# PSynUTC - Evaluation of a High Precision Time Synchronization Prototype System for Ethernet LANs

**Martin Horauer**
Technikum Vienna

Höchstädtplatz 3, A-1200 Vienna
horauer@technikum-wien.at

**Klaus Schossmaier, Ulrich Schmid**
Department of Automation
Vienna University of Technology
Treitlstraße 1, A-1040 Vienna
Klaus.Schossmaier@cern.ch, s@auto.tuwien.ac.at

**Roland Höller, Nikolaus Kerö**
Department of Computer Technology
Vienna University of Technology
Gusshausstr. 27-29, A-1040 Vienna
{Roland.Hoeller, Nikolaus.Keroe}@tuwien.ac.at

**Abstract**

*This article presents an overview and some evaluation results of our PSynUTC[1] prototype system for GPS time distribution and time synchronization in Ethernet-based LANs. PSynUTC does not need dedicated GPS receivers at every computing node but uses ordinary data packets for disseminating time information. High accuracy is achieved by combining (1) hardware packet timestamping at the network interface of nodes and switches, (2) high-resolution, high-frequency adder-based clocks with superior adjustment capabilities, and (3) clock rate synchronization algorithms compensating typical TCXO drifts. Our technology can be employed with any network controller chipset that supports the media-independent interface (MII) standard. Moreover, standard device drivers and protocol stacks can be used without any change. Our evaluation results show that PSynUTC will achieve a worst case synchronization accuracy in the 100 ns range, which is an improvement of at least 4 orders of magnitude over conventional software-based approaches like NTP.*

## 1 Introduction

Designing distributed systems is considerably simplified if accurately synchronized clocks are available. Temporally ordered events are in fact beneficial for a wide variety of tasks, ranging from synchronous data acquisition and simultaneous trigger-

[1]The SynUTC-project (*http://www.auto.tuwien.ac.at/Projects/SynUTC/*) 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. Further development of our technology, including PSynUTC, has now been taken over by our spin-off company Oregano Systems (*http://www.oregano.at*).

ing of actuators at different computing nodes up to fully-fledged distributed algorithms [Lis93]. Applications with very high accuracy requirements are also known: Examples are carrier/data synchronization in high-speed and wireless networks (see e.g. *http://www.datum.com/timepieces/*) or on-line fault locating in power distribution grids [SHK99]. In the latter example the arrival times of the transient wave emanating from a possible break or short circuit at both endpoints of a (typically buried) high-voltage power cable are measured. Since transient waves travel about 200 meters/$\mu$s, a precision in the 10 ns-range is required here.

The enabling technology for such applications is also at hand: Due to GPS, it is no longer an issue to supply highly accurate time information to some computing nodes. Disseminating GPS-time to *all* nodes in a distributed system is a challenge, however: Dedicated receiver solutions suffer from a "forest" of antennas, poor fault-tolerance properties, and large node power up delays [SKM+00]. Solutions that disseminate GPS-time from a few sites to the remaining nodes by some other means are hence preferable.

For example, one could use CDMA cellular/wireless networking technology for time distribution within buildings (see e.g. *http://www.endruntechnologies.com/*). Still, most modern distributed systems are built atop of wireline LANs like Ethernet. Today, switched Fast Ethernet technology dominates business applications and is even penetrating the industrial automation (see *http://www.iaona-eu.com/*) and fieldbus domain. The most desirable solution for time distribution in such systems is using the data network, as done e.g. in NTP [Mil91]. Still, it is well-known that the worst case synchronization tightness achieved by any clock synchronization scheme depends on the worst case uncertainty (= maximum variability, jitter) $\varepsilon$ in the end-to-end transmission delay [LWL88]. For typical LANs, $\varepsilon$ lies in the ms-range, which makes it impossible to use a simple packet data exchange for disseminating time with high accuracy. Additional techniques are required for this purpose, which, however, must be compatible with existing network controller technology to be useful in practice.

Our research project SynUTC has been devoted to this problem. Apart from establishing a reasonably complete theoretical and algorithmic framework [Sch95, SS97, Sch97b, Sch00, SS01, SW99], we also developed prototypes for the required hardware support: Using our custom UTCSU-ASIC [SSHL97], our *Network Time Interface* (NTI) M-Module [SKM+00] provided 10 $\mu$s-range worst case synchronization precision in 10 Mb/s Ethernet networks. Further research [SHK99, Hor01] led to a new MII timestamping method, which not only increases synchronization precision by some additional 1-2 orders of magnitude but works in switched Ethernet networks as well.

This paper presents an overview of the PSynUTC prototype system, which is currently being built atop of our latest technology. It is organized as follows: After a brief overview of the challenges of high-accuracy clock synchronization in Section 2, we provide an overview of PSynUTC in Section 3. In the following sections, we address each of the major requirements for high accuracy individually: Section 4 is devoted to our MII timestamping technology, Section 5 surveys the benefits of our adder-based clock approach, and Section 6 shows how to deal with the relatively large drift of the TCXO oscillators that typically drive our clocks. Some conclusions and directions of further work in Section 7 eventually complete the paper.

## 2 Challenges of High-Accuracy Time Synchronization

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 [YM93] for a bibliography. If synchronized clocks must also maintain a well-defined and close relation ("accuracy") to some external time standard like *Coordinated Universal Time* (UTC), then the *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 received increased attention with the advent of GPS technology; a quite comprehensive collection of related research can be found in [Sch97a].

In principle, internal clock synchronization is rather simple: Consider a distributed system where every node $p$ is equipped with an adjustable clock $C_p(t)$ that is driven by a quartz oscillator with maximum drift $\rho$. If the nodes' clocks run freely for some period of time $P$, they could deviate from each other by up to $2P\rho$. Periodic resynchronization can nevertheless enforce a bounded *precision* $\pi$ securing $\forall t : |C_p(t) - C_q(t)| \leq \pi$ for any two non-faulty nodes $p, q$: *Clock synchronization packets* (CSPs) are periodically sent at local times $C_p(t) = k \cdot P,\ k = 1, 2, \ldots$ for this purpose, which are timestamped at the sender upon departure and at the receiver upon arrival. Since both timestamps are contained in the CSP, knowing the transmission delay allows to determine the difference of the receiver's clock and any remote sender's clock ("remote clock reading"). Note that the maximum clock reading error is equal to the transmission delay uncertainty $\varepsilon$ here. Therefore, a correction value can eventually be computed and applied to any receiver's clock, which ensures that all clocks in the system will be closely synchronized again.

When system time must also have a well-defined relation to external time, there is a promising alternative to this "one-dimensional" point of view sufficient for internal synchronization, namely, the *interval-based paradigm* [SS97]. Interval-based algorithms represent time information relating to an external standard like UTC by intervals that are known (better to say supposed) to contain UTC. Given a set of such intervals from different sources, a usually smaller interval that actually contains UTC may be determined, even if some of the source intervals are faulty [Sch00, SS01]. Interval-based clock synchronization algorithms in fact maintain an interval clock $\boldsymbol{C}_p(t) = [C_p(t) - \alpha_p^-(t), C_p(t) + \alpha_p^+(t)]$ at any node $p$, which not only provides clock time $C_p(t)$ by also online bounds on negative and positive accuracy $\alpha_p^-(t)$ and $\alpha_p^+(t)$, i.e., the local clock's deviation from real-time.

The worst case analysis of the achievable synchronization precision $\pi$ of our algorithms [Sch00] revealed—in accordance with conventional clock synchronization research [FC95]—that

$$\pi = c_1\varepsilon + c_2 G + c_3 u + c_4 P\rho \tag{1}$$

where $c_1, c_2, c_3, c_4$ are small integer constants depending upon the particular algorithm. Herein,

1. $\varepsilon$ denotes the transmission delay uncertainty (determining the remote clock reading error; ms-range for Ethernet),
2. $G$ gives the clock granularity (resolution of clock readings), and $u \leq G$ the rate

adjustment uncertainty (timing error due to discrete rate adjustment; usually $u = 1/f_{osc}$),

3. $P\rho$ denotes the clock drift during the resynchronization period (determined by the resynchronization period $P$ and the oscillator drift $\rho$; $\mu$s/s-range for TCXOs).

In order to achieve a precision in the 100 ns range, each and every of the factors $\varepsilon$, $G$, $u$ and $P\rho$ must be brought down to the 10 ns range. In Sections 4–6, we will explain how this is accomplished in PSynUTC.

# 3 PSynUTC General Architecture

The PSynUTC prototype system, which is currently being built by our spin-off company *Oregano Systems*, will demonstrate the feasibility of GPS time distribution and time synchronization in Ethernet-based LANs with a worst case synchronization precision in the 100 ns range. PSynUTC consists of a number of PC-104$^+$-based nodes running Linux, which are connected via a switched Fast-Ethernet network. Apart from a custom PCI *network interface card* (NIC) and an external *switch add-on*, PSynUTC solely uses common off-the-shelf (COTS) components for cabling, chipsets, switches, and operating system software.

Figure 1 depicts the functional blocks of a PSynUTC node. Its centerpiece is the *CSP timestamping unit* (see Section 4), which sits in between the *Media-Independent Interface* (MII) connecting a standard COTS Fast Ethernet Media Access Controller and a standard Physical Layer Device. CSPs are timestamped here when some specific byte in the data packet is passing by. Application support, like timestamping of external events and generation of point-in-time events, is also provided by this unit. Local time is maintained by a very high-resolution *Adder-based Clock* (see Section 5), which provides rate and state adjustment capabilities (see Section 6). Clock adjustments are carried out by a small integrated microcontroller, which executes a suitable clock synchronization algorithm. The $\mu$C also handles the *1 pps pulse* + serial interface protocol to the (optional) GPS receiver.

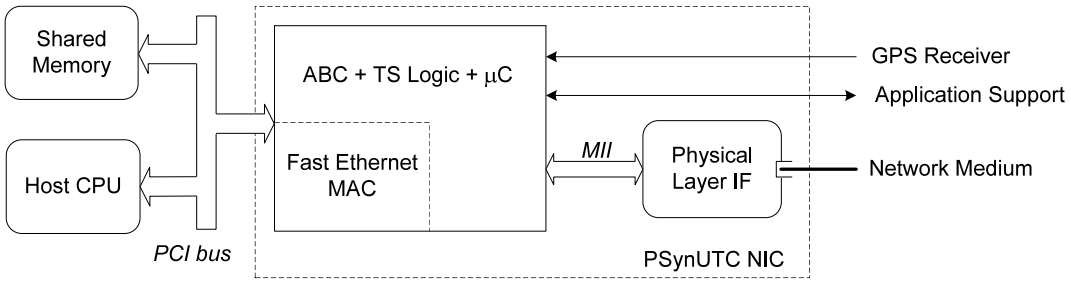


Figure 1: PSynUTC Network Interface Card Architecture

Equipping each node with a NIC that implements the above architecture is sufficient for networks with a shared media. The dominant technology for today's Ethernet networks is micro-segmentation via switches, however. Switches increase the overall bandwidth by learning the "location" of nodes: Packets are only forwarded to the switch port where the particular destination node is attached. This is done based on

the MAC destination address, which is located at the beginning of a packet. Currently, there are two major switching technologies:

- *Store and Forward*, which requires reception of the entire packet before forwarding.
- *Cut Trough*, where the switch tries to start forwarding as soon as it knows the destination address (and hence the outgoing switch port).

Unfortunately, both techniques add unpredictable delays to the end-to-end transmission time of a packet. This is particularly true for store and forward switches, which may increase the transmission delay uncertainty $\varepsilon$ up to seconds. This problem can be alleviated, however, by somehow measuring the time a packet stays on the switch. If this information is inserted into the CSP when it leaves the switch, the receiver node can compute the packet's actual transmission delay. Since known (deterministic) delays do not matter for clock synchronization, the adverse effect of switches upon $\varepsilon$ is effectively removed.

Figure 2 shows how PSynUTC's switch add-on accomplishes this task. Every switch port is equipped with a CSP timestamping unit, which draws timestamps from a common clock that is driven by a reasonably stable oscillator. When a CSP enters the switch, its time of arrival is inserted into a reserved portion of the packet. When a CSP leaves the switch, the difference of the time of departure and the time of arrival in the CSP is computed and inserted into another reserved portion of the CSP. This way, it is even possible to determine the actual transmission delay of CSPs routed over multiple switches.

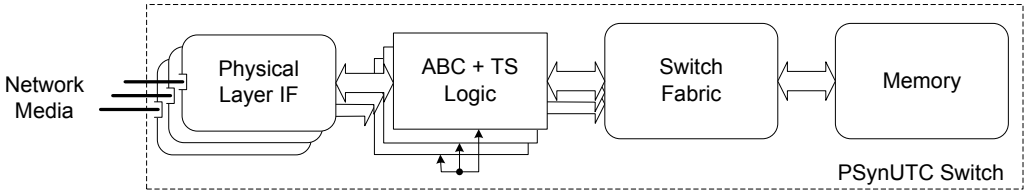


Figure 2: PSynUTC Switch Architecture

All the specific hardware is encapsulated in an ASIC on our network interface card for a PSynUTC client node. It is important to note, however, that this functionality could also be integrated into any Fast Ethernet MAC chip. Similarly, PSynUTC's external switch add-on could easily be incorporated within switches.

# 4 The remote clock reading error

In this section, we will show how PSynUTC achieves a maximum clock reading error in the 10 ns-range. According to equ. (1), this is the first requirement for a synchronization precision in the 100 ns-range. As outlined in Section 2, the clock reading error is essentially equal to the end-to-end transmission delay uncertainty $\varepsilon$, i.e., the variation between the point in time when the transmit timestamp is sampled into an outgoing CSP at the sender and the point in time when a receive timestamp is drawn by the receiver on CSP reception.

### 4.1 Basic principles

In pure software-based clock synchronization, timestamping is done by reading the clock when (1) assembling the CSP for transmission, and (2) when a CSP receive interrupt occurs. However, this implies that channel access delays in half duplex mode, variable delays due to routers (switches, hubs, etc.), interrupt latencies and queuing delays in software+hardware contribute to $\varepsilon$. High-accuracy clock reading is hence impossible here.

In order to reduce $\varepsilon$, timestamping must be performed in hardware and as close as possible to the physical layer. As a first step towards this goal, we developed a *memory-based timestamping* method [HSS98], which transparently inserts timestamps via memory-mapping techniques whenever the network controller reads a packet upon transmission. Similarly, a timestamp is drawn from the receiver's clock when the network controller writes a packet upon reception. Experiments conducted with a suitable evaluation system showed that some $\varepsilon$ in the $10\mu s$-range can be achieved in 10 Mb/s Ethernet-coupled systems. Our in-depth analysis [SKM+00] has shown, however, that network controller FIFOs are an inherent limiting factor for further reducing $\varepsilon$ with memory-based timestamping.

In [SHK99], we proposed an alternative *MII-based timestamping* technique, which is the method of choice for PSynUTC. Timestamps are inserted at the (essentially serial) standardized *Media Independent Interface* between Ethernet media access controllers and physical layer devices here. Our MII timestamping unit continuously monitors the bit stream at the MII data lines and draws a timestamp whenever a packet with a certain type field passes by. This timestamp is then inserted into the bit stream at some reserved position. Finally, the frame checksums are updated accordingly.

### 4.2 Measurement results

In order to quantify the transmission delay uncertainty $\varepsilon$ achieved by MII-based timestamping, we conducted some experiments. The measurement setup is illustrated in Figure 3. It consists of two nodes and a Stanford Research Systems SR620 Frequency Counter with a resolution of 25 *ps* and an ovenized timebase. For a number of COTS Fast Ethernet cards, we measured the $\varepsilon$ induced by the Physical Layer devices and the transmission over the channel. Note that there is no need to consider other sources of uncertainty, since overload conditions, interrupt latencies, bus contention, collisions, etc. are of no concern for MII timestamping. Note that even packet loss does not matter here (although it must be detected by our measurement system to avoid unmatched measurements).

Using a simple traffic generator program based on the link-layer library LibNet[2], we sent CSPs from node $p$ to node $q$. Whenever an outgoing CSP was detected at the MII at node $p$, the frequency counter was triggered for measuring the time it took for the CSP to show up at the MII of node $q$. The measurement result was then transferred via a serial interface to a log file at node $q$. In order to obtain meaningful statistical data, this single measurement was repeated 100.000 times, for different network interface cards, network configurations and interconnections. Table 1 and Figure 4 show a fairly representative set of measurement results. They were conducted with two Allied Telesyn[3] AT-2700TX Fast Ethernet Cards and a 99m long cross-connect cable.

[2] http://www.packetfactory.net
[3] http://www.alliedtelesyn.com/

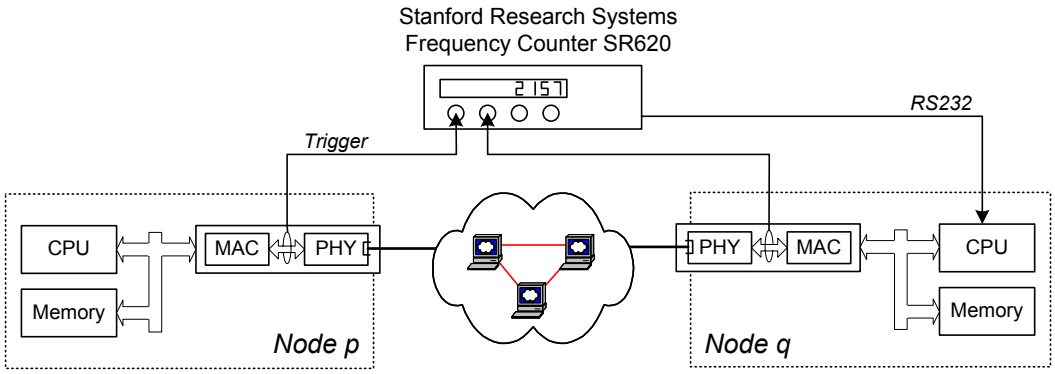


Figure 3: Measurement system to quantify the clock reading error in our PSynUTC architecture

| 99m CAT5 | Sender, Receiver: Allied Telesyn AT2700Tx | | | |
|---|---|---|---|---|
| | Half Duplex | | Full Duplex | |
| | 10BaseT | 100BaseTx | 10BaseT | 100BaseTx |
| Minimum | 8.05859e-06 | 9.02306e-07 | 8.05834e-06 | 8.78326e-07 |
| 95%-minimum | | 9.03549e-07 | | 8.79497e-07 |
| 95%-maximum | | 9.03552e-07 | | 8.795e-07 |
| Maximum | 8.36153e-06 | 9.0478e-07 | 8.36156e-06 | 8.80634e-07 |
| Average | | 9.03551e-07 | | 8.79499e-07 |
| Std. Dev. | | 2.88134e-10 | | 2.77675e-10 |

Table 1: End-to-end delays with 99m cross-connect cable (values given in sec.)

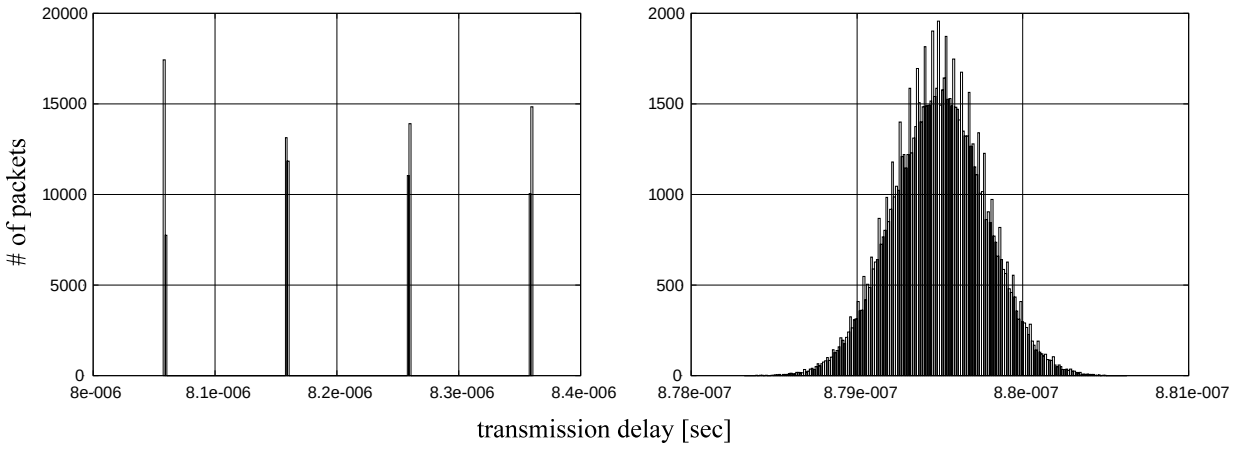


Figure 4: Transmission delay measurement in Full Duplex Mode: 10BaseT (left), 100BaseTx (right)

For 100 Mb/s Ethernet, it is apparent that $\varepsilon \approx 300$ ps only, both in full and half duplex mode. The distribution of the transmission delays is approximately Normal. Hence, the clock reading error can effectively be neglected here.

In sharp contrast, for 10 Mb/s Ethernet, the left hand side of Figure 4 reveals four discrete peaks in the distribution of the transmission delays. Since they are 100 ns apart, they cause some $\varepsilon \approx 303$ ns. Consequently, 100 ns-range clock synchronization precision cannot be guaranteed for 10 Mb/s Ethernet without additional measures. Note that 4–6 such peaks were observed for any network interface card, both in full and half duplex mode. We conjecture that those peaks are due to a synchronization uncertainty of the PLLs used for re-generation of the MII receive clock and the incoming data stream. After all, 10 Mb/s Ethernet employs a 10 MHz system clock frequency, and the receiver's PLL needs to lock anew at the beginning of every frame. Hence, delay jumps in the range of 100 ns could be explained by earlier/later locking of the PLL.

# 5 Clock granularity

In this section, we will show how PSynUTC deals with the problems arising from its discrete local time: Synchronizing delays and finite resolution inevitably introduce a uncertainty $G$ when timestamping events like CSP arrivals or external signals. In addition, a discrete clock can be adjusted only in multiples of some minimal step time $u = 1/f_{osc}$. According to equ. (1), both $G$ and $u$ must be brought down to the 10 ns-range.

## 5.1 Adder-based clock design

PSynUTC utilizes an *adder-based clock* (ABC) [SSHL97], which uses a large high-speed adder instead of a simple counter for summing up the elapsed time between consecutive oscillator ticks. Figure 5 shows its internal structure. The augend value is provided in a STEP-register (with resolution $2^{-64}$ s), which is loaded with $1/f_{osc}$ to achieve the desired rate of progress of the ABC. Speeding up or slowing down the clock can be accomplished by modifying the augend value in the STEP register. Owing to this, PSynUTC's local clock can be paced by any high-frequency source, is fine-grained rate adjustable, and even supports state adjustment via continuous amortization in hardware.

The clock granularity $G \geq 1/f_{osc}$ is determined by the resolution of the timestamps drawn from the local clock. Since the resolution of the time registers in our PSynUTC ASIC is $2^{-64}$ s, $G$ is actually determined by $1/f_{osc}$. Our analysis in [SS97] revealed that the rate adjustment uncertainty $u$ of an adder-based clock is also $1/f_{osc}$. Consequently, our handle for decreasing both $G$ and $u$ is increasing the frequency of the oscillator pacing the ABC. In order to accomplish this, however, the adder-based clock design must be made as fast as possible.

To increase the clock frequency of any integrated digital circuitry the number of logic levels between two storage elements (Flip-Flops) has to be minimized in order to reduce propagation delay through the logic. A commonly used method to accomplish this is to use pipeline stages to split the amount of logic that has to be passed in one clock cycle. With this it is possible to achieve acceptable frequencies, even in FPGA (Field Programmable Gate Arrays) devices, which are approximately a factor of 6–8 slower

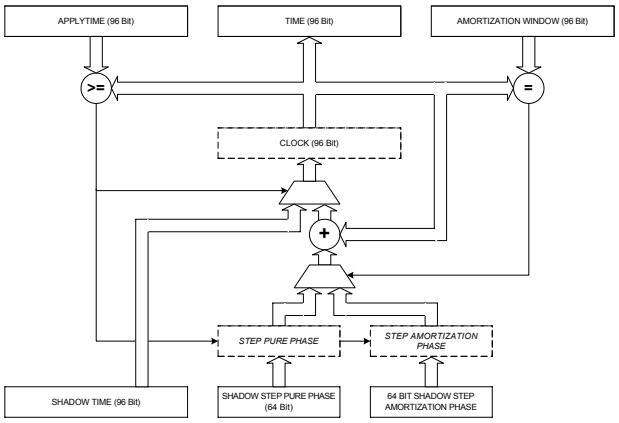


Figure 5: The adder based clock principle as it is used in our PSynUTC ASIC. 64 bit STEP registers hold the augend that increments the clock with every rising edge of the circuits clock signal. Also shown is the possibility to load the contents of the 96 bit CLOCK register.

than comparable ASIC technologies. Table 2 reveals that pipelining is compulsory to achieve clock frequencies in the 100 MHz-range.

| number of pipeline stages | FPGA 1 [MHz] | FPGA 2 [MHz] | CMOS 0.35 [MHz] |
|---|---|---|---|
| 0 | 114 | 82 | 242 |
| 1 | 139 | 126 | 257 |
| 2 | 166 | 174 | 247 |
| 5 | 155 | 181 | 293 |
| 10 | 134 | 192 | 297 |
| 20 | 122 | 210 | 383 |
| 30 | 123 | 216 | 396 |
| 47 | 110 | 245 | 443 |

Table 2: Overview of achievable clock frequencies for a 96 bit adder in several ASIC technologies.

Implementing a pipelined adder comes at a price, however. First of all, latency is added to the final result of the computation, which has to be taken into account by the clock synchronization algorithm. Fortunately, since this latency is fixed, this does not adversely affect the resulting overall clock synchronization precision. Moreover, setting the clock requires a special internal structure to allow all pipeline stages to be loaded with a correct value.

### 5.2 Synchronizer stages

Another issue closely related to the achievable frequency for the adder based clock is to determine the exact point in time when an event like CSP arrival or an external signal arrives. Obviously, there is no problem if the event occurs synchronously with the adder-based clock. For example, timestamping of CSPs upon transmission does not need synchronization, since PSynUTC derives the MII transmit clock from $f_{osc}$ as well. The MII receive clock, however, is recovered from the incoming serial data stream and is therefore not in synchrony with the ABC. Hence, timestamping of arriving CSPs

must be synchronized with the local clock and thus suffers from a variable delay up to $1/f_{osc}$. Fortunately, since this effect is already covered by $G$, there is no additional uncertainty here.

Using a pipelined adder-based clock with high resolution, PSynUTC employs an oscillator frequency $f_{osc}$ in the 100 MHz range. Both $u$ and $G$ can hence be brought down to the 10 ns-range as required.

# 6 The influence of clock drift and stability

In this section, we will show how PSynUTC reduces the drift term $P\rho$ in equ. (1) without an expensive oscillator at every node. Table 3 gives an overview of some oscillators along with their most important characteristics. A standard quartz oscillator cannot be used for our approach, since their $\rho$ is not better than 10 $\mu$s/s, which would wipe out all the efforts to reduce the $\varepsilon$, see Section 4. An OCXO is much better suited in terms of the drift, but the high component cost of more than 200 US$ would make such a NIC too expensive. A TCXO has a reasonable component cost, but a drift of about 1 $\mu$s/s is still not adequate. An atomic oscillator can only be considered as a reference.

| oscillator | $\rho$ [$\mu$s/s] | $\sigma$ [$\mu$s/s$^2$] | cost [US$] |
|---|---|---|---|
| uncompensated crystal oscillator (XO) | 10 | 0.05 | 1 |
| temperature compensated crystal oscillator (TCXO) | 1 | 0.0005 | $10-100$ |
| oven controlled crystal oscillator (OCXO) | 0.1 | 0.0001 | $200-2000$ |
| Rubidium atomic oscillator | 0.0005 | 0.00001 | $2000-8000$ |

Table 3: Oscillator characteristics

In order to provide a cost-efficient solution, PSyncUTC employs a TCXO together with a distributed algorithm that synchronizes the clock rate $v_p(t) = dC_p(t)/dt$ of a node $p$. The requirement on such an algorithm is to have the clock rates adjusted in such a way that the rate differences are bounded such that $\forall t : |v_p(t) - v_q(t)| \leq \gamma$ for any two non-faulty nodes $p, q$. The parameter $\gamma$ given in $\mu$s/s is called the *consonance* of the ensemble. It determines the amount a clock pair drifts apart during a resynchronization period $P$, which is $\gamma P$ instead of the traditional $2\rho P$. The goal is to achieve a $\gamma$ that is much smaller than $\rho$.

## 6.1 System model

Essential for rate synchronization is the separation of the oscillator and the clock. An oscillator is just a device that generates a periodic signal with frequency $f_p(t)$. Its drift rate from the nominal frequency $f_{osc}$ can be expressed with $\rho_p(t) = f_p(t)/f_{osc} - 1$. A clock is a device driven by such an oscillator that maintains a clock value $C_p(t)$ represented by a given number of bits. The coupling in terms of rate between the oscillator and the clock can be described by $v_p(t) = S_p f_p(t)$, where parameter $S_p$ is the *coupling factor*. In a conventional clock design $S_p$ is fixed to $1/f_{osc}$, thus no rate adjustments are possible. In case of the adder-based clock, however, $S_p$ can be explicitly set by the STEP-register, see Section 5.1.

Since $S_p$ needs to be kept constant for some period, rate synchronization can only work if oscillators do not alter their drift rate $\rho_p(t)$ too much during such a period. This behavior is captured by the system assumption that $\forall t_2 \geq t_1 : |\rho_p(t_2) - \rho_p(t_1)| \leq$

$\sigma_p(t_2 - t_1)$. The parameter $\sigma_p$ is the *short-term oscillator stability* given in $\mu s/s^2$ for a node $p$. Representative values of $\sigma$ are summarized in Table 3. In some cases data sheets contain such specifications, otherwise explicit frequency measurements on oscillators need to be carried out.

### 6.2 Clock Rate Algorithm

Having addressed the system aspects of rate synchronization, this section presents the *clock rate algorithm*. It works on the same principles as an interval-based *clock state algorithm*, see Section 2, but instead of an accuracy interval $\boldsymbol{A}_p(t)$ a *rate interval* $\boldsymbol{R}_p(t)$ is maintained by the clock rate algorithm at each node $p$. Formally, a rate interval $\boldsymbol{R}_p(t)$ is defined as correct at $t$ if $1/v_p(t) \in \boldsymbol{R}_p(t)$. They are always pre-computed in such a way that they are correct until their next update.

The clock rate algorithm is based on rounds, which are established by the clock state algorithm. In one such round, each node $p$ performs the following operations:

1: Broadcasts a packet, which contains the rate interval $\boldsymbol{R}_p$, to its peer nodes $q$ in the ensemble. The packet is timestamped with $T_p$ at the moment of sending.

2: Collects for a certain duration the packets from its peer nodes $q$ in the ensemble. Each packet is timestamped with $T_{p,q}$ at the moment of reception.

3: Computes the rate interval $\boldsymbol{R}_{p,q}$ from the received rate interval $\boldsymbol{R}_q$ for the peer nodes $q$ in the ensemble. This is done with the help of the relative rate measurement: Node $p$ knows the sending timestamp $T_q$ and receiving timestamp $T_{p,q}$ of a packet transmission as well as $T'_q$ and $T'_{p,q}$ from an earlier packet transmission. The quotient $(T_q - T'_q)/(T_{p,q} - T'_{p,q})$ gives the required ratio $v_q/v_p$.

4: Computes the new rate interval $\boldsymbol{R}^*_p$ by applying the interval-based *Fault-Tolerant Intersection* (FTI) function to the rate intervals $\boldsymbol{R}_{p,q}$. See [SS01] for the definition and properties of this function.

5: Adjusts the rate of the clock by enforcing the new coupling factor $S^*_p$, which is obtained from the current coupling factor $S_p$ and the new rate interval $\boldsymbol{R}^*_p$.

6: Waits for the next round without changing the coupling factor, thus the clock is free-running apart from state adjustments.

### 6.3 Analytic Results

A rigorous analysis of the above clock rate algorithm can be found in [Sch97b, Sch98] and [SW99]. The major result is the following bound on the consonance:

$$\gamma = 6\sigma R + \frac{4\varepsilon}{R} \tag{2}$$

The parameter $R$ is the resynchronization period of the clock rate algorithm, which should be a multiple of resynchronization period $P$ of the clock state algorithm. It is important to point out that the worst case consonance $\gamma$ of this rate algorithm does not depend on the oscillator drift $\rho$. To give a numerical example, let $\sigma = 0.0005$ $\mu s/s^2$ for a TCXO, $\varepsilon = 400$ ns taken from the measurements in Section 4 and $R = 30$ s, then $\gamma = 0.09$ $\mu s/s$ + $0.053$ $\mu s/s$ = $0.143$ $\mu s/s$ by virtue of formula (2). This is a remarkable result in comparison to the drift of $1$ $\mu s/s$ for clocks driven by a TCXO without any rate resynchronization.

## 6.4 Simulation Results

For a better illustration of the clock rate algorithm, simulations have been conducted with our SimUTC toolkit, see [WGSS99] and [Wei97]. The simulated system consists of $n = 5$ nodes that synchronize the clock state at every $P = 12$ s. Each node $p$ hosts an interval clock $\boldsymbol{C}_p(t)$, where the respective average drifts $\rho_p$ are 0 $\mu$s/s, 0.5 $\mu$s/s, $-0.5$ $\mu$s/s, 2 $\mu$s/s, and $-1$ $\mu$s/s. The stability $\sigma$ for all oscillators is 0.01 $\mu$s/s$^2$. The network has a transmission delay uncertainties $\varepsilon^-_{\max} = 10$ $\mu$s and $\varepsilon^+_{\max} = 20$ $\mu$s.

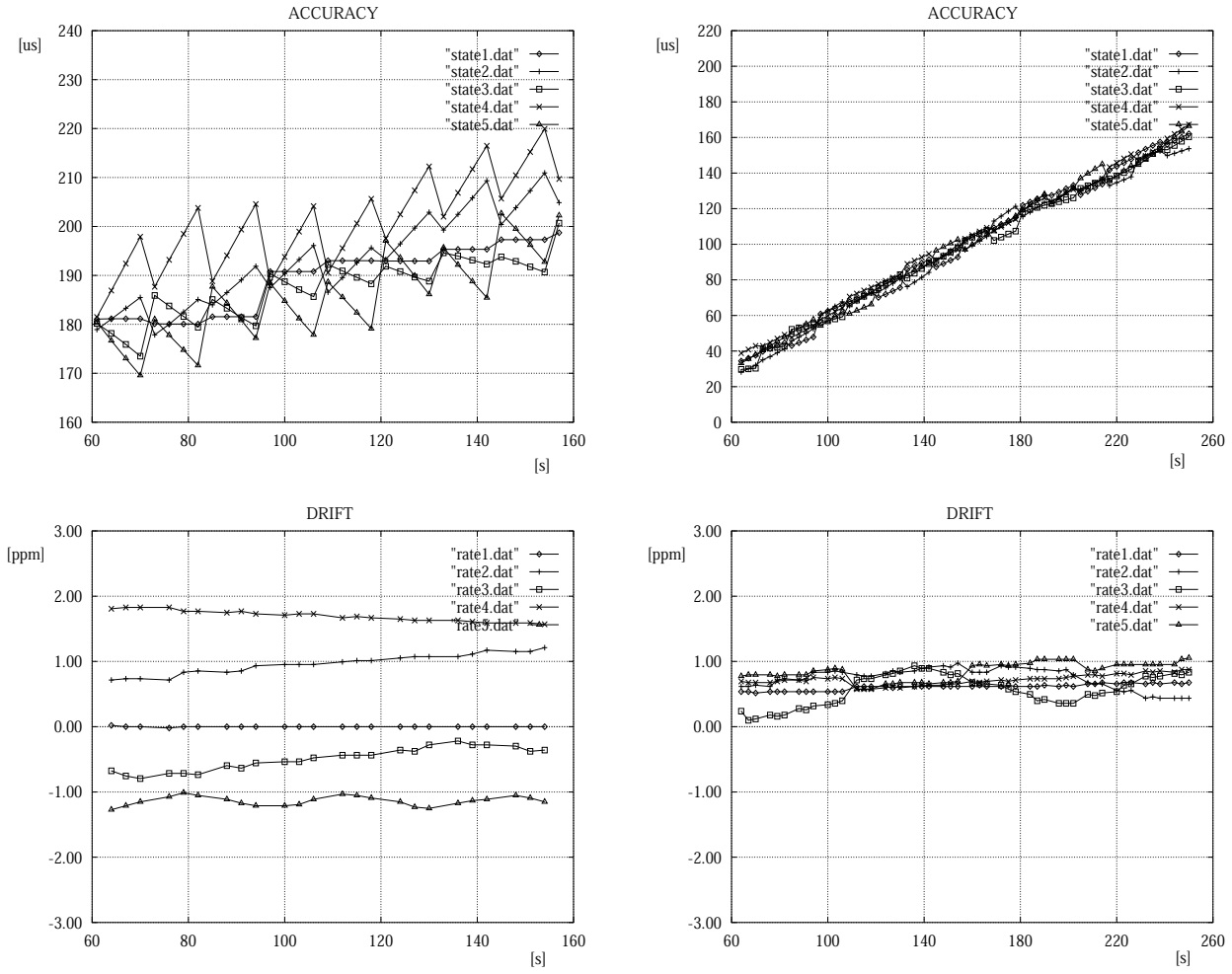


Figure 6: SimUTC simulations of clock synchronization: without rate algorithm (left plots) and with rate algorithm (right plots). synchronization

The upper left plot of Figure 6 depicts the (traditional) accuracy $\alpha_p(t) = C_p(t) - t$ of clocks without rate synchronization. In this simulation the precision is not better than 33 $\mu$s. In the corresponding lower left plot the clock drifts $\rho_p(t)$ are shown, where the consonance $\gamma$ is around 3 $\mu$s/s. The effects of an additional rate synchronization can be observed by the right plots in Figure 6. As depicted by the upper right plot, the previous sawtooth-like shape of the traditional accuracies $\alpha_p(t)$ has almost vanished due to the synchronized clock rates. The lower left plot shows the corresponding clock

drifts, which are now squeezed between 0 and 1.1 $\mu$s/s, reducing the consonance to around 0.7 $\mu$s/s.

## 7 Conclusion

We provided an overview of our PSynUTC prototype system for GPS time distribution in Ethernet networks and showed how it achieves an accuracy in the 100 ns-range: MII-based packet timestamping is used for reducing the clock reading error, a pipelined high-resolution adder-based clock driven by a 100 MHz-range oscillator provides the required adjustment capabilities without disturbing discretization effects, and clock rate synchronization is used for coping with the relatively large drift of TCXO oscillators.

Consequently, PSynUTC provides a number of attractive features: It provides an unrivaled clock synchronization precision and accuracy even in switched Ethernet networks. The achievable precision is in fact comparable to implementations where every node is equipped with a dedicated GPS receiver, or where dedicated cabling for time distribution is at hand. Moreover, PSynUTC is made up of standard COTS hardware and software components only. In particular, every Ethernet Controller and Physical Layer Device (and their device drivers) supporting the standard Media-Independent Interface can be used. The non-standard functionality added is neatly encapsulated in a single custom ASIC and a switch add-on, which could seamlessly be integrated into standard devices.

## References


[FC95] Christof Fetzer and Flaviu Cristian. An optimal internal clock synchronization algorithm. In *Proceedings 10th Annual IEEE Conference on Computer Assurance*, Gaithersburg, MD, June 1995.

[Hor01] Martin Horauer. Hardware support for clock synchronization on distributed systems. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'01)*, pages A–10–A–12, Göteborg, Sweden, July 1–4, 2001.

[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 (WPDRTS'98)*, pages 1067–1076, Orlando, Florida, March 30 – April 3 1998.

[Lis93] Barbara Liskov. Practical uses of synchronized clocks in distributed systems. *Distributed Computing*, 6:211–219, 1993.

[LWL88] Jennifer Lundelius-Welch and Nancy A. Lynch. A new fault-tolerant algorithm for clock synchronization. *Information and Computation*, 77(1):1–36, 1988.

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

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

[Sch97b] 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 21–24, 1997.

[Sch98] Klaus Schossmaier. *Interval-based Clock State and Rate Synchronization*. Dissertation, Vienna University of Technology, Faculty of Technical and Natural Sciences, 1998. (Appeared in Dissertationen der Technischen Universität Wien, Band 87. Österreichischer Kunst- und Kulturverlag, Wien, 2000, ISBN 3-85437-201-9.).

[Sch00] Ulrich Schmid. Orthogonal accuracy clock synchronization. *Chicago Journal of Theoretical Computer Science*, 2000(3):3–77, 2000.

[SHK99] Ulrich Schmid, Martin Horauer, and Nikolaus Kerö. How to distribute GPS-time over COTS-based LANs. In *Proceedings of the 31th IEEE Precise Time and Time Interval Systems and Application Meeting (PTTI'99)*, pages 545–560, Dana Point, California, December 1999.

[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):24–57, January 2000.

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

[SS01] Ulrich Schmid and Klaus Schossmaier. How to reconcile fault-tolerant interval intersection with the Lipschitz condition. *Distributed Computing*, 14(2):101 – 111, May 2001.

[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, October 19–22, 1999.

[Wei97] Bettina Weiss. Simulation environment for clock synchronization. Diplomarbeit, Vienna University of Technology, Department of Automation, June 1997.

[WGSS99] Bettina Weiss, Günther Gridling, Ulrich Schmid, and Klaus Schossmaier. The SimUTC fault-tolerant distributed systems simulation toolkit. In *Proceedings 7th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'99)*, pages 68–75, College Park, MD, USA, October 24–28, 1999.

[YM93] Z. Yang and T. A. Marsland. Annotated bibliography on global states and times in distributed systems. *ACM SIGOPS Operating Systems Review*, pages 55–72, June 1993.