High Accuracy Software-Based Clock Synchronization Over CAN

Sascha Einspieler, Nirmal Rathakrishnan, Arpitha Prabhakara, Benjamin Steinwender, *Member, IEEE,*, and Wilfried Elmenreich, *Senior Member, IEEE*

***Abstract*—In distributed real-time communication systems, common knowledge of the global time is crucial. It prevents message violations on the bus and allows independent components to collaborate within a real-time system on a timely basis. Systems with hard real-time requirements need to have high precision and accuracy of time. This is achieved by hardware-supported frame time-stamping mechanisms as found in dedicated protocols like Time-Triggered CAN (TTCAN), Flexray, and Time-Sensitive Networking (TSN)-enabled Ethernet. However, many microcontroller units are not specifically designed to provide such a hardware-based solution at the communication interface. Therefore, a software-based implementation of the time synchronization algorithm is needed. Nevertheless, some Commercial off-the-shelf (COTS) microcontroller units already provide an IEEE 1588-enabled Ethernet interface, including a high precision timer module with rate correction. This module can be used for time synchronization purposes to align a set of distributed clocks via various communication interfaces.**

This paper investigates the accuracy of software-based and hardware-supported time synchronization algorithm over the Controller Area Network (CAN) protocol using a COTS microcontroller. As a result, we present identified jitter and delay sources as well as the achieved time accuracy. We show that using an advanced timer module combined with additional system knowledge allows sub-microsecond precision and accuracies.

***Index Terms*— Real-Time, Software, Synchronization, Clock, Accuracy, CAN**

This work was funded by the Austrian Research Promotion Agency (FFG, Project No. 884573).

I. Introduction

In time-sensitive applications (e.g., automation), a common time base of multiple communication participants is an indispensable element [1]. The favored synchronization protocol in Ethernet-based industrial communication is the Precision Time Protocol (PTP) [2] which fills the gap between Network Time Protocol (NTP) [3] and Global Positioning System (GPS) timing accuracies. It applies a simple information exchange sequence that achieves an accuracy within the sub-microsecond range. A more accurate PTP extension is the *White Rabbit* protocol which reaches an accuracy in the sub-nanosecond range [4]. This is significantly better than the widely used NTP, which typically achieves results in the range of milliseconds [3]. The improvement is attributed to the Network Interface Card (NIC) hardware extension that deterministically timestamps incoming and outgoing messages directly at the physical port. This omits local time-jitter which is otherwise introduced by interrupts, context switching, and software execution making time reference measurements more accurate.

Besides Ethernet, the CAN protocol provides hardware-based time-stamping through ISO 11898-4 [5] resulting in the TTCAN protocol. Since the hardware implementation is exclusively implemented in the TTCAN transceivers, this feature is not readily available in generic CAN modules. To close this gap, [6] presents a possible CAN controller design where an external time-stamping unit is attached to a CAN controller returning a time-stamp on each capture event. However, not even CAN-FD [7] as a successor of CAN includes this technology. Therefore, available and future CAN controller units still require a software-based clock synchronization approach.

In this work, we want to investigate the achievable accuracy of a software-based and hardware-supported clock synchronization approach using CAN as communication interface and COTS controllers. A rate-adjustable high-resolution timer, exploited from an IEEE 1588-enabled Ethernet interface as a stand-alone module, provides the hardware support. Having this module, various communication protocols may be enabled for time synchronization instead of being limited to a single dedicated interface.

The remainder of this paper is organized as follows: Section II gives an overview of available synchronization techniques showing their benefits and drawbacks. In Section III, we propose our clock synchronization approach, which minimizes communication delays and jitter to improve accuracy. Section IV describes the test setup as well as the applied measurement techniques and tools. Section V presents the relevant delay and jitter sources followed by the improved synchronization result obtained by utilizing the rate correction mechanism. Section VI concludes the paper.

II. Related Work

Over the last decades, different software-based clock synchronization approaches over CAN have been presented. They can be divided into centralized and decentralized techniques. The former technique utilizes at least one dedicated participant that provides its local time as reference time. In contrast, the later technique uses implicit or published information to synchronize the distributed clocks.

As applied in TTP/C [8], a decentralized synchronization approach requires predefined knowledge of the transmission

S. Einspieler and A. Prabhakara are with KAI Kompetenzzentrum Automobil- und Industrie-Elektronik GmbH, Villach, Austria, and also with Networked and Embedded Systems, Alpen-Adria University, Klagenfurt, Austria. E-mail: sascha.einspieler@k-ai.at

B. Steinwender and N. Rathakrishnan are with KAI Kompetenzzentrum Automobil- und Industrie-Elektronik GmbH, Villach, Austria.

W. Elmenreich is with Networked and Embedded Systems, Alpen-Adria University, Klagenfurt, Austria.

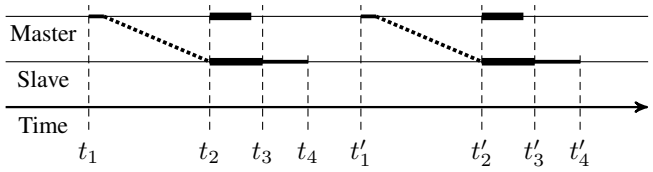


Figure 1: Two synchronization frames send from the master to the slave as presented by [12].

time of messages carrying important information. This technique thus relies on message broadcasts since all participants must receive a transmitted message at the same time to align their local clocks. Here, each node implements the same time correction routine to compare the message arrival times to the expected reception times obtained from the communication schedule. The measured difference is fed into a fault-tolerant averaging algorithm which periodically returns a correction term used to keep the local clock synchronized to the other clocks. This mechanism allows robust clock synchronization without requiring additional control messages. However, the synchronization mechanism depends on active communication. The clocks start to get imprecise as soon as the communication stops, or the communication rounds show enlarged windows where the bus is in an idle state.

Centralized synchronization, as mentioned before, requires at least one dedicated participant that provides its local time as reference time. This leads to two possible synchronization approaches that are named *client-server* and *master-slave*. The client-server approach is driven by the client who requests the server for resynchronization periodically or when it detects an increased time offset. This approach may lead to increased inaccuracies in distributed systems since the synchronization messages need to travel through multiple network components [9]. As found in heterogeneous complex networks, transmission delays also impact the communication's temporal quality and, therefore, also need to be compensated [10]. However, an automated transmission delay compensation mainly depends on the temporal accuracy of the distributed clocks such that the transmission delay of a certain communication path can be resolved.

A popular example for the client-server approach is NTP [11] which synchronizes all distributed clocks to a common time base provided by at least one server. Thereby, the time is not distributed by a server but requested by the clients. The exchanged packets have a fixed structure where both the requesting client and the server fill in the packet's transmission and reception time from their individual perspectives. After the information exchange, the client can now calculate the time difference to the master, including the message round-trip time, to correct its local clock. This approach works very well for Ethernet but is not feasible for CAN since each node needs to be treated individually. Consequently, the more nodes periodically synchronize their time, the less bandwidth remains for other messages.

A promising technique uses a master-slave approach where the master periodically transmits a synchronization frame [12]. As shown in Fig. 1, the master starts the frame transmission at time t_1, which finishes at time t_2. At t_2, all participants take a snapshot of their local time triggered by the master's transmission and the slave's reception event. Each timestamp taken by the master is used to be transmitted in the subsequent synchronization frame. This procedure also applies to the following frames. The slave's correction algorithm is performed between t'_3 and t'_4 where it adjusts its local time using the local and the master's timestamp from the previous synchronization round. The authors claim that their algorithm needs less than 20 messages per second to achieve an accuracy of $20\,\mu\mathrm{s}$ at a transmission speed of $1\,\mathrm{Mbit/s}$. Recent work of Akpinar *et al.* [13] improved this approach by $60\,\%$ by also taking the oscillator's clock drift into account.

Another interesting approach presented by Rodriguez-Navas *et al.* [14] also follows a master-slave scheme. To mitigate the effect of master-based clock drifts, they minimize the time between fetching and transmitting a timestamp. This is achieved by taking a timestamp on the resynchronization event of a frame's start-of-frame bit and adding it as a payload to the currently ongoing frame transmission. Since the timestamp is bound to the frame's starting point, it eliminates the risk of erroneous synchronization caused by missing synchronization messages and thus adds fault-tolerant behavior. Although the authors described a hardware-based solution, their approach can also be realized in software. However, this is only possible if the CAN controller provides a start-of-frame event and allows data modification while the currently transmitting frame is in the arbitration phase.

The available solutions solely correct the clock's offset but do not modify the clock's tick rate. As a drawback, the achievable accuracy depends on the system's worst clock drift rate. This impacts the synchronization quality and results in a discontinuous time which is a problem when a slave's clock ticks faster than the master's clock. From the slave's point of view, it jumps back in time when its clock offset is corrected. Consequently, two events may be assigned to the same local timestamp, although their global event time is different. This situation must not be allowed in a real-time system. Another interesting finding showed that the majority of these approaches do not investigate the communication path. Communication jitter and delays are mentioned, but neither their sources nor possible countermeasures are proposed. For instance, none of these works investigate the impact of the applied CAN transceivers or the microcontroller's interrupt behavior that also introduces delays and jitter.

With the rise of IEEE 1588, also known as PTP, various works were published investigating the limitations of this synchronization protocol or leveraging the IEEE 1588 timer to perform a partially hardware-assisted time-synchronization. The work of Ferencz *et al.* [15], for example, describes an Ethernet-based network where the nodes consist of COTS components. The node hardware thereby runs a standard, non-real-time version of Linux and special software implementations written by them. They showed that they could achieve a sub-microsecond accuracy where the measured time offset mean value is $\mu = 5.30\,\mathrm{ns}$ having a standard deviation $\sigma = 40.64\,\mathrm{ns}$. Another work written by Loschmidt *et al.* [16] presents the synchronization limits of IEEE 1588. The authors address the

main factors for jitter that negatively influence the accuracy of purely hardware-supported time synchronization over Ethernet. They claim that the most relevant jitter source is the local oscillator which frequency jitter must therefore be as low as possible. In [17], the authors extended a LAN eXtensions for Instrumentation (LXI)-based measurement system using PTP and showed significant improvements over previously applied synchronization technologies. They presented time-offset results using COTS and IEEE 1588-enabled network components and claimed that hardware timestamping is a key feature to achieve the most accurate synchronization results.

Over the last decades, great effort has been put into the topic of clock synchronization techniques. Purely hardware-based synchronization techniques result in very accurate time synchronization. Nowadays, this kind of technology is widely available in many Ethernet-enabled embedded devices. However, other popular embedded communication interfaces like CAN do not provide such a feature. To our best knowledge there is no work available which investigates the precision of a software-based and hardware-supported time-synchronization over CAN. This lack of information encouraged us to investigate relevant communication path components and synchronization mechanisms which outcome is used to propose a high-precision clock synchronization approach over CAN. Its performance is first demonstrated using a simple offset correction mechanism and extended by an additional rate correction method where we exploited an adjustable high-precision timer module.

III. Proposed Approach and Requirements

Many COTS microcontroller units with a built-in CAN periphery unit provide event sources indicating the successful transmission and reception of a CAN frame. Therefore, our work applies a similar approach as described by Gergeleit *et al.* in [12], where specific communication events are used to capture timestamps on every node. This approach eases the measurement of hardware and software delays which are necessary to correct the captured timestamps.

A. Exploiting Adjustable High-Precision Timer Modules

An increasing amount of COTS microcontroller units provide an IEEE 1588-enabled Ethernet interface. As a feature, this module includes a dedicated high-precision real-time timer that provides a clock rate correction mechanism. This mechanism guarantees a continuous time representation which is not easily achieved with a purely offset corrected approach. However, only when the timer module is accessible via software its rate correction capabilities can be exploited. This feature massively increases the distributed clocks' synchronization quality, which is proven by the results of this paper.

B. Communication Delays and Event Jitter

Microcontroller units usually do not output the proper voltage levels of a CAN interface. Thus, to connect a controller to a CAN bus, an additional CAN transceiver is required. Consequently, a CAN frame must be routed through the sender's transceiver to the bus and from there through the receiver's transceiver to the receiver. Compared to the delays introduced by the microcontroller, we expect a rather low impact from the transceivers. However, also small delays directly influence the synchronization quality, which requires them to be investigated.

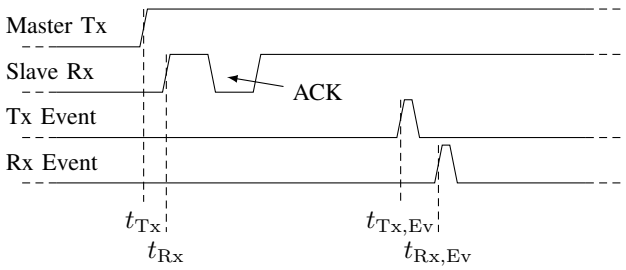


Figure 2: Investigation of distinct points in time to gather information about the communication path delays and the interrupt mechanisms.

We assume that static delays are not an issue for the quality of synchronization. They need to be known to compensate for their effect using an additional correction term. Unfortunately, this is not the case for jitter since we cannot fully compensate its influence. To mitigate this effect, we need detailed information about the relative time when the slaves and the master take their timestamps. Thus, we need to investigate specific points in time, as shown in Fig. 2, to determine the delays introduced by internal event generation mechanisms. t_{Tx} represents the end of the transmitting message, whereas t_{Rx} is the endpoint of the corresponding message at the receiving node. The starting point of the transmission-related or reception-related interrupt service routine is represented by $t_{\mathrm{Tx,Ev}}$ or $t_{\mathrm{Rx,Ev}}$.

The transceiver's impact is obtained by calculating its communication delay with $\Delta t_{\mathrm{trans}} = t_{\mathrm{Rx}} - t_{\mathrm{Tx}}$. Since the transmission and the reception interrupt may follow different mechanisms, we also need to investigate their behavior. The delay of the master's transmission event is obtained with $\Delta t_{\mathrm{Tx,Ev}} = t_{\mathrm{Tx,Ev}} - t_{\mathrm{Tx}}$, which gives information about the delay until the interrupt service routine is executed. Finally, the offset error between the slave's and the master's event is calculated with $\Delta t_{\mathrm{Ev}} = t_{\mathrm{Rx,Ev}} - t_{\mathrm{Tx,Ev}}$.

C. Synchronization Frames

In [12] and [13], two consecutive synchronization frames can be separated by numerous general-purpose messages. This behavior may influence the overall precision since the distributed clocks may drift apart while no synchronization frame is received. Therefore, our approach transmits two closely spaced messages per synchronization round which is favorable since we minimize the influence of clock drifts. As shown in Fig. 3, the master starts a new synchronization round by transmitting a *SYNC* message at t_1. When the message is transmitted at t_2, the master's transmission event trigger captures its current local timestamp t_{m}. Nearly simultaneously also the reception event of each slave triggers that causes them to capture their local timestamp t_{sync}. It was observed that the master's transmission event triggers slightly before the slave's reception event. Internal mechanisms of the CAN

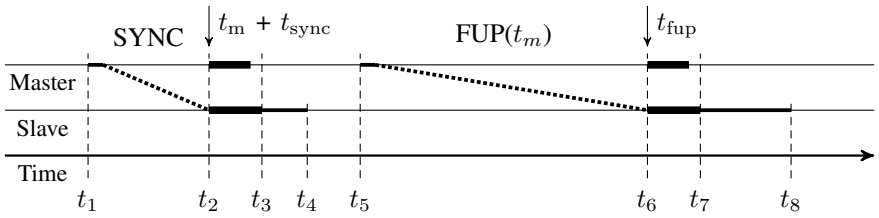


Figure 3: Two synchronization frames send from the master to the slave as presented by [12].

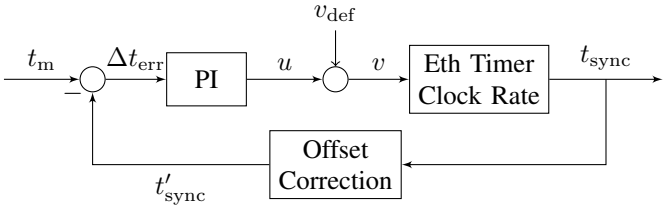


Figure 4: Control loop which is used to adjust the clock rate based on the determined time error.

module may cause this behavior. Thus, between t_3 and t_4, the agents compensate the observed average time error to the master by calculating $t'_\mathrm{sync} = t_\mathrm{sync} - \overline{\Delta t_\mathrm{Ev}}$, where $\overline{\Delta t_\mathrm{Ev}}$ is the average time error between the master and the slave event trigger. In the meantime, the master distributes t_m via the follow-up message labeled as *FUP* and starts its transmission as soon as possible at t_5. When the *FUP* message is received at t_6, the slaves take another timestamp t_fup before they extract t_m from the message. Using the collected information, the agents can now invoke the correction mechanisms at t_7, which terminates at t_8.

We are aware that this concept is vulnerable to partially missing synchronization messages. To avoid time corrections based on incomplete information, we propose some requirements that add fault tolerance to the synchronization procedure. Only if the following requirements are fulfilled, the correction procedures are allowed to be invoked:

- A *SYNC* message must be followed by a *FUP* message.
- The time interval between these two messages must be within a predefined time window.
- Provide multiple time-masters to avoid a single point of failure as described in [14].

D. Clock Rate Correction

The low synchronization accuracy of available software-based solutions is owed to the different clock rates within the system. Consequently, the available accuracy is a function of the maximum clock rate error and the frequency of synchronization rounds. The bigger the error, the more synchronization rounds must be run to keep the quality of synchronization. Thus, a clock rate correction mechanism paired with an initial offset compensation technique is the key to achieve a high synchronization accuracy.

As shown in Fig. 4, the control loop requires t_m and t'_sync to determine the current offset error. Based on this error, the control loop adjusts the Ethernet timer's clock rate by modifying the timer's *addend* register to minimize the clock error. The *addend* register is initialized with a default value as soon as the node is powered on. As mentioned before, timestamp t_m and t_{sync} are captured after the successful transmission of the *SYNC* message. Afterward, each slave performs an offset correction of t_sync based on which we obtain t'_sync. When the slave receives the *FUP* message, it captures t_fup and extracts the master's timestamp t_m from the payload. Following, it calculates the current offset error $\Delta t_\mathrm{err} = t'_\mathrm{sync} - t_\mathrm{m}$, which outcome is forwarded to the control loop. The obtained control value u is added to v_def, which equals the clock's default timer *addend* value. The resulting value named v is finally loaded to the timer's *addend* register that adjusts the timer's clock rate.

E. Clock Offset Correction

When a slave is powered on, its time offset to the master could be already too big, leading to an unstable behavior of the control loop. Thus, we need to define an upper boundary Δt_max which indicates the maximum allowed time offset. If $|\Delta t_\mathrm{err}| > \Delta t_\mathrm{max}$, an offset correction needs to be applied to align the slave's clock to the master's time.

As a first step, the time which elapsed during the transmission of the *FUP* message needs to be compensated. This is achieved by calculating the offset between the slave's timestamps with $\Delta t_\mathrm{diff} = t_\mathrm{fup} - t_\mathrm{sync}$. In contrast to the rate correction mechanism, we do not need to correct the timestamps since the hardware-based delay affects them. Consequently, only its jitter influences the calculated offset. Besides the hardware-based time errors, the software-based time errors also contribute to the total time error and thus need to be taken into account. The software-based time error named Δt_sw occurs at the slave's side between the reception of the *FUP* message and the update of the local clock's time register. During that time, the node extracts t_m from the *FUP* message upon which it performs the time correction operations shown in Eq. (1). In our experiment, Δt_sw was determined via measurements which results are presented in Table I. With this information, we can now derive the master's time

$$t'_\mathrm{m} = t_\mathrm{m} + \Delta t_\mathrm{diff} + \Delta t_\mathrm{sw} \tag{1}$$

at the point in time when the slave updates its local clock time.

IV. System and Measurement Setup

The measurement setup is used to extract communication-related time information and to monitor the distributed system's clock accuracy. The basis of our communication nodes is

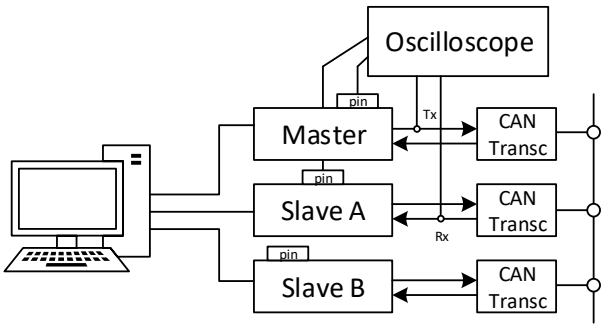


Figure 5: Hardware architecture consisting of a host PC, an oscilloscope, and three controller nodes.

the XMC4700 microcontroller [18] that contains an ARM Cortex-M4 core running at $144\,\mathrm{MHz}$. The utilized $12\,\mathrm{MHz}$ crystal oscillator [19] is not temperature compensated leading to different drift rates as shown in our results. The controller's Ethernet interface features a standalone 64-bit timer used for the IEEE 1588-based time synchronization. The lower double word represents the sub-second part having a $1\,\mathrm{ns}$ resolution, whilst the upper double word represents seconds. As a feature, this timer can be accessed and controlled via software which enables us to exploit the timer's offset and skew correction capabilities.

The communication architecture presented in Fig. 5 applies three nodes where one node is selected to be the time-master whilst the remaining nodes are the time-slaves. To distinguish between the obtained results, the slaves are denoted as slave *A* and slave *B*. Each node's communication lines are routed via a CAN transceiver [20] to the bus. The applied cable length between the master and the slaves is $50\,\mathrm{cm}$. The default synchronization round's period time is set to be one second. It is important to note that within the context of this paper, the temporal precision of a short-distance CAN bus is investigated. Thus, all assumptions and the depicted results neglect the frame propagation delay. Nevertheless, with an increasing bus length, the propagation delay may become relevant, which theoretical impact is discussed in Section V.

The CAN frame transmission and reception delays are acquired with the help of an oscilloscope [21]. Here, the master's transmission line and the reception line of slave *A* are monitored. To gather information about the transmission respectively reception event delay, the state of a dedicated pin is toggled as soon as the nodes enter their interrupt service routines. Since all slaves run the same firmware, we assume that they behave the same except for possibly different clock rates. Thus, the required information was gathered by monitoring a single slave's communication.

To analyze the accuracy of the distributed clocks, the slaves periodically transmit a set of timestamps to a monitoring host computer using their Ethernet interface. This set includes the received master's timestamp t_m and the corrected timestamp t'_sync. The transmission happens during the *FUP* frame's event execution and, thus, once per synchronization round.

A. Service Routine Execution Jitter

Interrupt service routines do not execute immediately when an event is detected. Depending on the currently executed instruction, the interrupt execution can be delayed for several clock cycles. In our case, the controller's execution delay ranges from one up to 12 clock cycles [18]. The maximum possible time error between the master and a slave occurs when, for example, the master's interrupt service routine triggers after one clock cycle while the slave's routine is delayed by 12 clock cycles. Although both nodes may have registered the event simultaneously, this scenario represents the worst case since it leads to the maximum time error, which is approximately $83\,\mathrm{ns}$.

While this execution jitter impacts both correction mechanisms, we assume that the accuracy of the rate correction mechanism mainly suffers from that jitter. Two setups were applied to investigate the cause of varying interrupt service routine execution delays. The first setup solely runs the Ethernet message handler within the main loop, while the second setup is extended by a bubble-sort algorithm that processes randomly generated data. The added lines of code change the pool of available instructions, whereas the additional execution paths influence how often specific instructions are executed. Based on this, we investigated the impact of code changes on the interrupt service routines' execution delay. The resulting distributions were obtained by running 10×10^3 synchronization rounds.

B. Low Jitter Software Execution

The quality of software-based clock synchronization also depends on the execution time jitter of crucial functions like interrupt service routines. Here, the sequence of function calls as well as the function's execution path is relevant. Since timestamps are time-dependent information, their fetching should be preferred over other routine executions. This is important since prior code execution would add a delay that influences the captured timestamp's result.

At the slaves, the execution of the offset-correction function during the *SYNC*'s interrupt service routine is not critical since it neither modifies the local clock's value nor its rate. Thus, this implementation is not required to follow specific optimization rules. This does not apply for the functions which are called within the *FUP* interrupt service routine before the clock's time register is updated. Here, the required code execution time directly influences the accuracy of the offset correction mechanism. Consequently, a constant execution time is desired to mitigate its effect by minimizing the execution time jitter, achieved by avoiding execution branches [22].

C. Software Execution Time Measurement

It is important to know the execution times of specific code segments within the interrupt service routines to correct the fetched timestamps accordingly Both the *SYNC* and the *FUP* interrupt service routine first call a function that fetches a timestamp. This is already the crucial code execution within the *SYNC* routine that is solely important for the rate correction mechanism. However, the *FUP* routine requires additional

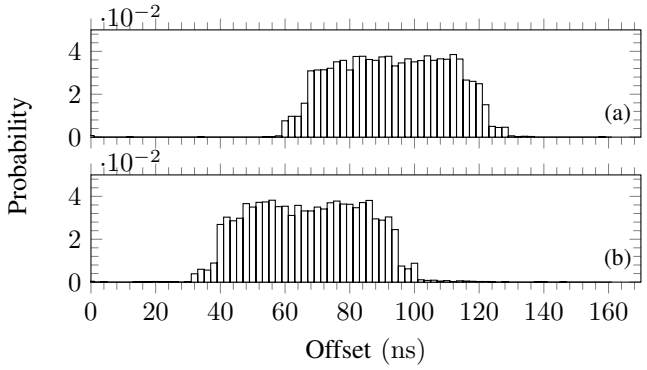


Figure 6: Influence of code changes on the trigger offset between the master and a slave. Example (a) shows the offset by solely executing the Ethernet message handler while example (b) additionally executes a bubble-sort algorithm.

information to compensate for the elapsed time until the offset correction is executed.

We applied two measurement techniques to obtain the required information: Technique (1) utilizes an output pin as described in Section III-B and monitors its state change using the oscilloscope. To determine the code execution time, we now also toggle the pin state when the interrupt service routine reaches its end. The software-based delay is now determined by measuring the difference between the two toggle events. Technique (2) utilizes the exploited timer where we take timestamps instead. The obtained time offset is gathered by transmitting it via the Ethernet interface to the host. Both results are presented in Table I.

V. Results

At the very beginning, we measured the timings as described in Section III-B. Our findings showed that the applied CAN transceiver adds a constant delay of $143\,\mathrm{ns}$ to the communication path. It exhibits a minimal jitter by having a standard deviation of $\sigma = 166\,\mathrm{ps}$ and is thus classified as a non-critical path component. However, since this offset applies to all participants (also for the master's reception path) it is unnecessary to use it as an error correction factor.

Following, we observed the event trigger offset Δt_{Ev}, which results are shown in Fig. 6. The distribution shown in Fig. 6a corresponds to the execution of the main loop that solely executes the Ethernet message handler. In Fig. 6b, the observed node additionally executes a bubble-sort algorithm which works on randomly generated data. The outcome shows that this small change already influences the service routine's invocation where the additional execution paths in (b), on average, require fewer clock cycles. The following outcomes have been obtained based on setup (a). Therefore, the event offset correction value has been set to $\Delta t_{\mathrm{Ev}} = 94\,\mathrm{ns}$ which corresponds to (a)'s mean value.

Finally, we measured the software-related delays, which results are shown in Table I. The first row shows the results obtained by measuring the state changes of a pin using an oscilloscope. In the second row, we present the results using timestamp measurements instead. The two approaches slightly

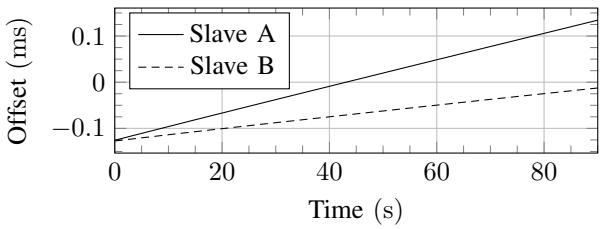


Figure 7: Time offset error of an uncontrolled system.

deviate since their number of executed instructions and thus their execution time differ. Since the implementation of the pin-based approach is simpler, its results are assumed to be more precise. Thus, the FUP result has been used to compensate for the software-related offset error. By taking Δt_{Ev} into account, we thus obtain two temporal correction values. The first value is the SYNC-related correction value $\Delta t_{\mathrm{SYNC,corr}} = \Delta t_{\mathrm{Ev}}$ which is crucial for the rate-correction technique. The second value is solely important for the offset-correction technique and is based on Eq. (1), where $\Delta t_{\mathrm{FUP,sw}} = 7.65\,\mu\mathrm{s}$.

As we gathered all relevant information, we observed the uncontrolled system behavior by measuring the time difference between the slaves and the master. Except for an initial clock alignment, no additional correction technique is applied. Thereby, all slaves initialize their time within the *fup_offset_irq()* routine using the raw timestamp which they received from the master. Thus, delays like the message transmission time, interrupt execution delays, and software execution times contribute to the offset, as shown in Fig. 7 at time $t = 0$. Both slaves show approximately the same time offset to the master, which is $t_{\mathrm{off}} \approx 125\,\mu\mathrm{s}$. The major portion of the delay is attributed to the transmission of the fully-loaded standard CAN frame.

Although both slaves are very precise at $t = 0$, their various clock rates lead them to drift away with the progression of time immediately. Based on this measurement, the observed time error for slave *A* is $2.86\,\mu\mathrm{s}$, while slave *B*'s time error is $1.54\,\mu\mathrm{s}$. Assuming that the master's system clock is perfectly ticking with $144\,\mathrm{MHz}$, the determined frequency error for slave *A* is $f_{\mathrm{err}} \approx 412\,\mathrm{Hz}$, while for slave *B* it is $f_{\mathrm{err}} \approx 222\,\mathrm{Hz}$. However, in general, a node's clock rate heavily depends on the stability of the applied crystal oscillator and thus may drastically change

Measurement	μ	σ
Pin-Toggle	$2.31\,\mu\mathrm{s}$	$19.76\,\mathrm{ns}$
Timestamp	$2.92\,\mu\mathrm{s}$	$15.44\,\mathrm{ns}$

(a) SYNC interrupt service routine.

Measurement	μ	σ
Pin-Toggle	$7.65\,\mu\mathrm{s}$	$27.52\,\mathrm{ns}$
Timestamp	$8.93\,\mu\mathrm{s}$	$63.87\,\mathrm{ns}$

(b) FUP interrupt service routine.

Table I: Software execution times depicting average measured time and standard deviation using different measurement approaches.

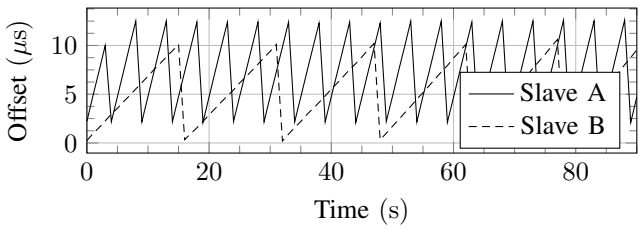


Figure 8: Observed synchronization frequencies when exclusively running the offset correction technique.

Sync Rate (Hz)	μ (ns)	σ (ns)
1	1621.25	108.16
4	517.42	73.48
32	180.29	40.83

(a) Slave A

Sync Rate (Hz)	μ (ns)	σ (ns)
1	2036.26	103.81
4	611.23	71.11
32	191.18	39.93

(b) Slave B

Table II: Offset error's mean value and standard deviation for different synchronization rates.

with a changing environmental temperature.

In the following step, we applied $\Delta t_{\mathrm{FUP,corr}} = \Delta t_{\mathrm{diff}} + \Delta t_{\mathrm{FUP,sw}}$ to compensate for the time delays. In addition, we introduced an upper time deviation boundary $\Delta t_{\mathrm{err,max}} = 10\,\mu\mathrm{s}$ beyond which the offset correction mechanism is triggered. As shown in Fig. 8, we obtained saw-tooth-shaped curves that are distinctive for this technique. Slave *B* requires a higher offset correction frequency than slave *A* to keep the same synchronization quality which origins from slave *B*'s increased clock rate error. The first saw-tooth of slave *A* violated the upper boundary earlier, leading to an earlier offset correction. Its zero-line offset error is a measurement artifact since the obtained timestamp is taken before the offset error is corrected. Thus, the observed error is the clock drift that happened between two synchronization rounds. Without a clock rate correction feature, this behavior mainly impacts the synchronization quality.

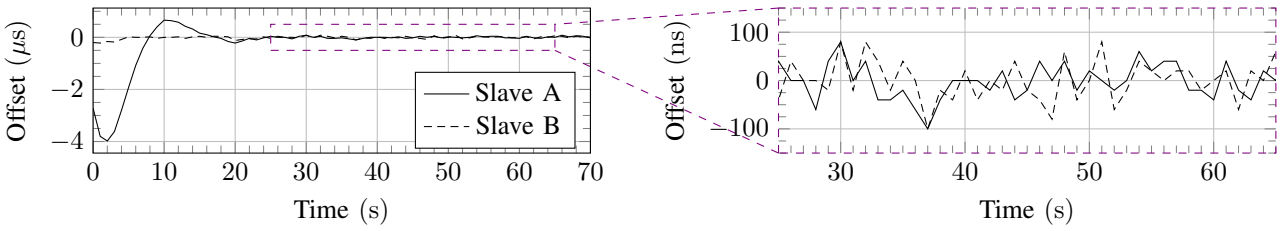
The only possibility to improve the accuracy of an offset correction is to increase the synchronization rate. The master's synchronization rate was set to three different values to prove this, as shown in Table II. As a first outcome, both slaves show a maximum time offset in the range of two microseconds. As it is shown, the slaves become more accurate and more precise with an increasing synchronization rate. However, as a drawback, the additional synchronization messages shrink the available bandwidth.

Finally, we enabled the rate correction mechanism using the settings described in Section III-D. The outcome of this correction is seen in Fig. 9a, where both clock rates are corrected after 24 synchronization rounds. Since slave *A* has an increased clock rate error, its transient behavior is more distinct than slave *B*.

In Fig. 9b, the steady-state offset errors of both slaves are shown. Within this short time frame, the observed uncertainty stays within $\pm 100\,\mathrm{ns}$. To better understand of the system's accuracy over time, we observed the offset error for 11×10^3 synchronization rounds which corresponds to about 3 hours run-time. The obtained accuracy distributions are presented in Fig. 10, where both slaves have nearly the same mean value and standard deviation. These results confirm that the rate-correction technique guarantees a constant accuracy even over a longer time period. By comparing this outcome with the values of Table II, our approach outperforms the offset-correction technique in terms of accuracy, synchronization quality, and the number of required synchronization messages. This is also true compared to a purely software-based synchronization solution, as Gergeleit *et al.* [12] presented, achieving an accuracy of $20\,\mu\mathrm{s}$. At the same time, it requires less than 20 messages per second. Although Akpinar *et al.* [13] improved this software synchronization approach by $60\,\%$, it is still less accurate than our approach. A comparable result is achieved by Ferencz *et al.* [15], who managed to measure an average accuracy of $\mu = 5.30\,\mathrm{ns}$ while having a standard deviation of $\sigma = 40.64\,\mathrm{ns}$ using a Linux-based system where nodes are synchronized via Ethernet.

As mentioned in Section IV, these results were obtained by investigating a short-distance bus. Thus, the question arises of how increased bus lengths impact the accuracy of this synchronization approach. We assume that the propagation delay, as a function of the bus length and the transmission medium between the sending and the receiving node, adds a constant time error to the system. Thus, increased bus lengths may have only minimal influence on the overall clock synchronization since they can be easily compensated if known. In such a system, the propagation delays can be determined via two approaches: Approach (1) applies offline measurements where the individual distances between these slaves and the master are determined. With that knowledge, it is now possible to configure each slave using its specific path correction value. We assume that this approach can compensate for the propagation delay very well. However, its downside is an increased measurement effort that needs to be repeated as soon as the bus configuration changes. Approach (2) uses an automated path-correction approach where the master performs propagation delay measurements with each slave when the bus is initialized. Since the entire propagation delay measurement is automated, it requires no additional effort even when the bus is changed. Nevertheless, the quality of the determined correction value may decrease with a decreasing bus length since the system might be unable to resolve the actual propagation delay.

The maximum bus length for CAN at $1\,\mathrm{Mbit/s}$ is specified as $40\,\mathrm{m}$. Assuming a propagation delay of $5\,\mathrm{ns/m}$ over a twisted-pair cable [23], this bus length introduces a constant delay of $200\,\mathrm{ns}$ from the sending node's transceiver to the receiving node's transceiver. With the achieved accuracy, it would thus be possible to distinguish between the short and the maximum allowed path distance. However, based on our results, this approach seems to be very imprecise for the applied setup since path lengths below $20\,\mathrm{m}$ result in propagation delays in the system's precision noise range.



(a) Settling procedure and steady state.

(b) Zoomed steady state.

Figure 9: Time offset error to the master with offset and rate correction applied.

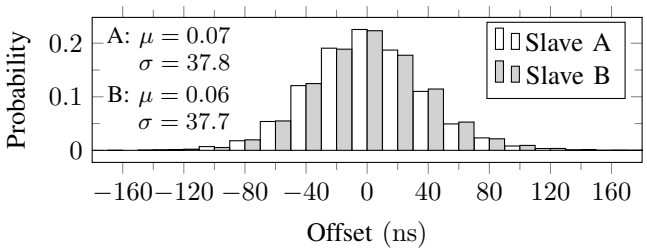


Figure 10: Offset error to the master node using the (a) mid-offset and the (b) individual offset.

VI. Conclusion

We presented a scalable CAN-based clock synchronization technique where we exploited a high-precision timer part of a microcontroller's IEEE 1588-enabled Ethernet interface. We showed that the timer's rate-correction capability drastically improves the clock synchronization as well as its quality. The influence of clock drifts is minimized by transmitting two messages closely together. The first message indicates a new synchronization round, while the second message contains the previously captured master timestamp. Due to the minimized clock rate error, the number of synchronization rounds can also be decreased. This consequently increases the available bandwidth used by other frames. The achievable accuracy lies in the sub-microsecond range, which outperforms purely offset-based synchronization approaches.

References

[1] H. Kopetz, "Global Time," in *Real-Time Systems*, 2nd ed., Springer, Boston, MA, 2011, pp. 51–78.

[2] S. T. Watt, S. Achanta, H. Abubakari, E. Sagen, Z. Korkmaz, and H. Ahmed, "Understanding and applying precision time protocol," in *2015 Saudi Arabia Smart Grid (SASG)*, IEEE, Dec. 2015.

[3] P. Ruiqing, H. Peng, Y. Wenxue, G. Min, and Z. Bin, "The optimization techniques for time synchronization based on NTP," in *2010 2nd International Conference on Future Computer and Communication*, IEEE, vol. 2, 2010, pp. V2–296.

[4] M. Lipinski, T. Wlostowski, J. Serrano, and P. Alvarez, "White Rabbit: A PTP application for robust sub-nanosecond synchronization," in *2011 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, IEEE, 2011, pp. 25–30.

[5] F. Hartwich, B. Müller, T. Führer, R. Hugel, *et al.*, "Timing in the TTCAN network," in *Proceedings of the 8th International CAN Conference (iCC'02)*, 2002.

[6] F. Hartwich, "CAN frame time-stamping - supporting AUTOSAR time base synchronization," in *CAN in Automation*, 2017.

[7] ——, "CAN with flexible Data-Rate," in *Internation CAN Conference*, 2012.

[8] W. Elmenreich and R. Ipp, "Introduction to TTP/C and TTP/A," in *Workshop on Time-Triggered and Real-Time Communication*, Dec. 2003.

[9] D. Jayasimha, S. S. Iyengar, and R. L. Kashyap, "Information integration and synchronization in distributed sensor networks," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 21, no. 5, pp. 1032–1043, 1991.

[10] Z. Wu, Y. Wu, Z.-G. Wu, and J. Lu, "Event-based synchronization of heterogeneous complex networks subject to transmission delays," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 48, no. 12, pp. 2126–2134, 2017.

[11] D. L. Mills, "Internet time synchronization: The network time protocol," *IEEE Transactions on Communications*, vol. 39, no. 10, pp. 1482–1493, 1991.

[12] M. Gergeleit and H. Streich, "Implementing a distributed high-resolution real-time clock using the CAN-bus," in *Proceedings of the 1st International CAN Conference*, vol. 94, 1994.

[13] M. Akpinar, K. W. Schmidt, and E. G. Schmidt, "Improved clock synchronization algorithms for the controller area network (CAN)," in *28th International Conference on Computer Communication and Networks (ICCCN)*, IEEE, 2019.

[14] G. Rodriguez-Navas, S. Roca, and J. Proenza, "Orthogonal, fault-tolerant, and high-precision clock synchronization for the controller area network," *IEEE Transactions on Industrial Informatics*, vol. 4, no. 2, pp. 92–101, 2008.

[15] B. Ferencz and T. Kovácsházy, "Hardware assisted COTS IEEE 1588 solution for x86 Linux and its performance evaluation," in *2013 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication (ISPCS) Proceedings*, IEEE, 2013, pp. 47–52.

[16] P. Loschmidt, R. Exel, A. Nagy, and G. Gaderer, "Limits of synchronization accuracy using hardware support in IEEE 1588," in *2008 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, IEEE, 2008, pp. 12–16.

[17] H. Huckeba and R. Dlugy-Hegwer, "Precise time synchronization using IEEE 1588 for LXI applications," in *2006 IEEE Autotestcon*, IEEE, 2006, pp. 129–135.

[18] *XMC4700 / XMC4800 Microcontroller Series for Industrial Applications*, v1.3, Infineon Technologies AG, Jul. 2016.

[19] Kyocera, *Crystal units CX3225CA*, Dec. 2019.

[20] *TJA1051 high-speed CAN transceiver*, NXP Semiconductors, Nov. 2017.

[21] T. LeCroy, *HDO4000A 12-bit oscilloscopes datasheet*, Mar. 2019.

[22] P. Puschner, "Transforming execution-time boundable code into temporally predictable code," in *IFIP Working Conference on Distributed and Parallel Embedded Systems*, Springer, 2002, pp. 163–172.

[23] S. Corrigan, "Controller area network physical layer requirements," Texas Instruments, Tech. Rep., 2008.

Sascha Einspieler received his Dipl. Ing. degree in information technology in 2013 from the Alpen-Adria University Klagenfurt, Austria. His research interests include the design and development of distributed embedded systems as well as in-site measurement applications for semiconductor reliability testing. With his experience in microcontroller firmware development and embedded communication, he is a member of the reliability test equipment and methodology team at the KAI Kompetenzzentrum Automobil- und Industrieelektronik GmbH in Villach, Austria. Currently, he is pursuing his PhD studies where his research focuses on a flexible real-time control and diagnostic system for application-related stress testing.

Nirmal Krishna Rathakrishnan received his Bachelor degree in Mechatronics in 2017 from Anna University, Chennai, India and the M.Sc. degree in Mechatronics in 2021 from Technical University Hamburg, specializing in Embedded Systems. From April 2019 to May 2020, he was involved in development of swarm algorithms under the Institute of Control Systems, TUHH, Hamburg. From July 2020 to Feb 2021, he was writing his master thesis at KAI GmbH in Villach, under the field of distributed systems. His research interests include time synchronization over distributed systems, RF and CAN communication, microcontrollers, robotics, swarm algorithms and embedded firmware development.

Arpitha Prabhakara is currently carrying out doctoral research at KAI Kompetenzzentrum Automobil-u. Industrieelektronik GmbH. She has more than 7 years of experience in the field of embedded systems in the Semiconductor Industry. Arpitha obtained her bachelor's in Electrical & Electronics Engineering in 2010 from Visvesvaraya Technological University. During her Masters, she worked as student intern in Continental AG, and Airbus Defence and Space GmbH in Germany. In 2017, she graduated from Hochschule Darmstadt, Germany with Master's Degree in Electrical Engineering & Information Technology. Her research interests include Networking and Informatics, Distributed Control, Real-time Information Systems and Industrial Informatics.

Benjamin Steinwender studied at Carinthia University of Applied Sciences from 2005 to 2010 where he received BSc and MSc in the fields of Electronic Engineering and Microelectronics with distinction. Since August 2008, he is employed as a scientific researcher at KAI GmbH in Villach. Mr. Steinwender received his doctoral degree at Alpen-Adria Universität Klagenfurt in 2016 with distinction. His research interests include the design of distributed embedded in-situ measurement systems and data management for semiconductor reliability testing.

Wilfried Elmenreich is Professor of Smart Grids at the Institute of Networked and Embedded Systems at the Alpen-Adria-Universität Klagenfurt. He studied computer science at the Vienna University of Technology and in 2008 received the venia docendi for technical computer science. In 2007 he moved to the Alpen-Adria-Universität Klagenfurt as Senior Researcher. After a visiting professorship at the University of Passau Elmenreich in 2013, he followed the call to the University of Klagenfurt. Wilfried Elmenreich is a member of the Senate at the Alpen-Adria-Universität Klagenfurt, Counselor of the IEEE Student Branch, and is involved in the master program on Game Studies and Engineering. He is the author of several books and has published over 200 articles in the field of networked and embedded systems. Elmenreich research interests include intelligent energy systems, self-organizing systems and technical applications of swarm intelligence.