Secure Time Synchronization with Sub-Microsecond Accuracy in Controller Area Networks

Adrian Musuroi and Bogdan Groza

***Abstract*—We achieve sub-microsecond accuracy with an AUTOSAR-compliant time synchronization protocol on CAN-FD. In addition to this, we discover two attacks, double replays and forecasting, on the AUTOSAR CanTSyn standard and design fixes for them. Several simple and efficient algorithms are tested, e.g., weighted learning, windowed and continuous averaging, in order to determine the correct ratio between participants' clocks with minimal computational and communication overheads. We also point out that, at such a high level of synchronization accuracy, there may be significant differences when using simple or double precision floats for encoding the clock ratio with some of the algorithms. Our approach also exploits the Direct Memory Access subsystem instead of CPU interrupts during protocol executions, which reduces the processor load, making the solution suitable for real-time systems. We evaluate the proposed protocol in a realistic scenario by deploying it on an automotive-grade setup with Infineon Aurix development boards.**

***Keywords*-CAN, security, time synchronization, AUTOSAR**

I. Introduction and Related Work

Contemporary vehicles incorporate state of the art technologies that augment the driving experience while providing enhanced user comfort and delivering valuable feedback data through telemetry. From a design perspective, these technologies are mediated by dozens of Electronic Control Units (ECUs) which are connected through in-vehicle buses, more commonly the Controller Area Network (CAN), with the purpose of exchanging control, sensor, diagnostics and other types of data. In this context, time synchronization is a core pillar that supports proper coordination between ECUs and the environment. Highly accurate time synchronization enables ECUs to timestamp individual data points such that these can be correctly placed on a timeline and morphed into high-level objects facilitating driver assistance features. Security protocols are often relying on timestamps for validating the freshness of the received data during authenticity checks. Moreover, there are many applications that require time knowledge to create logs for telemetry or data recorders that can be subsequently used in forensic analysis. While indeed the previously mentioned tasks may not need sub-microsecond synchronization accuracy, the AUTOSAR specification of Time Synchronization over CAN (CanTSyn) [1] asks for a time base reference clock with a worst-case accuracy of $2\mu s$, and we consider it a plus that our proposal goes well below this limit. Moreover, in the attacks that we later discuss, we show that the insecurity of CanTSyn can be exploited to increase the synchronization error.

Adrian Musuroi and Bogdan Groza are with the Faculty of Automation and Computers, Politehnica University of Timisoara, Romania, Email: {adrian.musuroi, bogdan.groza}@aut.upt.ro

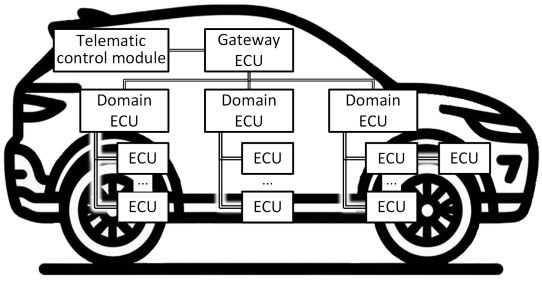


Fig. 1. Automotive domain topology: ECUs interconnected via CAN buses

Early work by Gergeleit and Streich [2], addressing CAN software-based time synchronization, achieved an accuracy of $\sim 20\mu s$ employing less than 20 synchronizations per second. Later, Lee and Allan [3] brought fault-tolerant enhancements to Gergeleit's protocol, and reported a synchronization accuracy of $\sim 10\mu s$ with 1-second cyclic synchronizations. Further improvements to Gergeleit's protocol were proposed by Akpinar et al. in [4] and [5]. In [4], a new strategy for timestamp gathering is employed to achieve a simulated accuracy of $\sim 9\mu s$, while the work in [5] applies a control loop to account for clock drifts, achieving the accuracy of $\sim 5\mu s$. The core mechanism from Gergeleit's protocol was also inherited by AUTOSAR's CanTSyn protocol [1], although it requires an additional frame per synchronization. In [5], Akpinar et al. evaluate the performance of CanTSyn. In their setup, the protocol is able to achieve $\sim 100\mu s$ accuracy with a standard implementation and up to $\sim 5\mu s$ accuracy when enhanced with a custom control loop for drift corrections. The authors then improved on drift corrections in [6], achieving $\sim 4\mu s$ accuracy. Further, Einspieler et al. [7] showed that $\sim 100ns$ accuracy can be obtained by applying the control loop from IEEE 1588 Ethernet interfaces for drift corrections in CanTSyn. Evidently, this scenario is applicable to targets that benefit from the availability of such peripherals. In all of the aforementioned publications, CPU interrupt service routines are employed for timestamping CAN Tx confirmation and Rx indication events in order to improve accuracy. Differently, to align with the constraints of some real-time systems, Luckinger and Sauter [8] base their work on the CanTSyn protocol with deferred CAN event processing in a real-time embedded operating system. The authors employ an exponential weighted average algorithm for clock rate adjustments, achieving an accuracy of $\sim 50\mu s$. Other lines of work implemented the IEEE Precision Time Protocol [9] to synchronize automotive-grade ECUs over CAN-FD. In [10], an accuracy of $\sim 1.26\mu s$ was achieved employing a single CAN-FD bus while in [11], the results

yield an accuracy of $\sim 12.32\mu s$ in a zone-based architecture. In both cases, Infineon TC275 targets were considered.

Without delving into details, we acknowledge works which rely on specialized hardware such as [12], [13], [14]. Interestingly, Akpinar et al. [4], [6] propose the use of the CAN phase-error for drift corrections. This approach does not require modifications to the CAN protocol, but access to information which CAN controllers typically do not provide. The authors show a simulated performance of $\sim 4.25\mu s$ [4] and later achieve $\sim 2\mu s$ accuracy in a practical setup [6]. Finally, Akpinar et al. propose a predictable timestamp gathering method [15] in which CAN events get triggered by the controller in the middle of a transmission, i.e., instead of in the beginning or at the end. Similar to the previous proposals, this approach requires a feature which is usually not provided by commercial controllers. Nevertheless, by employing this strategy, the authors achieve $\sim 1.4\mu s$ accuracy for software-based timestamp gathering as well as $\sim 120ns$ employing specialized hardware-based timestamps.

Last but not least, there is a substantial corpus of works on CAN bus security. Although none of them addresses the security of time synchronization protocols on CAN, some of them may have a tighter relation to time synchronization. For example, [16] uses clock skews to identify ECUs on the CAN bus, but this methodology was proven insecure since an adversary may change its clock skew to mimic the clock of a legitimate ECU [17] (this is called a cloaking attack). Needless to say, with use of the clock corrections introduced by CanTSyn and in particular if the ECUs dynamically adjust their skews as we later discuss, the previously mentioned defense or attack methodologies become obsolete (since all ECUs adjust their skews according to a reference time base).

As stated, to the best of our knowledge, while there are many papers addressing CAN security, this is the first proposal that addresses the security of CAN time synchronization protocols. To align with industry demands, we build upon AUTOSAR CanTSyn [1], which was introduced as early as 2014 as a standard way to synchronize ECU clocks over CAN buses in domain-based architectures, as suggested in Figure 1. As a brief summary, our contributions are threefold and organized as follows throughout this work. Firstly, in Section II, we discuss several limitations of the AUTOSAR CanTSyn which lead to two new attacks: double replays and forecasting. Secondly, in Section III, we introduce a variation which is based on individual authentication of participants, resulting in a secure protocol which deviates only from draft requirements of the latest AUTOSAR release [1], i.e., our proposal is compliant with the standardized elements of the protocol, allowing for immediate adoption. We also analyze four clock rate adjustment strategies that can be easily integrated into the standardized protocol for drastic performance improvements. Thirdly, in Section IV, we provide a comprehensive experimental analysis on automotive-grade setups, showing that the achieved accuracy is generally less than $1\mu s$, with top performances in the range of 40-200ns. In a novel implementation approach, we show that Direct Memory Access (DMA) can replace CPU interrupts for precise timestamp gathering, making highly accurate time synchronization suitable for real-time systems in which CPU interrupts may not be an option.

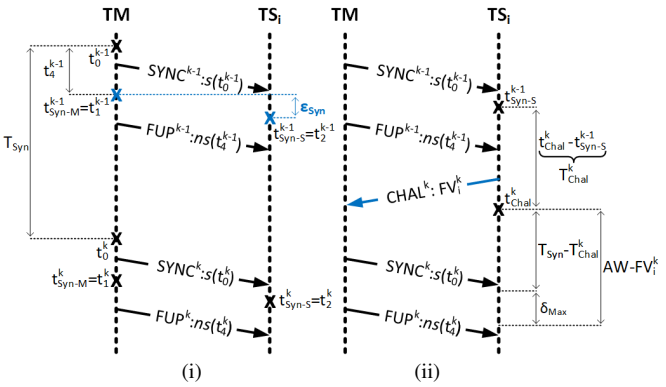


Fig. 2. Standardized AUTOSAR CanTSyn protocol (i) and our proposed variant (ii) with an additional CHAL frame sent by each TS

II. AUTOSAR CAN Time Synchronization

This section presents the standard AUTOSAR CanTSyn time synchronization protocol and its limitations.

A. Background

In the AUTOSAR architecture, time bases are maintained by the Synchronized Time-Base Manager (StbM) [18], which uses the CanTSyn protocol in order to synchronize with other ECUs that are on the same CAN bus. The standard defines two types of time bases: Synchronized Time Bases and Offset Time Bases. A Synchronized Time Base is a virtual clock that is maintained by an ECU and to which other units may synchronize through the CanTSyn protocol. An Offset Time Base is composed of a Synchronized Time Base and a fixed offset value which is added to it, i.e., it is not obtained directly through a clock synchronization protocol. With regards to CAN communication, both types of time bases require message exchanges. Although in this work we focus on the security of time synchronization messages, i.e., messages employed for adjusting Synchronized Time Bases, the same design principles can be easily applied to offset transmissions.

B. Time encoding and standardized protocol steps

Time information is stored by the StbM using a 48-bit *seconds field* and a 32-bit *nanoseconds field*, i.e., one nanosecond is the tiniest unit of time that can be represented. Similar to UNIX-based operating systems, the timestamp values indicate the amount of time that has passed since January 1st, 1970. For the purpose of CAN time synchronization, the standard employs only the least significant 32 bits of the *seconds field*. These bits cover a range of ~ 136 years and are considered sufficient for automotive applications. The remaining 16 most significant bits of the *seconds field* are typically used as an epoch indicator and they are not expected to change during the lifetime of a vehicle. In the upcoming discussions, we use the notation $s(\mathtt{t})$ according to the CanTSyn standard to denote the truncation to the least significant 32 bits of the *seconds field* of timestamp $\mathtt{t}$. Further, to denote the entire *nanoseconds field* of timestamp $\mathtt{t}$, we use the notation $ns(\mathtt{t})$.

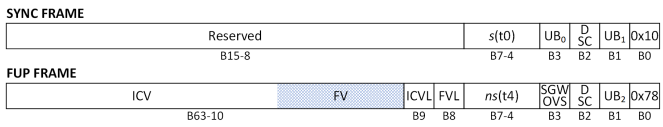


Fig. 3. Structure of the standardized AUTOSAR CanTSyn SYNC and FUP frames, in a configuration in which authentication is enabled

The time moderator (TM) sends periodical time synchronization data to the time subordinate (TS) ECUs. We denote the synchronization period as $\mathsf{T_{Syn}}$, which is a configurable value in the AUTOSAR specification. Each protocol execution consists of two sequential transmissions, i.e., a synchronization frame (SYNC) and a follow-up frame (FUP), as shown in Figure 2 (i). Since CanTSyn calls for periodical executions, we use k as a superscript to differentiate between protocol iterations. When a synchronization milestone is reached, TM measures its clock $\mathsf{t_0}$ and triggers the broadcast transmission of the truncated *seconds field* $s(\mathsf{t_0})$ in a SYNC message. Due to unforeseen hardware and software delays, it is expected for the actual message dispatch to encounter unpredictable delays. Thus, when the SYNC transmission completes, the TM captures $\mathsf{t_1}$ upon the CAN Tx complete event, while TS captures $\mathsf{t_2}$ upon the CAN Rx indication event. To compensate for the transmission delay, TM computes $\mathsf{t_4} = \mathsf{t_1} - s(\mathsf{t_0})$ and subsequently broadcasts $ns(\mathsf{t_4})$ in a FUP message. Given the payloads of the two messages, TS can then reconstruct $\mathsf{t_1}$. As timestamps $\mathsf{t_1}$ and $\mathsf{t_2}$ are employed for the actual synchronization, we will continue to refer to them as $\mathsf{t_{Syn\text{-}M}}$ and $\mathsf{t_{Syn\text{-}S}}$. Employing these notations, the TS computes its clock offset relative to the TM as:

$$\Delta_{\mathsf{CanTSyn}} = \mathsf{t_{Syn\text{-}S}} - \mathsf{t_{Syn\text{-}M}} \tag{1}$$

An intrinsic property of the CanTSyn protocol is that its synchronization error ϵ_{Syn} strictly depends on the distance between the moments of capturing $\mathsf{t_{Syn\text{-}M}}$ and $\mathsf{t_{Syn\text{-}S}}$ (clearly, in an ideal scenario this distance is zero). In other words, the synchronization error is $\epsilon_{\mathsf{Syn}} = \mathsf{t_{Syn\text{-}S}}' - \mathsf{t_{Syn\text{-}M}}'$, where $\mathsf{t_{Syn\text{-}S}}'$ and $\mathsf{t_{Syn\text{-}M}}'$ are timestamps captured by an external reference clock (later in the experiments we employ a logic analyzer for this purpose) when $\mathsf{t_{Syn\text{-}S}}$ and $\mathsf{t_{Syn\text{-}M}}$ are recorded. Since CanTSyn leverages the acknowledgment mechanism of the CAN protocol, which is immediate and requires a single confirmation bit, highly accurate synchronization is achievable – especially when timestamp gathering is implemented within transmission complete and data received interrupts.

C. Structure of protocol frames

Some fine-grained details on the structure of CanTSyn frames are worth mentioning. This structure is illustrated in Figure 3. Although certain elements can vary with the CanTSyn module configuration, we deliberately chose an instance in which message authentication is enabled so that we can assert the security of the protocol. The specifics of other configurations are not of relevance for the enhancements and results presented in this work, so we do not delve into further details regarding them. We note that all configurations in which authentication is enabled are supported only for CAN-FD buses and are in draft state in the latest release [1].

TABLE I
ATTACKS, MITIGATION AND SHORTCOMINGS OF CANTSYN

Attack	Mitigation	Shortcoming	Proposed fix
Spoofing	Group-MAC authentication	TS impersonates TM	Multi-MAC authentication
	Signature-based authentication	None	N/A
Delay	SYNC Tx delay compensation	Postponed transmissions possible	Challenge-response
Replay	Incorporate FV	Double replays set TS clock behind	Non-overridable SYNC
Forecasting	Reset and raise event E_SEQ	Set TS clock behind if followed by replay	Add nonce to SYNC

The SYNC and FUP messages are multiplexed under the same CAN ID according to the standard. A value for this CAN ID is not imposed nor suggested leaving it to the developer's choice. As depicted in Figure 3, the least significant byte is the multiplexer, i.e., the value that distinguishes between the SYNC and FUP message types. In both messages, the following byte encodes a user byte (UB) which can be unrestrictedly employed for embedding additional data if needed. Similarly, the third byte is equally split into the time domain identifier (D) which is used by the upper layer software for linking components to time bases, and a sequence counter (SC) that gets incremented for every protocol iteration. The fourth byte encodes another UB in the SYNC message. For FUP, this byte is split into the Synchronized to Gateway (SGW) and the Overflow Seconds (OVS) flags. SGW indicates whether the synchronization is received directly from the time base owner or, when the owner is on a different bus, from a gateway, while OVS flags whether a seconds overflow has occurred when computing $\mathsf{t_4}$, i.e., if $s(\mathsf{t_4})$ is greater than zero, in which case the TS needs to compensate for the overflow. For SYNC, the remaining bytes are reserved, while for FUP messages they are used for authentication purposes. The Freshness Value Length (FVL) specifies the length of the Freshness Value (FV). The content of the FV is not specified by the standard, however, the StbM specification [18] states that this value should be a counter or timestamp, thus preventing replay attacks. The Integrity Check Value Length (ICVL) specifies the length of the Integrity Check Value (ICV), i.e., a cryptographic authenticator computed over the SYNC and FUP payloads. The standard imposes the ranges [0,8] and [0,54] for the FVL and ICVL, with the condition that their sum does not exceed the 54 most significant bytes allocated for FV and ICV.

D. Security analysis and limitations

We consider the following four attack types: (i) *spoofing attacks*, in which the adversary sends bogus messages in an attempt to impersonate the TM, (ii) *delay attacks* in which the adversary withholds genuine synchronization frames for an arbitrary period of time, (iii) *replay attacks*, in which the adversary re-transmits legitimate time synchronization frames, and (iv) *forecasting attacks*, a variation of the classic *replay attacks* in which the adversary anticipates the payloads of legitimate time synchronization frames and transmits them

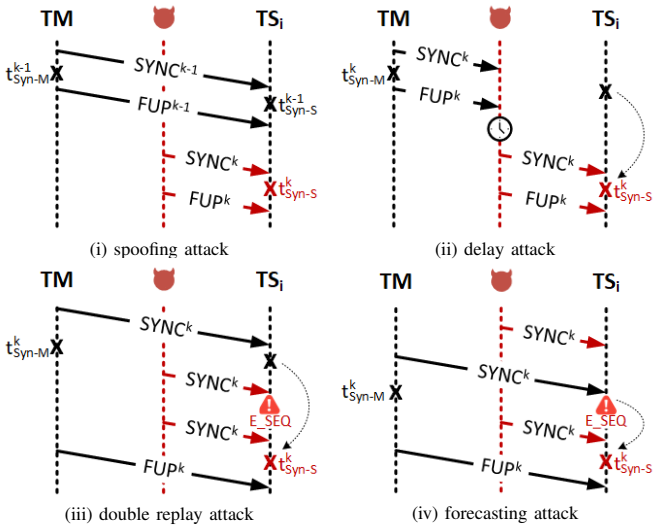


Fig. 4. Schematics of spoofing (i), delay (ii), double replay (iii) and forecasting (iv) attacks against the standardized AUTOSAR CanTSyn

beforehand. Against these four types of attack, CanTSyn has some resilience but there are also some shortcomings which are summarized in Table I and discussed in what follows.

Spoofing attacks. These attacks, suggested in Figure 4 (i), are addressed by CanTSyn through the inclusion of authenticators, i.e., the ICV values, in FUP frames. The AUTOSAR standard allows either digital signatures or group-MACs, i.e., MACs computed with a symmetric key that is shared between the TM and each of the TS nodes from the bus, to be employed for computing ICV values. Therefore, the ICV is computed according to the authentication method either as a signature or as a group-MAC: $\text{ICV-SIG}^k = \text{Sig}(\text{SYNC}^k[0{:}15] \,||\, \text{FUP}^k[0{:}9] \,||\, \text{FV}^k)$ or $\text{ICV-GMAC}^k = \text{MAC}(\text{SYNC}^k[0{:}15] \,||\, \text{FUP}^k[0{:}9] \,||\, \text{FV}^k)$. Here, $||$ denotes concatenation, while the brackets indicate byte ranges.

But group-MACs cannot identify which group member sent the data and therefore only the signature-based protocol variant is resilient against impersonation attacks launched by TS ECUs. This solution may not be always applicable since digital signatures bring additional computational, memory and communication overheads, which may not be supported by all in-vehicle controllers. Moreover, the size of the ICV field maxes out at 54 bytes and it is further constrained by the size of the FV value. As a consequence, the inclusion of a digital signature may be impractical in some setups.

Delay attacks. These attacks, suggested in Figure 4 (ii), are addressed in CanTSyn by compensating the transmission delay, i.e., through computing and broadcasting $ns(\mathsf{t}_4)$ in FUP. Thus, CanTSyn successfully prevents attacks that are based on delaying the dispatch of the SYNC frame, e.g., through the injection of frames with higher priority on the bus. However, this countermeasure is ineffective against man-in-the-middle attacks, e.g., through gateways, or against delayed executions, e.g., slowing down CPU interrupt routines. DMA timestamping prevents the latter while the freshness element FV has the purpose of preventing replays and delays. However, the management of this value is not covered by the standard, but outsourced to a Freshness Value Manager (FvM) module which has to be designed and implemented independently. Several design approaches are suggested in Annex A of the AUTOSAR Specification of Secure Onboard Communication (SecOC) [19], which rely either on synchronized time bases, i.e., timestamps, or synchronized monotonic counters. We note that the timestamp-based method may be unsatisfactory since the synchronization of the time bases is itself the objective of the protocol (resulting in a cyclic argument for security). On the other hand, the counter-based approach does not ensure the freshness of the data since a receiver cannot distinguish between a message that is delayed and one that is not (assuming a man-in-the-middle adversary). In this approach, e.g., according to [19], one or multiple freshness counters are synchronized between the sender and receivers, i.e., TM and each of the TS nodes from the bus. The counters are incremented for every CanTSyn protocol execution and periodically synchronized, e.g., at startup, through one-way control messages. A TS is then able to detect replays by comparing the received FV to its local counter. However, the TS will not be able to distinguish a fresh message from a delayed one, which could be orchestrated by a man-in-the-middle adversary.

Replay attacks. By enforcing non-repeatable FV values, CanTSyn prevents an adversary from replaying FUP frames or entire synchronization sequences. Regarding SYNC replays, the protocol disallows non-valid frame sequences ([1], p. 50, req. SWS_CanTSyn_00182) and receiving two consecutive SYNC frames will cause a TS to trigger an internal security event, denoted as E_SEQ in Figure 4 (iii), discard both SYNC frames, and reset its protocol state machine, i.e., proceed into the waiting for SYNC state. But while a succession of two SYNC frames triggers the security event, a third SYNC frame in this sequence will be accepted since the protocol state machine was reset and receivers are waiting for it. Consequently, a double replay will not be prevented. A *double replay attack* begins with the TM sending a SYNC message, followed by an immediate replay from the adversary (causing the E_SEQ event on the TS ECU), then the adversary replays the SYNC frame again before the TM proceeds with the FUP frame. This attack results in setting the TS clock behind, since $\mathsf{t}_{\text{Syn-M}}$ is captured on the transmission of the first SYNC frame, while $\mathsf{t}_{\text{Syn-S}}$ is captured on the reception of the third SYNC frame.

Forecasting attacks. A variation of the previous attack leads to *forecasting attacks*, as illustrated in Figure 4 (iv). Because $s(\mathsf{t}_0)$ and the sequence counter SC can be predicted based on the previous SYNC frame (note that the synchronization is cyclic, i.e., has a fixed period T_{Syn}), an adversary can anticipate legitimate SYNC payloads and send them beforehand. Subsequently, when the TM reaches the synchronization milestone and repeats the same SYNC frame, the TS will encounter an E_SEQ event. As in the previous attack, the adversary replays the SYNC frame again, before the FUP frame gets to be transmitted by the TM. This attack also results in setting the TS clock behind, with $\mathsf{t}_{\text{Syn-M}}$ being captured on the transmission of the second SYNC frame and $\mathsf{t}_{\text{Syn-S}}$ being captured on the reception of the third SYNC frame.

Finally, CanTSyn is restricted to offset corrections only. Following each synchronization, the accuracy steadily declines

until the subsequent correction is applied. Thus, the maximum error can only be constrained by elevating the synchronization frequency, resulting in increased bus and CPU overheads. In what follows, we address all these limitations.

III. Challenge-Response CanTSyn with Clock Rate Corrections

This section covers our proposed protocol modifications for improving the security and accuracy of CanTSyn.

A. Strengthening CanTSyn security

To tackle the aforementioned security shortcomings of CanTSyn, we use the following key modifications: (i) we use an authentication strategy based on multi-MACs to prevent *spoofing attacks* performed by a TS on the TM, (ii) we introduce a challenge-response protocol to prevent *delay attacks*, (iii) we introduce a requirement for non-overridable SYNC frames to prevent *replay attacks* and (iv) we add a random nonce in the SYNC frame in order to prevent *forecasting attacks*. These modifications are detailed in what follows.

Figure 2 (ii) illustrates the proposed protocol variation. We reserve an independent tuple $(\mathsf{FV_i}, \mathsf{ICV_i})$ for each $\mathsf{TS_i}, \mathsf{i} \in [1, 18]$, where $\mathsf{FV_i}$ is the freshness value that is managed by $\mathsf{TS_i}$ and $\mathsf{ICV_i}$ is the authenticator computed by the TM for $\mathsf{TS_i}$ as: $\mathsf{ICV}\text{-}\mu\mathsf{MAC}_\mathsf{i}^\mathsf{k} = \mathsf{MAC}(\mathsf{SYNC}^\mathsf{k}[0{:}15] \,||\, \mathsf{FUP}^\mathsf{k}[0{:}9] \,||\, \mathsf{FV}_\mathsf{i}^\mathsf{k})$.

The formats of the protocol frames are adopted as shown in Figure 5. For both frames, the new format preserves the fields covered by the least significant 10 bytes. We take advantage of the reserved bytes from the SYNC frame to embed a random nonce, making *forecasting attacks* impractical due to the difficulty of predicting this value. In FUP, since the task of managing FV values is delegated to the subordinates themselves, the FVL field is set to 0, indicating that the TM does not include any FV information in FUP transmissions. The remaining 54 most significant bytes, i.e., which carry one ICV field in the original protocol, are then split into 3-byte ICV fields, where $\mathsf{ICV_i}, \mathsf{i} \in [1, 18]$ is the authenticator intended for $\mathsf{TS_i}$. Figure 5 also introduces the structure of a new frame type, denoted as CHAL, to be employed by TS units for transmitting freshness values. We designate the least significant byte of the frame to serve as a message type identifier. Further, the second least significant byte holds the $\mathsf{TS_i}$ identifier within its bus $\mathsf{SID} = \mathsf{i}, \mathsf{i} \in [1, 18]$. The next 8 bytes hold the value of $\mathsf{FV_i}$, followed by a 6-byte ICV field that prevents impersonation attacks against subordinates. Before synchronization milestone k is reached, every $\mathsf{TS_i}$ sends its own nonce $\mathsf{FV_i^k}$ and captures the timestamp $\mathsf{t_{Chal}^k}$ when the transmission of the CHAL frame completes. Then, by employing its CanTSyn timestamp from the previous synchronization, i.e., $\mathsf{t_{Syn\text{-}S}^{k-1}}$, the CanTSyn synchronization period, i.e., $\mathsf{T_{Syn}}$, and a tolerance factor δ_{Max} for benign delays, $\mathsf{TS_i}$ creates an acceptance window for $\mathsf{FV_i^k}$ as $\mathsf{AW}\text{-}\mathsf{FV_i^k} = \mathsf{T_{Syn}} - (\mathsf{t_{Chal}^k} - \mathsf{t_{Syn\text{-}S}^{k-1}}) + \delta_{\mathsf{Max}}$. For a missing $\mathsf{FV_i^k}$ value, i.e., $\mathsf{TS_i}$ fails to communicate the freshness before protocol iteration k starts, the TM will simply omit the computation and set $\mathsf{ICV_i^k}$ to 0, without affecting the authentication process for the other subordinates. Further, to prevent *replay attacks*, we require the implementation to enforce non-overridable SYNC receptions. In this setting, the reception of two consecutive SYNC frames will not cause the TS to reset its internal state machine, but will still trigger the E_SEQ security event. Moreover, the second SYNC frame shall not override the payload or the Rx timestamp of the first SYNC frame. We note that no additional implementation is required for handling dropped FUP frames, e.g., TM fails to send the FUP frame after the SYNC frame, since AUTOSAR already implements a timeout period ([1], p. 50) which causes the TS to reset its CanTSyn state machine if no FUP frame arrives within the tolerated time span. To help us validate the security of the proposed protocol fix, we wrote a small ASLAN model of the CanTSyn protocol, then we used CLAtse, one of the model-checkers of the AVANTSSAR platform [20], to verify the attacks and fixes. As expected, the model immediately discovered the previously mentioned *forecasting attack*, while the attack was no longer discoverable once introducing the nonce in the SYNC package.

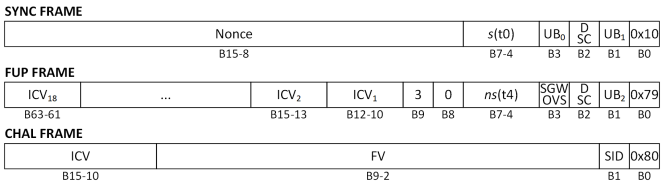


Fig. 5. CHAL frame structure and our SYNC/FUP adaptations for improved security (nonce in SYNC frame and multiple 3-byte ICV fields in FUP frame)

Our proposal trades between bus load and ECU resources in order to fulfill the security demands which are not satisfied by the AUTOSAR variants, i.e., FV management as well as resilience against various attacks. Most of the additional load falls on top of the TM. To synchronize N nodes, generating multi-MACs instead of group-MACs brings $\mathsf{N}-1$ additional MAC computations and keys. Then, the TM needs N additional MAC computations to check the authenticity of the CHAL frames. For each of the TS nodes, only one additional MAC operation is needed for authenticating CHAL frames. Finally, the challenge-response approach for managing FV values increases the bus load by N additional CHAL frames.

B. Enhanced clock rate correction from CanTSyn timestamps

While the previously discussed protocol variation has better security properties, we are still to answer the efficiency of integrating clock rate correction algorithms into CanTSyn. We introduce a clock rate correction parameter which we denote as ϕ. The value of ϕ defaults to 1 and gets adjusted by the TS during each CanTSyn protocol iteration. We employ the CanTSyn timestamps to compute the drift rate of the TS relative to the TM as: $(\mathsf{t_{Syn\text{-}M}^k} - \mathsf{t_{Syn\text{-}M}^{k-1}})/(\mathsf{t_{Syn\text{-}S}^k} - \mathsf{t_{Syn\text{-}S}^{k-1}})$.

We explore four different algorithms for adjusting ϕ: immediate adjustment, weighted learning, windowed averaging and continuous averaging. A trivial approach is contained in Algorithm 1, where ϕ is immediately updated based on the most recent clock drift measurement. An obvious shortcoming of this method is its unaccounted exposure to glitched delays

Algorithm 1 Rate adjustment based on immediate observation

procedure UPDATEPHIONFUPRECEIVED($t^k_{\text{Syn-M}}$, $t^k_{\text{Syn-S}}$, $t^{k-1}_{\text{Syn-M}}$, $t^{k-1}_{\text{Syn-S}}$)
  $\phi \leftarrow (t^k_{\text{Syn-M}} - t^{k-1}_{\text{Syn-M}})/(t^k_{\text{Syn-S}} - t^{k-1}_{\text{Syn-S}})$
  $t^{k-1}_{\text{Syn-M}} \leftarrow t^k_{\text{Syn-M}}$
  $t^{k-1}_{\text{Syn-S}} \leftarrow t^k_{\text{Syn-M}}$
  return ϕ
end procedure

Algorithm 2 Rate adjustment with weighted learning

procedure UPDATEPHIONFUPRECEIVED($t^k_{\text{Syn-M}}$, $t^k_{\text{Syn-S}}$, $t^{k-1}_{\text{Syn-M}}$, $t^{k-1}_{\text{Syn-S}}$)
  $\phi \leftarrow \alpha\phi + \beta(t^k_{\text{Syn-M}} - t^{k-1}_{\text{Syn-M}})/(t^k_{\text{Syn-S}} - t^{k-1}_{\text{Syn-S}})$
  $t^{k-1}_{\text{Syn-M}} \leftarrow t^k_{\text{Syn-M}}$
  $t^{k-1}_{\text{Syn-S}} \leftarrow t^k_{\text{Syn-M}}$
  return ϕ
end procedure

Algorithm 3 Rate adjustment with windowed averaging

procedure UPDATEPHIONFUPRECEIVED($t^k_{\text{Syn-M}}$, $t^k_{\text{Syn-S}}$, $t^{k-1}_{\text{Syn-M}}$, $t^{k-1}_{\text{Syn-S}}$)
  $\text{w\_buf}[\text{w\_idx}\%\text{w\_size}] \leftarrow (t^k_{\text{Syn-M}} - t^{k-1}_{\text{Syn-M}})/(t^k_{\text{Syn-S}} - t^{k-1}_{\text{Syn-S}})$
  $\text{w\_idx} \leftarrow \text{w\_idx} + 1$
  if $\text{w\_idx} > \text{w\_size}$ **then**
    $\phi \leftarrow \text{SUM}(\text{w\_buf})/\text{w\_size}$
  end if
  $t^{k-1}_{\text{Syn-M}} \leftarrow t^k_{\text{Syn-M}}$
  $t^{k-1}_{\text{Syn-S}} \leftarrow t^k_{\text{Syn-M}}$
  return ϕ
end procedure

Algorithm 4 Rate adjustment with continuous averaging

procedure UPDATEPHIONFUPRECEIVED($t^k_{\text{Syn-M}}$, $t^k_{\text{Syn-S}}$, $t^{k-1}_{\text{Syn-M}}$, $t^{k-1}_{\text{Syn-S}}$)
  if $\text{count} < \text{max\_count}$ **then**
    $\text{acu} \leftarrow \text{acu} + (t^k_{\text{Syn-M}} - t^{k-1}_{\text{Syn-M}})/(t^k_{\text{Syn-S}} - t^{k-1}_{\text{Syn-S}})$
    $\text{count} \leftarrow \text{count} + 1$
    $\phi \leftarrow \text{acu/count}$
    $t^{k-1}_{\text{Syn-M}} \leftarrow t^k_{\text{Syn-M}}$
    $t^{k-1}_{\text{Syn-S}} \leftarrow t^k_{\text{Syn-M}}$
  end if
  return ϕ
end procedure

that may lead to high accuracy fluctuations. The following algorithms address this shortcoming by accounting for past measurements as well. Algorithm 2 employs weighted learning, where we set parameters α, β to control the learning rate, i.e., $[(\alpha, \beta) \mid \alpha, \beta \in [0, 1], \alpha + \beta = 1]$. Clearly, a higher β value leads to faster learning with higher fluctuations, while a higher α will induce a slower but more stable convergence towards the ideal rate correction value. Next, Algorithms 3 and 4 rely on averaging multiple drift rate measurements. In Algorithm 3 the average is computed over a fixed window limited to w_size, while Algorithm 4 continuously accumulates values until a maximum iteration number max_count is reached, i.e., to prevent overflows. To provide a visual comparison of the algorithms, we simulate the evolution of ϕ for each of them in Figure 6, where we induce random noise and a glitch at the 50th synchronization. For weighted learning, we simulated the algorithm in two configurations: $\alpha = 0.9, \beta = 0.1$ as well as $\alpha = 0.5, \beta = 0.5$. Clearly, the immediate algorithm is the most susceptible to noise and glitches. For the weighted adjustments, the $\alpha = 0.9, \beta = 0.1$ configuration is more stable, however, the convergence towards the ideal ϕ value is slow. The averaging algorithms are converging faster and present little fluctuations overall. Since continuous averaging takes into account the greatest number of drift values, its stability outperforms all other algorithms.

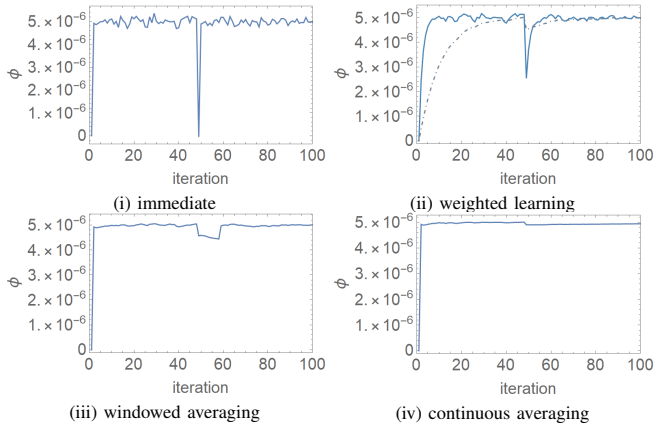


Fig. 6. Simulated performance of the (i) immediate, (ii) weighted learning, (iii) windowed averaging and (iv) continuous averaging clock rate adjustment algorithms, with random noise and a glitch inserted at the 50th iteration

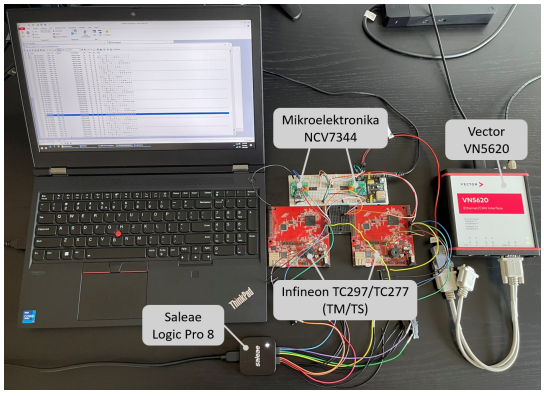


Fig. 7. Experimental setup with two Infineon controllers, NCV7344 CAN-FD transceivers and Logic Pro 8, VN5620 measuring devices

IV. Implementation and Results

This section presents the implementation and the experiments that we carry in order to determine the best alternative.

A. Experimental setup

We deployed the solution on two AURIX TriBoard development boards: a TC297 board as TM and a TC277 board as TS. The targets were configured to operate at their maximum CPU speed, i.e., 300MHz for the TC297 and 200MHz for the TC277 microcontroller, while the CAN controllers were set at nominal baud rate 500kbps and fast baud rate 2Mbps. We used external CAN-FD transceivers, i.e., NCV7344 from Mikroelektronika, which were connected through jump wires

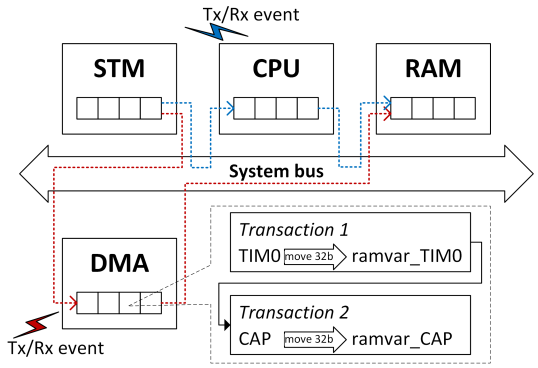


Fig. 8. Depiction of the system response on CAN events, emphasizing the difference between CPU (blue path) and DMA (red path) timestamping

and a breadboard. We later transitioned to a setup that is representative of automotive CAN networks, by connecting the boards to a real in-vehicle harness [21] and replaying a trace that was collected from a real vehicle. The traffic is composed of 89 classical CAN frames, i.e., up to 8 bytes of payload, with cycle times ranging from $10ms$ up to $3s$ and IDs ranging from 0x20 to 0x550, resulting in a total bus load of $\sim 37\%$. On top of this, we simulated up to 15 additional TSs in order to evaluate the performance under full protocol load. As for the CAN IDs required for time synchronization, we simply chose the values 0x220 (SYNC and FUP) and 0x330-0x33F (CHAL) such that they don't overlap with any existing IDs from the trace. We note that in this configuration the time synchronization messages do not have the highest priority on the bus. The tools that supported our evaluation are the Saleae Logic Pro 8 analyzer, which operates at 500MS/s, together with Vector's VN5620 CAN Interface and CANoe 16 software. Figure 7 illustrates the experimental setup.

B. Software implementation

The software implementation was supported by Infineon's AURIX Development Studio toolchain. We developed a stack of software components that manage the virtual time, CAN communication, IO channels and so forth.

Virtual time. In our implementation, the ECU virtual clock is backed up by the System Timer (STM), i.e., a 64-bit free-running timer operating at the frequency of 100MHz, which can be synchronously read through the TIM0 and CAP 32-bit registers. We configured the timer to generate CPU interrupts every $50ms$, by the means of which we cyclically update the ECU virtual clock. When queried for the current time, the software reads STM and adds the elapsed ticks since the last interrupt to the latest virtual clock value. Further, we added a clock rate correction parameter (with the default value of 1.0) which has the option to be applied statically, i.e., only when the virtual clock is read, or dynamically, i.e., every $50ms$ as well, when the virtual clock is updated.

CAN Tx/Rx events and the DMA. The synchronization error, i.e., ϵ_{Syn}, strictly depends on how close Tx and Rx events are timestamped. Our first strategy employs the handling of the events through CPU interrupts, during the execution of which we immediately collect the timestamps. The second strategy leverages the on-chip DMA controller for timestamp gathering. In this case, CAN events are configured to trigger the DMA engine to perform a set of two 32-bit move operations, i.e., one for each of the TIM0 and CAP registers, while the CPU resorts to deferred processing of the frame contents. Since the offset between TIM0 and CAP is 0x1C and the address of TIM0 is not 64-bit aligned, i.e., a single transaction with circular buffers would necessitate a high number of iterations, we opted for a configuration of two DMA transactions consisting of one 32-bit transfer each. We depict the distinction between CPU and DMA timestamping in Figure 8. To enforce the requirement of non-overridable SYNC frames, the DMA channel was configured to operate in single mode, in which it gets disabled after each SYNC frame reception. The CPU has to re-enable the channel after processing the SYNC frame.

Security parameters for CanTSyn-μMAC. We configured the protocol to be cyclically executed every second, i.e., $\text{T}_{\text{Syn}} = 1s$. For frame authentication, we use 128-bit symmetric keys and AES-CMAC from WolfCrypt (https://www.wolfssl.com/products/wolfcrypt/). Further, when clock rate corrections are enabled, we experiment with both single and double precision floating-point numbers with the compiler set to comply with the IEEE standard for floating-point arithmetic [22], i.e., TASKING option "0 - Strict IEEE-754".

C. Data collection

To measure the synchronization error, i.e., ϵ_{Acc}, we implemented IO interrupts during the beginning of which the ECUs capture timestamps denoted as $\text{t}_{\text{Acc-M}}$ and $\text{t}_{\text{Acc-S}}$. The interrupts are triggered approximately $800ms$ after each protocol execution, i.e., 80% of the synchronization interval, and the resulting $\text{t}_{\text{Acc-M}}$ and $\text{t}_{\text{Acc-S}}$ timestamps are subsequently transmitted through dedicated CAN frames. Additionally, the subordinate will also embed into the CAN frame its latest clock correction rate ϕ. To account for delays regarding the launch of the IO interrupt service routines, e.g., if one of the targets is busy handling a different interrupt with higher priority when the input pin toggle happens, we employ the logic analyzer to timestamp the moments before the virtual clocks are queried. We denote these logic analyzer timestamps as $\text{t}_{\text{IO-M}}$ and $\text{t}_{\text{IO-S}}$. The synchronization error is therefore computed as:

$$\epsilon_{\text{Acc}} = (\text{t}_{\text{Acc-M}} - \text{t}_{\text{Acc-S}}) - \underbrace{(\text{t}_{\text{IO-M}} - \text{t}_{\text{IO-S}})}_{\epsilon_{\text{IO}}} \qquad (2)$$

D. Results

We conducted an extensive suite of experiments to determine the performance in various settings. Our results are summarized in Table II, which is divided into four test scenarios.

In the first two scenarios, which include 23 experiments with all of the previously described algorithms, the CAN bus is built with simple jump wires and the timestamp gathering is performed employing either CPU interrupts or the DMA engine. For the latter two scenarios, which include 4 experiments with the best performing algorithms, we switched to the real-world automotive CAN bus on which we replayed the

TABLE II
PERFORMANCE EVALUATION OF ALGORITHMS IN VARIOUS SCENARIOS

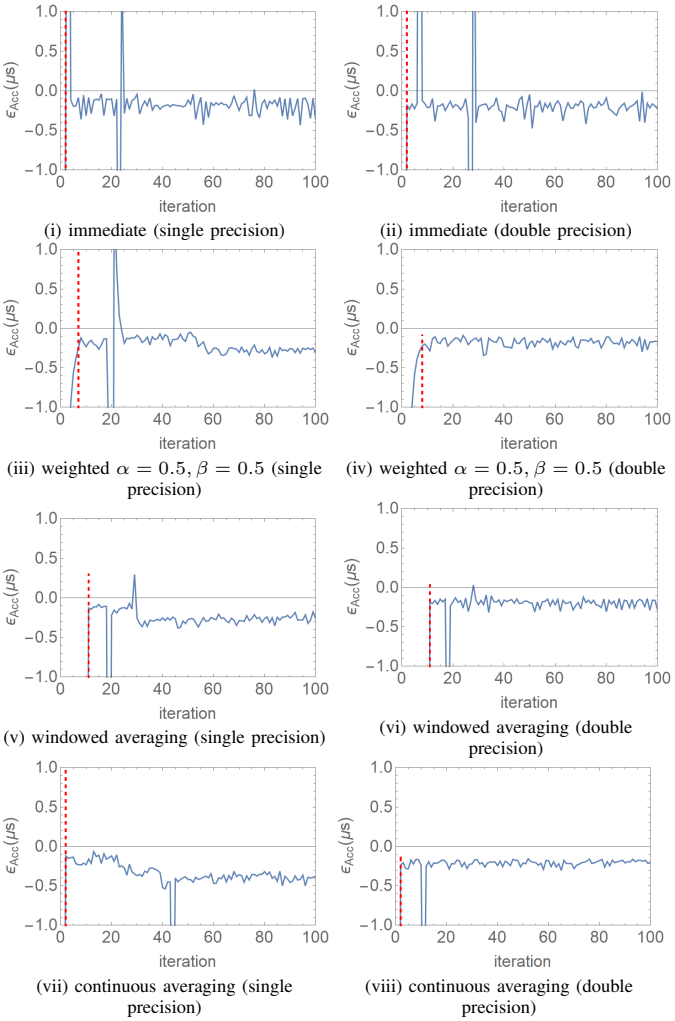
Cnt.	Adjustment alg.	FP	Adj.	$\mathsf{T}_{\mathsf{Reach}}$ (iter.)	Acc. (μs)	$\mathsf{T}_{\mathsf{Stab}}$ (iter.)	Acc. (μs)	Complete trace Median (μs)	Complete trace Mean (μs)	Complete trace SD (μs)	Q4 Median (μs)	Q4 Mean (μs)	Q4 SD (μs)	$\overline{\epsilon}_{\mathsf{Acc}}$ (μs)
CPU interrupts for timestamp gathering, CAN bus from jump wires														
1	N/A (CanTSyn only)	N/A	N/A	N/A	N/A	11	-6.778	6.584	6.605	0.073	6.544	6.543	0.022	490.73
2	Immediate	Double	Dyn.	25	-0.205	18	-0.360	0.323	0.449	1.115	0.330	0.323	0.093	24.21
3	Immediate	Double	St.	2	-0.174	16	-0.156	0.197	0.303	0.851	0.184	0.210	0.099	15.76
4	Immediate	Single	Dyn.	9	-0.092	12	-0.312	0.221	0.330	1.065	0.186	0.224	0.128	16.81
5	Immediate	Single	St.	2	-0.180	47	-0.186	0.159	0.238	0.613	0.182	0.184	0.122	13.80
6	Weighted (0.9/0.1)	Double	Dyn	37	-0.237	39	-0.425	0.291	0.502	0.916	0.243	0.272	0.102	20.37
7	Weighted (0.9/0.1)	Double	St.	41	-0.249	50	-0.267	0.203	0.429	0.877	0.195	0.201	0.032	15.05
8	Weighted (0.9/0.1)	Single	Dyn.	113	-0.222	97	-0.464	0.424	0.580	0.821	0.370	0.320	0.122	24.01
9	Weighted (0.9/0.1)	Single	St.	N/A	N/A	41	-0.630	0.489	0.691	0.906	0.418	0.428	0.048	32.15
10	Weighted (0.5/0.5)	Double	Dyn.	7	-0.227	20	-0.119	0.231	0.322	0.572	0.216	0.250	0.117	18.74
11	Weighted (0.5/0.5)	Double	St.	8	-0.219	18	-0.219	0.171	0.225	0.435	0.172	0.186	0.049	13.95
12	Weighted (0.5/0.5)	Single	Dyn.	8	-0.247	56	-0.124	0.184	0.255	0.484	0.130	0.177	0.139	13.25
13	Weighted (0.5/0.5)	Single	St.	7	-0.240	27	-0.132	0.239	0.324	0.843	0.226	0.231	0.042	17.32
14	Windowed avg. (10)	Double	Dyn.	10	0.224	41	-0.055	0.258	0.511	1.306	0.266	0.287	0.149	21.53
15	Windowed avg. (10)	Double	St.	10	-0.189	23	-0.165	0.198	0.438	1.219	0.202	0.216	0.058	16.18
16	Windowed avg. (10)	Single	Dyn.	11	-0.242	32	-0.168	0.334	0.520	1.141	0.314	0.312	0.118	23.39
17	Windowed avg. (10)	Single	St.	10	-0.128	40	-0.258	0.282	0.512	1.223	0.262	0.265	0.044	19.87
18	Continuous avg.	Double	Dyn.	4	-0.085	19	-0.117	0.333	0.373	0.513	0.329	0.324	0.159	24.27
19	Continuous avg.	Double	St.	2	-0.234	19	-0.166	0.205	0.240	0.394	0.211	0.215	0.030	16.14
20	Continuous avg.	Single	Dyn.	4	-0.197	16	-0.317	0.679	1.158	0.988	2.276	2.167	0.405	162.55
21	Continuous avg.	Single	St.	2	-0.116	16	-0.220	0.555	1.110	0.926	2.320	2.163	0.415	162.23
DMA engine for timestamp gathering, CAN bus from jump wires														
22	Windowed avg. (10)	Double	St.	11	0.136	25	0.204	0.192	0.387	1.159	0.176	0.161	0.049	12.10
23	Continuous avg.	Double	St.	2	0.188	16	0.156	0.214	0.250	0.468	0.220	0.223	0.130	16.72
CPU interrupts for timestamp gathering, CAN bus from real in-vehicle harness, close ECUs														
24	Windowed avg. (10)	Double	St.	11	-0.117	28	-0.123	0.146	0.371	1.160	0.145	0.157	0.041	11.80
25	Continuous avg.	Double	St.	2	-0.115	12	-0.115	0.145	0.178	0.372	0.162	0.170	0.039	12.73
CPU interrupts for timestamp gathering, CAN bus from real in-vehicle harness, distant ECUs														
26	Windowed avg. (10)	Double	St.	11	-0.191	22	-0.154	0.165	0.383	1.138	0.164	0.176	0.038	13.19
27	Continuous avg.	Double	St.	3	-0.170	39	-0.071	0.066	0.124	0.596	0.047	0.055	0.034	4.09

in-vehicle collected trace. In this setting, we first mounted the ECUs closer to each other, i.e., $1.9m$ apart, after which we placed them at the bus ends, i.e., $5.1m$ apart. The first column of Table II indicates the experiment number. Columns two to four specify which clock rate adjustment algorithm is applied, the configured precision of the floating-point engine, and the configured clock rate correction type, i.e., static or dynamic. The following columns characterize the protocol performance.

We use several metrics to assert the efficiency of the algorithms. Firstly, we use $\mathsf{T}_{\mathsf{Reach}}$ to denote the number of CanTSyn protocol iterations until reaching the $0.250\mu s$ accuracy threshold, which was chosen based on empirical evidence. Secondly, we use $\mathsf{T}_{\mathsf{Stab}}$ to denote the number of protocol iterations until the estimation stabilizes - this is defined as the iteration when a smoothed window computed over the last 10 consecutive accuracy measurements yields an error of less than $0.001\mu s$, which was chosen since one nanosecond is the tiniest unit of time that can be represented. For each threshold, we display the synchronization error that was measured upon reaching it. Thirdly, we consider the median, mean and standard deviation which are usual statistical metrics. We present these metrics as computed over the entire trace as well as over the last quarter, i.e., Q4, when the synchronization is presumably more stable. Finally, for Q4, we also computed the total synchronization error $\overline{\epsilon}_{\mathsf{Acc}}$ as the sum of the absolute values of all individual synchronization errors ϵ_{Acc}.

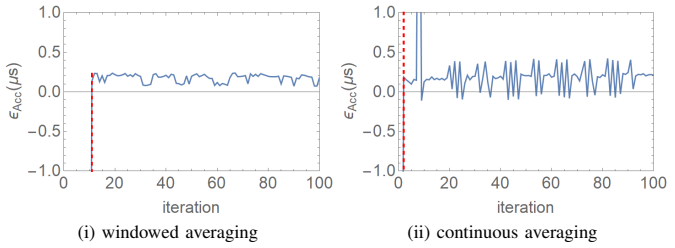
1) Free-run and CanTSyn-μMAC without rate corrections: For our first experiment, we disabled clock corrections altogether to observe the individual clock drifts of the two boards employing the VN clock as reference. During one second, the TM clock remains behind the VN clock by an average of $75.4\mu s$, while the subordinate lags behind by an average of $83.2\mu s$. During these measurements we found that the TM clock is more stable, with a clock jitter of approximately $\pm 1.25\mu s$ from the average value. In comparison, the precision of the TS clock fluctuates by approximately $\pm 3.5\mu s$. Next, we enabled CanTSyn offset corrections in order to establish a baseline for the protocol performance. Unsurprisingly, the execution period of one second is not sufficient to maintain sub-microsecond accuracy. As Table II shows, the threshold $\mathsf{T}_{\mathsf{Reach}}$ is never reached, instead the protocol stabilizes fairly soon with an accuracy of $-6.778\mu s$. This error value is in line with the clock drifts that we previously mentioned.

2) CanTSyn-μMAC with clock rate corrections: Clearly, clock rate corrections enable substantial performance improvements. In all cases, we observed that dynamic corrections exhibit greater fluctuations and therefore our forthcoming discussions are focused on static corrections exclusively. For weighted learning, we first evaluated the $\alpha = 0.9, \beta = 0.1$ configuration and concluded it to be too slow, in some cases not reaching the $\mathsf{T}_{\mathsf{Reach}}$ threshold at all. We then tuned the weighted learning algorithm to $\alpha = 0.5, \beta = 0.5$. For windowed averaging, we chose the window size of 10. Figure 9 illustrates the accuracy evolution associated with each measurement. Here, the results from the left column were obtained with single precision floating-point numbers, while the right column contains results obtained employing double precision floats. We outline with vertical lines the iterations in which threshold $\mathsf{T}_{\mathsf{Reach}}$ is surpassed. Generally, all algorithms yield better results when their computations are based on doubles. Opting for single precision does not compromise by much, except for the continuous averaging method, where the



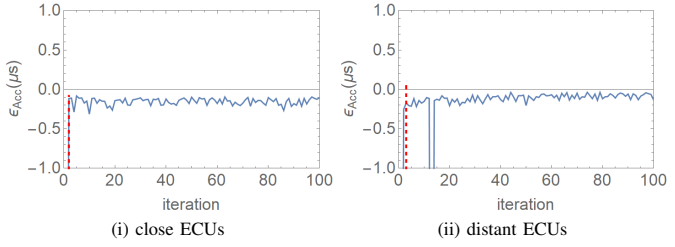
(i) immediate (single precision)
(ii) immediate (double precision)
(iii) weighted $\alpha = 0.5, \beta = 0.5$ (single precision)
(iv) weighted $\alpha = 0.5, \beta = 0.5$ (double precision)
(v) windowed averaging (single precision)
(vi) windowed averaging (double precision)
(vii) continuous averaging (single precision)
(viii) continuous averaging (double precision)

Fig. 9. Synchronization error ϵ_{Acc} for various algorithms during the first 100 iterations, the vertical lines mark the attainment of $\mathbf{T_{Reach}}$



(i) windowed averaging
(ii) continuous averaging

Fig. 10. Synchronization error ϵ_{Acc} during the first 100 iterations, for windowed averaging (i) and continuous averaging (ii) using DMA timestamps, the vertical lines mark the attainment of $\mathbf{T_{Reach}}$



(i) close ECUs
(ii) distant ECUs

Fig. 11. Synchronization error ϵ_{Acc} during the first 100 iterations, using the continuous averaging algorithm in the real-world automotive test bed, the vertical lines mark the attainment of $\mathbf{T_{Reach}}$

TABLE III
IMPACT OF THE ADDITIONAL CHAL FRAMES ON BUS LOAD, MEASURED ON COLLECTED TRACE WITH NOMINAL 500KBPS BAUD RATE

	No. of subordinates, i.e., of CHAL frames per synchronization						
	0	1	4	7	10	13	16
Min(%)	36.28	36.32	36.43	36.50	36.44	36.41	36.44
Max(%)	36.68	36.79	36.89	37.04	37.10	37.24	37.35
Avg.(%)	36.43	36.47	36.58	36.70	36.82	36.93	37.05

accuracy declines quickly due to cumulative precision losses. This suggests that when the target does not support double precision floats, other algorithms should be preferred. In this case, windowed averaging is the most suitable candidate since it has the best resilience against sporadic glitches. Indeed, during our measurements we encountered CAN controller glitches which lead to ϵ_{Syn} spikes and subsequently to significant ϕ fluctuations as suggested by some plots in Figure 9. Further, for scenarios in which doubles are available, we conclude that the accumulated averaging algorithm outperforms the others. Although the mean and median metrics from Table II are comparable, the standard deviation proves that accumulated averaging is the most stable approach.

3) DMA timestamp gathering: We evaluated the windowed and continuous averaging algorithms employing timestamps acquired through the DMA engine. Consistent with our previous results, static corrections outperform dynamic corrections. As shown in Figure 10, the target encountered extensive clock jitter at the beginning of the latter measurement, i.e., when continuous averaging was employed. According to Table II, the clock was stabilized in Q4 and the results are comparable to the CPU timestamp gathering use case. For windowed averaging, the clock encountered lenient jitter and the results are better than in the CPU timestamp gathering use case. We therefore do not attribute the fluctuations to the DMA engine, but rather to the clock instabilities of the TC277 board.

4) In-vehicle CAN network with real-world traffic: Finally, our results show that the proposed solution is applicable to real-world scenarios as well. The visual representation of these results is presented in Figure 11. Again, we ran the windowed and continuous averaging algorithms, obtaining the best results so far. The bus load measurements are shown in Table III. The case when no CHAL frames are transmitted corresponds to the bus load obtained with the original protocol from the AUTOSAR specification [1]. When evaluating our proposal, we observed a bus load increase of under 1%. Furthermore, all results presented in Table III were obtained using the 500kbps baud rate for the entire frame. When switching to the fast 2Mbps baud rate for data, we measured an average bus load of 36.54% in case of 16 subordinates. This result emphasizes the minimal impact of the additional CHAL frames on bus load. Based on our measurements with the logic analyzer, the transmission time of a CHAL frame is $\sim 130\mu s$ and therefore if all 16 CHAL frames are issued simultaneously, a sender of a low-priority frame will encounter a delay of $\sim 2.08ms$. For a deeper analysis of the delays, we use the recently proposed framework from [23], which extends the reference work [24] to CAN-FD. In order to determine the *worst-case response time* (WCRT) for each message M_i, i.e., $R_i = \max_{0 \le q < Q_i} R_i(q)$, where

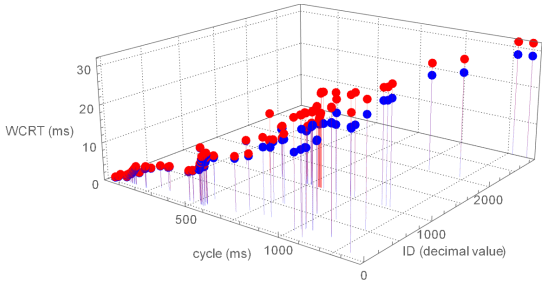


Fig. 12. WCRT for all IDs computed over the original trace (blue) and after adding the CanTSyn frames (red)

i is the frame ID and Q_i the number of expected transmissions of M_i during the bus busy period, we solve the equation:

$$R_i(q) = J_i + w_i(q) - qP_i + C_{D_i} \tag{3}$$

Where J_i is the queueing jitter, w_i is the worst-case transmission delay, P_i is the transmission period and C_{D_i} is the worst-case transmission time of message M_i. Due to the low bus load, all computed Q_i values were equal to 1 in our case, therefore $q = 0$. Since it is out of scope for the current work, we do not include a fault model in our analysis. For classic CAN frames, C_{D_i} is computed considering the nominal and data bit times $\tau_{bit} = \tau_{dbit} = 2\mu s$, while for the 2Mbps CAN-FD frames we employed $\tau_{dbit} = 500ns$. Further, w_i is obtained by solving the recurrence:

$$w_i^{n+1} = B_i + \sum_{k<i} \left\lceil \frac{w_i^n + J_k + \tau_{bit}}{P_k} \right\rceil C_{D_k} \tag{4}$$

Here, B_i is the maximum blocking time caused by a message with lower priority. Figure 12 shows a 3D plot of the WCRTs computed over the original trace compared with those obtained after adding the CanTSyn frames for the case of 16 subordinates. As expected, the increase of the response time is proportional with the cycle time and inversely proportional with the priority of the ID. For most IDs, the added delay is below 1-2ms and only for low-priority IDs it tops around 8ms. With optimal traffic scheduling, the response times can be reduced to 0 since the bus load is below 40%, but a complete analysis goes beyond the scope of this work.

V. Conclusion

Secure and highly accurate clock synchronization can be easily achieved with few modifications to the AUTOSAR CanTSyn protocol. Out of the four algorithms that we tested, we conclude that the averaging-based methods perform better. When double precision floating-point numbers are available, the continuous averaging performs best, but with single precision floats the accuracy declines quickly, making the windowed average approach the preferred option. Clearly, statically applied clock rate corrections provide better resilience against sporadic noise and jitter. Nevertheless, our results show that employing the DMA for timestamp gathering yields similar performance results, i.e., in the range of 200ns, making this mechanism feasible for systems in which high accuracy is demanded and CPU interrupts are not an option.

References

[1] "Specification of Time Synchronization over CAN," AUTOSAR, Munich, Germany, Standard, Nov. 2023. [Online]. Available: https://www.autosar.org/fileadmin/standards/R23-11/CP/AUTOSAR_CP_SWS_TimeSyncOverCAN.pdf

[2] M. Gergeleit and H. Streich, "Implementing a distributed high-resolution real-time clock using the can-bus," 1994.

[3] D. Lee and G. Allan, "Fault-tolerant clock synchronisation with microsecond-precision for can networked systems," in *Proceedings of the 9th International CAN Conference, Munich, Germany*, 2003.

[4] M. Akpinar, K. W. Schmidt, and E. G. Schmidt, "Improved clock synchronization algorithms for the controller area network (can)," in *28th Intl. Conference on Computer Communication and Networks*, 2019, pp. 1–8.

[5] M. Akpınar, E. G. Schmidt, and K. Werner Schmidt, "Drift correction for the software-based clock synchronization on controller area network," in *IEEE Symposium on Computers and Communications*, 2020, pp. 1–6.

[6] M. Akpınar, E. G. Schmidt, and K. W. Schmidt, "Highly accurate clock synchronization with drift correction for the controller area network," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 4071–4082, 2022.

[7] S. Einspieler, N. Rathakrishnan, A. Prabhakara, B. Steinwender, and W. Elmenreich, "High accuracy software-based clock synchronization over can," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 52, no. 7, pp. 4438–4446, 2022.

[8] F. Luckinger and T. Sauter, "Software-based autosar-compliant precision clock synchronization over can," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 10, pp. 7341–7350, 2022.

[9] IEEE, "Ieee standard for a precision clock synchronization protocol for networked measurement and control systems," pp. 1–499, 2020.

[10] Y. S. Do, M. Ho Kim, and J. W. Jeon, "Time synchronization method between can-fd nodes," in *2021 IEEE Region 10 Symposium (TENSYMP)*, 2021, pp. 1–5.

[11] S. B. Oh, Y. S. Do, and J. W. Jeon, "The time synchronization of can-fd and ethernet for zonal e/e architecture," in *Intl. Technical Conference on Circuits/Systems, Computers, and Communications*, 2023, pp. 1–5.

[12] T. Fuehrer, B. Mueller, F. Hartwich, and R. Hugel, "Time triggered can (ttcan)," *SAE transactions*, pp. 143–149, 2001.

[13] 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.

[14] F. C. Carvalho and C. E. Pereira, "A runtime stability analysis of clock synchronization precision on a time-triggered bus prototype," *Sba: Controle & Automação Sociedade Brasileira de Automatica*, vol. 20, pp. 45–52, 2009.

[15] M. Akpınar and K. W. Schmidt, "Predictable timestamping for the controller area network: Evaluation and effect on clock synchronization accuracy," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 54, no. 3, pp. 1926–1935, 2024.

[16] K.-T. Cho and K. G. Shin, "Fingerprinting electronic control units for vehicle intrusion detection," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 911–927.

[17] S. U. Sagong, X. Ying, A. Clark, L. Bushnell, and R. Poovendran, "Cloaking the clock: Emulating clock skew in controller area networks," in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, 2018, pp. 32–42.

[18] "Specification of Synchronized Time-Base Manager," AUTOSAR, Munich, Germany, Standard, Nov. 2023.

[19] "Specification of Secure Onboard Communication," AUTOSAR, Munich, Germany, Standard, Nov. 2023.

[20] A. Armando, W. Arsac, T. Avanesov *et al.*, "The AVANTSSAR platform for the automated validation of trust and security of service-oriented architectures," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2012, pp. 267–282.

[21] L. Popa, A. Berdich, and B. Groza, "Cartwin—development of a digital twin for a real-world in-vehicle can network," *Applied Sciences*, vol. 13, no. 1, 2023.

[22] "Ieee standard for floating-point arithmetic," *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pp. 1–84, 2019.

[23] O. Ikumapayi, H. Olufowobi, J. Daily, T. Hu, I. C. Bertolotti, and G. Bloom, "Canasta: Controller area network authentication schedulability timing analysis," *IEEE Transactions on Vehicular Technology*, vol. 72, no. 8, pp. 10 024–10 036, 2023.

[24] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien, “Controller area network (can) schedulability analysis: Refuted, revisited and revised,” *Real-Time Systems*, vol. 35, pp. 239–272, 2007.