_s = [T_s \pm \boldsymbol{\alpha}_s]$ and compute

$$\boldsymbol{I}_s = \boldsymbol{C}_s + T^R - T_s + (T^R - T_s)\boldsymbol{\rho}_s,$$

2. compute

$$\boldsymbol{I}_s^R = \mathcal{CV}_m^f(\boldsymbol{I}_s^1 \cap \boldsymbol{I}_s, \ldots, \boldsymbol{I}_s^m \cap \boldsymbol{I}_s), \quad (3)$$

3. if $\boldsymbol{I}_s^R \neq \emptyset$ then SetupClockCorrection(T^R, $\boldsymbol{I}_s^R$),
4. $k := k + 1$.

Basic operations:

- *BroadcastCSP(X, data)* unreliably sends a CSP containing *data* to all X-nodes, either by hardware broadcasting or multiple point-to-point transmissions. Each CSP is time+accuracystamped with the local interval clock upon actual transmission.
- *SetupClockCorrection(time, interval)* sets up the UTCSU's clock correction dutytimer for time *time* to initiate local interval clock correction towards *interval.*
- $\mathcal{CV}_m^f(\boldsymbol{I}_1, \ldots, \boldsymbol{I}_m)$ provides a new value for the local interval clock. It computes a fault-tolerant intersection [Sch97b] and [SS99] of its input intervals that tolerates at most f faulty or non-existing intervals, and selects a suitable point (usually, the midpoint) within that interval as the new clock value.

The time distribution algorithm shown in Figure 8 and 9 proceeds in rounds of duration P seconds, which are indexed by $k \geq 1$ and scheduled according to each node's local clock. Apart from locking a P-node's UTCSU-clock to GPS-time upon every 1pps transition in step (L), each round ends with a CSP exchange (S) $\rightarrow$ (R) and (D) $\rightarrow$ (G) between any S- and P-node, followed by a resynchronization of the S-node's interval clock in step (C). The CSP round-trip is used for both measuring[6] the transmission delay and for distributing GPS-time.

The formulas given in Figure 8 and 9 try to preserve a given interval's accurateness by elementary techniques developed in our interval-based framework, see [SS97] for an elaborate discussion. For example, to ensure accurateness of the remote clock estimation eq. (2), the accuracy interval $\boldsymbol{C}_{ps}$ received from node p must be enlarged by ε_{ps} to account for the variable transmission delay $\delta'_{ps} \in [\overline{\delta}_{ps} \pm \varepsilon_{ps}]$ (*delay compensation*); the terms involving $\overline{\Delta T}_s$ and $\overline{\Delta T}_p$ compensate drift-induced errors in the computation of $\overline{\delta}_{ps}$. In addition, when shifting the resulting interval from T_s^p to resynchronization time T^R, a sufficient enlargement (deterioration) of the shifted interval is required to compensate $C_s(t)$'s drift $\rho'_s \in \boldsymbol{\rho}_s$. Recall that this *drift compensation* is performed continuously by the UTCSU in hardware as well.

The remote clock estimations $\boldsymbol{I}_s^p$ are intersected with the S-node's own information $\boldsymbol{I}_s$ and eventually fed into the convergence function $\boldsymbol{\mathcal{CV}}$ in eq. (3). This step ultimately allows us to neglect the rare excessive transmission delays apparent in Figure 7 and choose $\varepsilon_{ps} = [-3\mu\text{s}, 3\mu\text{s}]$ in Table 1 instead: Since $\boldsymbol{\mathcal{CV}}$ tolerates up to f faulty input arguments, a larger f (i.e., a larger $m \geq 2f+1$) allows to choose a more risky ε_{ps} without increasing the probability of an S-node failure. The intersection with $\boldsymbol{I}_s$ constitutes a simple clock validation technique [Sch95], which is effective for detecting excessive faults even in case of $f = 0$. Note that accurateness of $\boldsymbol{CV}$'s arguments is not affected by this intersection as long as the S-node s is non-faulty.

It is not difficult to analyze the worst case accuracy and precision of our algorithm. Since we established in [Sch97b] and [SS99] that the result of a suitable convergence function is contained in the intersection of its $m-2f$-largest *non-faulty* input intervals, it follows that $\boldsymbol{I}_s^p$ yields node s's interval of accuracies $\boldsymbol{\alpha}_s$ immediately after resynchronization. Plugging in the simplified parameters from Table 1 and the obvious bounds $\overline{\Delta T}_s, \overline{\Delta T}_p \leq \Delta$, we obtain

$$\begin{aligned}\boldsymbol{\alpha}_s \subseteq \boldsymbol{\alpha} &= \Big(\boldsymbol{\alpha}_0 + 2\boldsymbol{u} + \overline{\boldsymbol{G}} + 1\text{s}\cdot\boldsymbol{\rho}\Big) + \Big(\boldsymbol{u} + \overline{\boldsymbol{G}} + \boldsymbol{\varepsilon} + (3\Delta + \Gamma)\boldsymbol{\rho}\Big)\\ &= \boldsymbol{\alpha}_0 + 3\boldsymbol{u} + 2\overline{\boldsymbol{G}} + \boldsymbol{\varepsilon} + (3\Delta + \Gamma + 1\text{s})\boldsymbol{\rho}.\end{aligned}$$

Moreover, the interval of accuracies immediately before the next resynchronization is bounded by

$$\boldsymbol{\alpha}' = \boldsymbol{\alpha} + P\boldsymbol{\rho} + \boldsymbol{u} + \boldsymbol{G}$$

for $\boldsymbol{G} = [0, G]$, so that the worst case precision of any two non-faulty S-nodes in the system evaluates to $\pi = |\boldsymbol{\alpha}'|$. Plugging in our particular parameter values finally gives the worst case accuracy and precision shown in Table 2. It is instructive to compare this table with [SKM+00, Table 3], which shows the corresponding results when correctness is secured for any CSP transmission by choosing $\boldsymbol{\varepsilon} = [-3\mu\text{s}, 14\mu\text{s}]$. It is apparent that our more advanced algorithm considerably improves the achievable worst-case accuracy.

[6]Note that we omit the initial, say, 10–20 round-trips needed for setting up the running averages for simplicity.

P [s]	$\boldsymbol{\alpha}$ [μs,μs]	$\boldsymbol{\alpha}'$ [μs,μs]	π [μs]
10	[-3.9,3.7]	[-5.0,4.9]	9.9
50	[-3.9,3.7]	[-9.0,8.9]	17.9
100	[-3.9,3.7]	[-14.0,13.9]	27.9

Table 2: *Worst case accuracy and precision*

Still, the roughly Bernoullian transmission delay distribution in Figure 7 suggests that the average case accuracy should be considerably better than the above worst case results indicate. To verify this conjecture, we developed a pSOS^{+m} evaluation application atop of the general software architecture of Figure 6, which allowed us to monitor and statistically evaluate the synchronization accuracy of our time distribution algorithm: A master node's duty timer is used to generate periodic state transitions at the HWSNAP-signal fed to all nodes in the system, cf. Figure 5. A dedicated task at each node reads the timestamps triggered by HWSNAP-transitions and sends it to a dedicated evaluation task at the master node for processing. The interested reader is referred to [APS00] for further details and our evaluation results.

5 NTI ENHANCEMENTS

Whereas the μs-range accuracy achieved by our memory-based NTI implementation should be sufficient in most cases, we are aware of some more demanding applications. The currently most challenging one is on-line fault location in power distribution grids, i.e., the problem of finding out when and, in particular, where a fault (cable break, short circuit, partial discharge etc.) occurred in a —usually buried— power cable. One promising solution is to attach detectors at the power trunks in each power/transformer station, which timestamp the instant when the transient wave emanating from a fault location arrives. By relating the timestamps gathered at both ends of a cable, it is possible to determine the fault location on-line within a few 10 meters. Still, as the transient waves travel about 200 meters/μs, a precision in the 10 ns-range is mandatory for this application.

However, our experimental evaluation [SN99] and [SKM+00] revealed that memory-based timestamping cannot be improved by the required two orders of magnitude: Large on-chip FIFOs found in modern COMCOs severely impair ϵ, and if a COMCO can store an entire packet in its transmit FIFO, the method does not work at all. Another drawback of our current NTI is the required CPU intervention for moving receive timestamps into the CSP, cf. [HSS98]: Since receive time+accuracystamps are sampled into dedicated UTCSU-registers, they must be moved into an unused portion of the received packet before the next one drops in. The present NTI uses a high-priority interrupt service routine for this purpose, which, however, generates a high interrupt load on the CPU in case of back-to-back receptions. Moreover, as M-Modules provide a single interrupt line only, any UTCSU-interrupt is forced to be a high-priority one.

In order to circumvent those shortcomings, we recently developed a novel *Media Independent Interface*-based timestamping method that can be used in conjunction with almost any modern 10/100 Mb/s Ethernet chipset. Figure 10 outlines the basic architecture of our next generation MII-NTI, which will be built as a PCI board. Note that supporting Fast Ethernet technology was considered mandatory, since it is widely used in office automation and becomes increasingly popular in other areas as well. For example, the *Fieldbus Foundation* (*http://www.fieldbus.org*) and the *Industrial Automation Open Network Alliance* (IAONA, *http://www.iaona-eu.com/*) are trying to establish Fast Ethernet technology in the area of fieldbusses and automation, respectively.

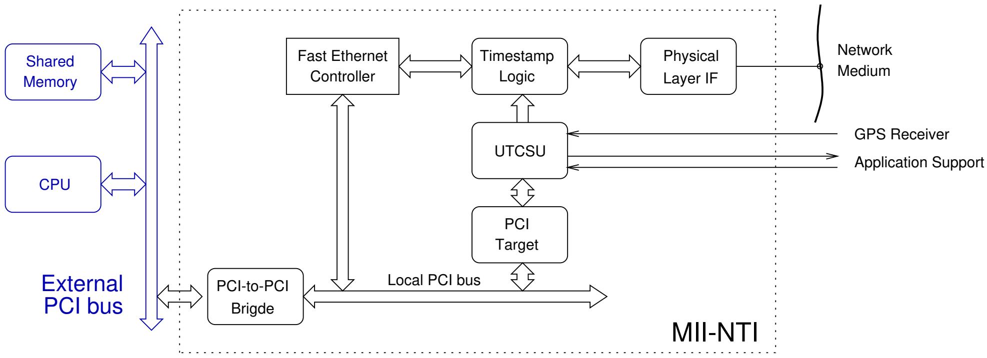


Figure 10: *Basic architecture of the MII-NTI*

MII-based timestamping exploits the fact that almost all Fast Ethernet controllers support the standard IEEE 802.3-compliant *Media Independent Interface* (MII) to physical layer devices. The *Timestamp Logic* outlined in Section 2 is placed into this MII data path, i.e., timestamps are now inserted on the network side of the COMCO rather than on its CPU side. More specifically, controlled by a state machine that recognizes CSPs by means of a special type field, time+accuracystamps are latched from the UTCSU's multiplexed NTPA-bus, serialized and then inserted transparently into the MII data stream. Since this is also done upon reception, no CPU intervention is required for moving the receive timestamps anymore.

The MII-NTI neither incorporates the *CPU* that executes the clock synchronization software nor the *Shared Memory* that holds the data packets, cf. Figure 1. Appropriate host components must be present instead, which can access the board via its PCI interface. A dedicated *Local Bus* is provided on-board the NTI to ensure that both the Fast Ethernet controller and the UTCSU can be accessed by the CPU. As Fast Ethernet chipsets are usually equipped with an integrated PCI interface, we chose PCI as our local bus and use a *PCI Target* chip for mediating accesses to the UTCSU. However, PCI boards may only host a single PCI interface, hence a *PCI-to-PCI Bridge* is finally used for interfacing to the external PCI bus.

All remaining interfaces and devices are implemented in a similar fashion as on the current NTI M-Module.

We are reasonably convinced that the MII-NTI will eventually provide a transmission delay uncertainty ε in the few 10 ns-range. Combined with a faster UTCSU and more advanced clock synchronization algorithms, we should therefore be able to achieve a precision and accuracy in the 10 ns-range. A forthcoming paper will elaborate on the detailed description and experimental evaluation of our solution.

6 CONCLUSIONS

We presented an overview of our SynUTC time distribution approach, which facilitates fault-tolerant distribution of GPS-time even in Ethernet-based distributed systems. Rather than equipping each node with a modular GPS receiver, we rely upon a small network-controller-level add-on hardware that allows exact timestamping of data packets as they leave and arrive at any node. With memory-based timestamping, as employed in our NTI M-Module, a worst-case synchronization accuracy down to the μs-range can be achieved. Compared with pure software-based timestamping, this constitutes an improvement of three orders of magnitude. Our novel MII-based timestamping method will allow to improve this already remarkable result even further. Apart from achieving a synchronization accuracy in the 10 ns-range, it can also be applied to 100 Mb/s Ethernet and network controllers with large FIFOs.

The NTI in conjunction with interval-based clock synchronization algorithms also overcomes both fault-tolerance limitations and practical problems inherent in any "dedicated receiver"-solution: GPS-receivers do deliver wrong 1pps pulses now and then [HS97], and the large time-to-fix may cause a joining delay of 30 seconds or more for newly powered up nodes. Last but not least, the "forest" of antennas required for a, say, distributed factory automation system with 100 nodes is difficult to accommodate and connect. Our approach simultaneously increases the fault-tolerance degree and decreases the number of GPS receivers required in the system — without additional (cabling) costs, by using the existing data network only. The only price to be paid is some moderate hardware support and decreased precision/accuracy, which is hopefully acceptable for most applications.

7 REFERENCES

References

[APS00] Daniel Albeseder, Gerold Pawelka, and Ulrich Schmid. Evaluation of NTI-based GPS-time distribution over Ethernet. Technical Report 183/1-97, Department of Automation, TU Vienna, January 2000. (forthcoming).

[HS97] Dieter Höchtl and Ulrich Schmid. 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)*, pages 165–180, Long Beach, California, December 1997.

[HSS98] Martin Horauer, Ulrich Schmid, and Klaus Schossmaier. NTI: A Network Time Interface M-Module for high-accuracy clock synchronization. In *Proceedings 6th International Workshop on Parallel and Distributed Real-Time Systems (WP-DRTS'98)*, pages 1067–1076, Orlando, Florida, March-April 1998.

[KO87] Hermann Kopetz and Wilhelm Ochsenreiter. Clock synchronization in distributed real-time systems. *IEEE Transactions on Computers*, C-36(8):933–939, 1987.

[Loy96] Dietmar Loy. *GPS-Linked High Accuracy NTP Time Processor for Distributed Fault-Tolerant Real-Time Systems.* Dissertation, Technische Universität Wien, Faculty of Electrical Engineering, April 1996.

[LWL84] Jennifer Lundelius-Welch and Nancy A. Lynch. An upper and lower bound for clock synchronization. *Information and Control*, 62:190–204, 1984.

[Mil91] David L. Mills. Internet time synchronization: The network time protocol. *IEEE Transactions on Communications*, 39(10):1482–1493, October 1991.

[MNS99] Thomas Mandl, Herbert Nachtnebel, and Ulrich Schmid. Network Time Interface user manual. Technical Report 183/1-87, Department of Automation, Technische Universität Wien, January 1999. (in German).

[MUM96] MUMM. *ANSI/VITA 12-1996, M-Module Specification.* Manufacturers and Users of M-Modules e.V., 1996.

[RSS99] Gerda Richter, Michael Schmidt, and Ulrich Schmid. i82596 NTI Device-Driver software documentation. Technical Report 183/1-90, Department of Automation, TU Vienna, February 1999.

[Sch94] Ulrich Schmid. Synchronized UTC for distributed real-time systems. In *Proceedings 19th IFAC/IFIP Workshop on Real-Time Programming (WRTP'94)*, pages 101–107, Lake Reichenau, Germany, 1994.

[Sch95] Ulrich Schmid. Synchronized Universal Time Coordinated for distributed real-time systems. *Control Engineering Practice*, 3(6):877–884, 1995. (Reprint from Proceedings 19th IFAC/IFIP Workshop on Real-Time Programming (WRTP'94), Lake Reichenau/Germany, 1994, p. 101–107.).

[Sch97a] Ulrich Schmid. Interval-based clock synchronization with optimal precision. Technical Report 183/1-78, Department of Automation, Technische Universität Wien, July 1997. (submitted to Information and Computation).

[Sch97b] Ulrich Schmid. Orthogonal accuracy clock synchronization. Technical Report 183/1-77, Department of Automation, Technische Universität Wien, March 1997. (submitted to Chicago Journal of Theoretical Computer Science).

[Sch97c] Klaus Schossmaier. An interval-based framework for clock rate synchronization algorithms. In *Proceedings 16th ACM Symposium on Principles of Distributed Computing*, pages 169–178, St. Barbara, USA, August 1997.

[SKM+00] Ulrich Schmid, Johann Klasek, Thomas Mandl, Herbert Nachtnebel, Gerhard R. Cadek, and Nikolaus Kerö. A Network Time Interface M-Module for distributing GPS-time over LANs. *J. Real-Time Systems*, 18(1), 2000. (to appear).

[SM99] Ulrich Schmid and Thomas Mandl. Implementation of the NTI Device-Handler. Technical Report 183/1-86, Department of Automation, TU Vienna, January 1999.

[SN99] Ulrich Schmid and Herbert Nachtnebel. 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)*, pages 59–68, Schloß Dagstuhl, Germany, May/June 1999.

[SS95] Klaus Schossmaier and Ulrich Schmid. UTCSU functional specification. Technical Report 183/1-56, Technische Universität Wien, Department of Automation, July 1995.

[SS97] Ulrich Schmid and Klaus Schossmaier. Interval-based clock synchronization. *J. Real-Time Systems*, 12(2):173–228, March 1997.

[SS99] Ulrich Schmid and Klaus Schossmaier. How to reconcile fault-tolerant interval intersection with the Lipschitz condition. Technical Report 183/1-96, Department of Automation, TU Vienna, September 1999. (submitted to Distributed Computing).

[SSHL97] Klaus Schossmaier, Ulrich Schmid, Martin Horauer, and Dietmar Loy. Specification and implementation of the Universal Time Coordinated Synchronization Unit (UTCSU). *J. Real-Time Systems*, 12(3):295–327, May 1997.

[SW99] Klaus Schossmaier and Bettina Weiss. An algorithm for fault-tolerant clock state & rate synchronization. In *Proceedings 18th IEEE Symposium on Reliable Distributed Systems (SRDS'99)*, pages 36–47, Lausanne, Switzerland, 1999.

[US99] Martina Umlauft and Ulrich Schmid. GPS Device-Driver software documentation. Technical Report 183/1-97, Department of Automation, TU Vienna, October 1999.