# Improved Clock Synchronization Start-Up Time for Switched Ethernet-Based In-Vehicle Networks

Aboubacar Diarra
Robert Bosch GmbH
Email: aboubacar.diarra2@de.bosch.com

Thomas Hogenmueller
Robert Bosch GmbH
Email: thomas.hogenmueller@de.bosch.com

Armin Zimmermann
Ilmenau University of Technology, Ilmenau, Germany
Email: armin.zimmermann@tu-ilmenau.de

Andreas Grzemba
Deggendorf Institut of Technology, Deggendorf, Germany
Email: a.grzemba@th-deg.de

Umair Asrar Khan
Ilmenau University of Technology, Ilmenau, Germany
Email: umair-asrar.khan@tu-ilmenau.de

***Abstract*—Clock synchronization is a necessary feature in in-vehicle time- and safety-critical networks. The reason is that exact timing information has to be maintained in nodes, for instance for time stamps of messages that are important for safety-critical functions. A common time base has to be established at start-up time before time-sensitive data handling can begin. In vehicular applications, this clock initialization phase is only one part of tasks that have to be finished before the vehicle is fully functional. To avoid lengthy delays after vehicle ignition, this start-up phase has to be very short depending on the requirements of functions. Advanced driver assistance systems require high-bandwidth communication such as Ethernet Audio Video Bridging (AVB). Its corresponding IEEE 802.1 AS time synchronization protocol is, however, not able to guarantee fast synchronization start-up. This paper thus proposes and analyses several variants of the time synchronization protocol to achieve a shorter start-up time. Simulation-based performance analysis shows that start-up times can be reduced by a factor of 40 compared to the standard IEEE 802.1 AS with the proposal, without considerably affecting synchronization error.**

## I. Introduction

One challenge in designing a time-sensitive system is the establishment of a common time base. This is because communicating devices in a distributed network have different views of time because they have generally different clock characteristics (frequency drift, granularity etc.) with different initial times. This general problem is well-known in the literature for distributed systems [1], [2].

In the special case of automotive time- and safety-critical applications, especially with requirements aiming to guarantee a very safe car driving, there are hard real-time requirements on time stamp accuracy. Electronic control units (ECUs) need to have the same view of time.

Clock synchronization methods are a part of standard automotive communication networks and protocols such as FlexRay [3], [4]. To support advanced driver assistance systems with their higher bandwidth demands, other communication protocols such as Ethernet AVB are considered recently [5]. However, as this protocol is intended for audio/video streaming [6], it has not been designed for applications with strict real-time requirements. The technology enables simultaneous playback of an audio or video stream by several devices. There is an obvious need for clock synchronization in the concurrent playback devices, which is implemented according to the IEEE 802.1 AS protocol to establish a common time base in the Ethernet-based infotainment network. However, its accuracy requirements are not as strict as required for safety-critical real-time applications, and specifically, its initial setup is not critical.

This paper focuses on this second aspect and shows how clock synchronization can be sped up by a factor of up to 40. The main contribution is a proposal of a set of methods based on IEEE 802.1 AS to reduce synchronization start-up time. The results are validated and the performance compared through simulation. It is noticeable that the standard is basically intended to be set of a layer 2 mechanisms. However, this specification does not exclude the possibility of having the standard implemented in layer 1 where improvements may be needed.

The paper is organized as follows: The subsequent section presents an automotive network architecture that has been proposed earlier for future advanced driver assistance applications [7], and motivates the general requirement of clock synchronization. Related work on clock synchronization is analysed in Section III. The next section gives an overview of the standard IEEE 802.1 AS time synchronization mechanism. Section V introduces ideas towards a shorter synchronization start-up time and gives an overview of their potential integration variants. The resulting variants are presented in the subsequent section, together with a theoretical analysis, before Section VII presents validating simulation results. Finally, the study is summarized and further questions on the topic are raised.

## II. An Example Architecture

An example of an advanced driver assistance system (ADAS) application is road situation interpretation for (semi-) automated driving. For this application, typical sensors include camera, radar and lidar (Light Detection and Ranging) devices, which need to send critical messages as well as the time at which they have been recorded. These messages are merged by a processing ECU in charge of generating data for actuators control (e.g. brakes, engine etc.). Figure 1 gives an overview of a generic architecture of such an ADAS introduced in [7]. The sample architecture includes two groups with three sensors each, which are connected via Ethernet to a first switch (or bridge). From there the data is forwarded over a media independent interface (MII) or Ethernet connection to a second switch, which merges the data flows and sends them together with possible control and data packets from infotainment nodes and fall-back ADAS nodes to the right-most node, which carries the actual application processing. More details can be found in [7].

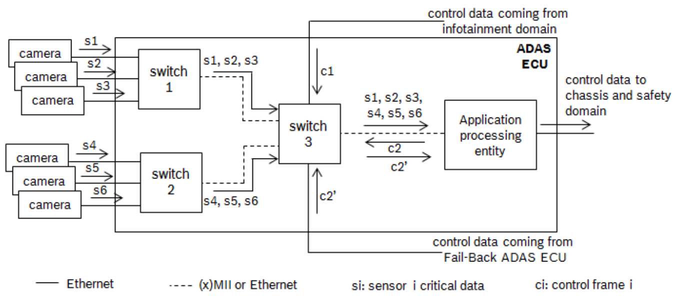
control data coming from infotainment domain
ADAS ECU
camera
camera
camera
s1
s2
s3
switch 1
s1, s2, s3
c1
switch 3
s1, s2, s3, s4, s5, s6
c2
c2'
Application processing entity
control data to chassis and safety domain
camera
camera
camera
s4
s5
s6
switch 2
s4, s5, s6
c2'
control data coming from Fail-Back ADAS ECU
Ethernet
(x)MII or Ethernet
si: sensor i critical data
ci: control frame i

Figure 1: System Communication Architecture

Figure 2 shows a concrete timing behavior and message interleaving in an ADAS network, showing that a missing common time base will lead to errors. In the example, two

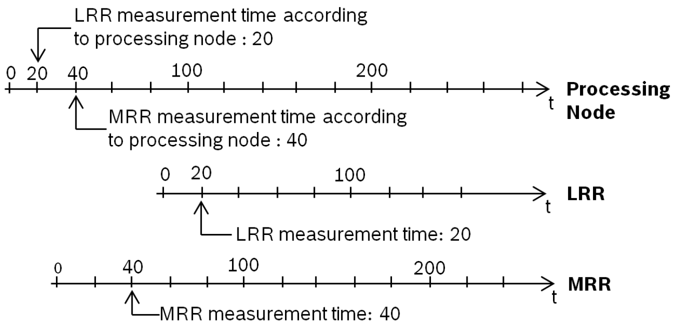
LRR measurement time according to processing node : 20
0 20 40 100 200
t
Processing Node
MRR measurement time according to processing node : 40
0 20 100
t
LRR
LRR measurement time: 20
0 40 100 200
t
MRR
MRR measurement time: 40

Figure 2: Wrong situation interpretation on processing node

radar sensors, one long range (LRR) and one mid-range (MRR) transmit critical messages to a node in charge of processing them. LRR recorded object data at $t = 20$ according to its local clock and MRR did it at $t = 40$ according to its local time source. As processing node, MRR, and LRR all started up at different points in time, there is an offset induced between their different clock time values. Clock frequency drifts add even more to time deviation, but they are not considered in this example. Two kinds of errors are caused when time deviations between different nodes are not compensated. The first problem is a wrong interpretation of data recording times on different sensors by processing node. Object data recording time points according to processing node differ from the actual times they were taken. In Figure 2, $t = 20$ for processing node does not correspond to $t = 20$ for LRR. Similarly, $t = 40$ for processing node does not correspond to $t = 40$ for MRR. The second problem is a wrong classification of different events on processing node. Actually, Figure 2 shows that object data have been in the reality recorded by MRR before LRR did it. However, as recorded times have been taken by each sensor according to its own local clock, processing node understands that LRR recorded its object data before MRR, which is in reality not correct. This wrong event classification can be harmful in time-and safety-critical applications. Therefore, a time synchronization mechanism is needed to compensate clock deviations between participating nodes in the mentioned ADAS especially before any device starts handling critical data for the first time in the network. It is important to ensure that synchronization start-up time is short, which is the time needed to set up a time reference in the whole Ethernet-based in-vehicle network.

## III. Related work

Even though Ethernet AVB has been originally specified for industrial applications, several studies investigated the applicability of IEEE 802.1 AS for in-vehicle networks [6], [3], [8]. Following [9], [10], the maximum synchronization error obtained with IEEE 802.1 AS is $1\,\mu s$ over seven hops in a network. In [11], a test on Broadcom devices confirmed this performance. [12] attested that this performance is guaranteed for slow temperature variations from -10 degrees Celsius (14 F) to 70 degrees Celsius (158 F). Another confirmation based on simulation is presented in [13]. [6] did a similar analysis and observed the same performance for up to 75 % network load.

The mentioned references show that the achieved accuracy of IEEE 802.1 AS protocol is sufficient for in-vehicle networks. Despite its good performance in terms of accuracy, IEEE 802.1 AS does not provide a sufficciently fast synchronization start-up for automotive applications. This is due to the dynamic initialization behavior of the protocol. Moreover, synchronization start-up time was not the main purpose of the protocol specification. The authors of [3] analyzed synchronization start-up time and gave a generic formula, but its improvement is not considered. To the best of the authors' knowledge, there is so far no proposed method to reduce it.

## IV. An Analysis of IEEE 802.1 AS Synchronization Start-Up

IEEE 802.1 AS [9] is based on a variant of the generalized precision time protocol (gPTP, [14]) and is described in this section. It basically consists of a best master clock selection, link delay measurement and time information distribution.

### A. Best Master Clock Selection

The selection of the best (most accurate according to its design parameters) clock among the nodes to serve as the master clock is achieved via the Best Master Clock Algorithm (BMCA). It is based on announce messages exchanged between different nodes in the network to elect the node having the best clock. That node is called the grand master (GM) and propagates time information to establish a common time base in the whole Ethernet-based network. Nodes called slave nodes use this time information to correct their clocks.

## B. Link delay

The link delay ($PathDelay$) is the propagation time of the packet through the cable. It is needed by slave nodes to correct their time via the peer delay mechanism. In Figure 3 a), the slave node sends a $Pdelay\_Req$ message and saves the sending time as $t_1$. The master node saves the corresponding time as $t_2$ upon reception of a $Pdelay\_Req$ message before sending back a $Pdelay\_Resp$ message to the slave node with time $t_2$. This transmission is followed by a $Pdelay\_Resp\_Follow\_Up$ message that contains the transmission time of $Pdelay\_Resp$ ($t_3$). After reception of a $Pdelay\_Resp$ message, the slave node saves the reception time as $t_4$. From $Pdelay\_Resp\_Follow\_Up$ message it gets the information about $t_3$. Now that all four timestamps are known locally, it can calculate the link delay by using the following formula:

$$PathDelay = [(t_4 - t_1) \cdot neighborRateRatio - (t_3 - t_2)]/2$$

$$neighborRateRatio = (t_3 - t'_3)/(t_4 - t'_4)$$

In addition to that, the $neighborRateRatio$ is computed to compensate for frequency drifts between master and slave clocks (c.f. Figure 3 b)). To calculate it, it is necessary for a slave node to have the transmission and reception times of two messages of the same size. The reason is that the neighbor rate ratio is the ratio between two time intervals given by two different clocks. The first interval is the time difference between the transmissions of two frames by the sender, while the second one is the time difference between the receptions of these frames at the receiver. Thus, to evaluate the frequency drift between sender and receiver's clocks, these time intervals have to be identical when there is no drift between the two clocks. This is achieved when the two transmitted and received frames have the same length. Therefore, two consecutive $Sync$ or $Pdelay\_Resp$ messages can be used for the neighbor rate ratio calculation. It is preferable to compute it from two $Sync$ messages because they are transmitted more frequently and synchronization can be done quickly upon start-up. The messages mentioned in this section have been defined in the protocol specifications of the IEEE 802.1 AS standard.

## C. Clock Information Forwarding

The following parameters are introduced here to explain how synchronization information for local clocks is forwarded with IEEE 802.1 AS (termed *time distribution* in the standard):

- $Sync\_to\_FU$: time interval between the reception of a $Sync$ frame and the reception of the corresponding $Follow\_Up$ frame
- $FU\_to\_Sync$: time interval between the reception of a $Follow\_Up$ frame and the reception of the corresponding $Sync$ frame

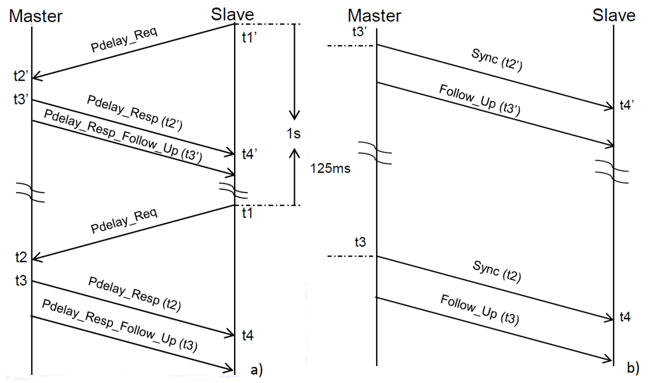

Figure 3: Peer delay mechanism and synchronization messages for neighbor rate ratio calculation

- $resTime$: residence time on a bridge — time interval between the reception of a $Sync$ frame and its forwarding to the subsequent time-aware node
- $PD1$: processing delay 1 on a node — time interval between the reception of a $Follow\_Up$ frame and the transmission of the corresponding $PDelay\_Req$ frame
- $PD2$: processing delay 2 on a bridge — time interval between the reception of the last needed synchronization message to perform clock correction and the forwarding of the $Sync$ frame to the subsequent time-aware node
- $PDelay\_calculation\_time$: time needed to perform the peer delay mechanism
- $P\_Req\_Res\_time$: time interval between the reception of a $PDelay\_Req$ frame and the transmission of the corresponding $PDelay\_Res$ frame
- $P\_Res\_FU\_time$: time interval between the transmission of a $PDelay\_Res$ frame and the transmission of the corresponding $PDelay\_Res\_Follow\_Up$ frame
- $synchInterval$: time interval between the transmission of two consecutive $Sync$ messages
- $current\_time\_Bridge$: current corrected time in the bridge
- $current\_time\_ED$: current corrected time in the end device
- $max\_hops$: the maximum number of hops from GM to end stations
- $d_1$: time interval between reception of the last $Sync$ frame and the calculation of current time in the bridge
- $d_2$: time interval between reception of the last $Sync$ frame and the calculation of current time in the end device

A message sequence for time distribution between three participating time-aware systems (GM, Bridge and end device) is depicted in Figure 4. The GM sends $Sync$ and $Follow\_Up$ messages to all of its ports connected to time-aware systems. The $Sync$ transmission frequency by GM is typically 8 Hz (every 125 ms) according to IEEE 802.1 AS standard. The time at which $Sync$ message has been transmitted ($t_1$) is saved by GM and put in a $Follow\_Up$ message that is transmitted to the bridge node. The $Follow\_Up$ message contains the following information:

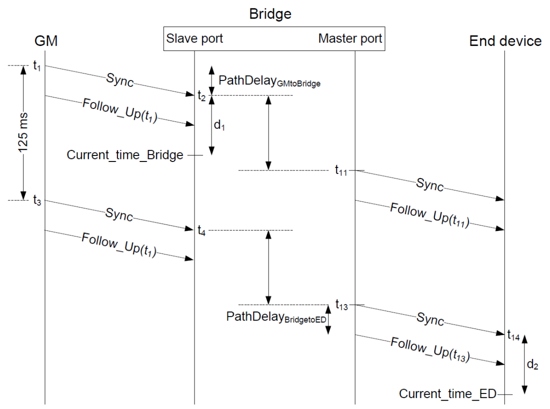

Figure 4: Time distribution between time-aware devices

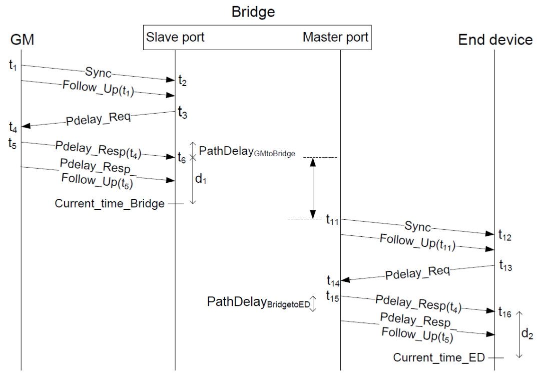

Figure 5: gPTP with $Pdelay\_Resp$ frame size modification

1) **PreciseOringTimeStamp:** This field contains the transmission time ($t_1$) of $Sync$ message according to GM clock.
2) **CorrectionField:** This field contains the sum of path delay and resident time in GM time base. This field is equal to 0 in the $Follow\_Up$ message sent by GM.
3) **RateRatio:** This field contains the cumulative rate ratio between GM and the time-aware system transmitting it. It is equal to 1 in a $Follow\_Up$ message sent by GM.

Upon reception of a $Sync$ message, the bridge saves the corresponding time as $t_2$ (c.f. Figure 4). After receiving $Follow\_Up$ message, it can now correct its clock using the following formula:

$$current\_time\_Bridge = t_1 + PathDelay_{GMtoBridge} + d1 \cdot neighborRateRatio_{GMtoBridge}$$

The bridge forwards then $Sync$ message at time $t_{11}$ and $Follow\_Up$ message to end device (ED). $PreciseOriginTimeStamp$ is transported via $Follow\_Up$ message carrying also $neighborRateRatio_{GMtoBridge}$ and $CorrectionField$ expressed in the following formula:

$$CorrectionField_{Bridge} = pathdelay_{GMtoBridge} + (t_{13} - t_4) \cdot neighborRateRatio_{GMtoBridge}$$

Upon reception of the second $Follow\_Up$ message from bridge, ED will be able to correct its clock time as follow:

$$current\_time\_ED = t_3 + CorrectionField_{Bridge} + PathDelay_{BridgetoED} + d2 \cdot neighborRateRatio_{GMtoBridge} \cdot neighborRateRatio_{BridgetoED}$$

The time distribution mechanism described in Figure 4 is based on three participants. For more than three nodes, it will work in the same way.

### D. An Analysis of Performance and Requirements

A 100 ms synchronization start-up time is normally required for in-vehicle networks based on CAN (Controller Area Network) and FlexRay (see [3]). Meeting the same requirement in an Ethernet-based network is the goal of our study. An Ethernet-based network with five hops is considered in our study. According to IEEE 802.1 AS, the calculation of neighbor rate ratio requires two consecutive $Sync$ messages. These two messages are needed to establish a common time base at network start-up. As IEEE 802.1 AS specified 125 ms between two consecutive $Sync$ messages, it is then clear that the previously mentioned requirement cannot be fulfilled. Therefore, an optimization of the gPTP is imperative to reduce synchronization start-up time in order to meet the constraints. This is the motivating background of this paper to reduce the synchronization start-up time as much as possible without considerably affecting synchronization error.

## V. Techniques for a Shorter Synchronization Start-Up Time

This section introduces methods to shorten the start-up time.

### A. Modification of Sync frame size

In the standard IEEE 802.1 AS synchronization method, a slave node has to wait for the second $Sync$ and $Follow\_Up$ messages to calculate the neighbor rate ratio. This increases the synchronization start-up time. Therefore, the proposed solution here is to adapt the protocol by reducing the number of packets sent from GM to slave to avoid waiting until the next cycle of $Sync$ messages. The GM needs to send two messages of the same size to the slave node then. Bit padding can then be done in $Sync$ frames to make its size equal to $Pdelay\_Resp$ size. In this way, the slave will be able to correct its time after receiving $Pdelay\_Resp\_Follow\_Up$ already (c.f. Figure 5).

The synchronization process starts with the GM sending the $Sync$ message to the subsequent node. The sending time of the $Sync$ message is time-stamped by the GM and inserted in the $PreciseOriginTimestamp$ field of the $Follow\_Up$ message. The bridge stores the time at which the $Sync$ message is received as $t_2$, and sends a $Pdelay\_Req$ message at $t_3$. The GM replies with the $Pdelay\_Req$ reception time $t_4$ in a $Pdelay\_Resp$ packet and sends the $Pdelay\_Resp$ transmit time $t_5$ in a $Pdelay\_Resp\_Follow\_Up$ message. The calculation of the neighbor rate ratios are then different from the standard method. In the bridge, the neighbor rate ratio to GM is (c.f. Figure 5):

$$neighborRateRatio_{GMtoBridge} = (t_5 - t_1)/(t_6 - t_2)$$

In the end device, the neighbor rate ratio to the bridge is:

$$neighborRateRatio_{BridgetoED} = (t_{15} - t_{11})/(t_{16} - t_{12})$$

## B. One-Step Messaging

To further reduce the time to establish a common time base, one-step messaging is proposed. It is similar to the two-step messaging idea except for the fact that no $Follow\_Up$ message is sent after $Sync$ message and no $Pdelay\_Resp\_Follow\_Up$ after $Pdelay\_Resp$. Instead, fast hardware is needed to insert time information into the packet on the fly. Such a hardware time-stamp unit (TSU) can be located either in the media access controller (MAC) or in the physical layer (PHY) of the node performing one-step messaging (c.f. Figure 6).

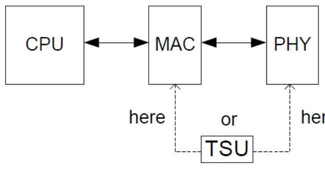

Figure 6: Time-stamp unit location

To perform one-step messaging, $Follow\_Up$ information TLV (Type Length Value) has to be inserted in $Sync$ frame. This increases the size of $Sync$ message to 76 bytes. Second, both timestamps $t_4$ and $t_5$ (c.f. Figure 7) have to be sent in $Pdelay\_Resp$ for rate ratio and path delay calculations. In order to make one-step messaging work properly at start-up time, $Sync$, $Pdelay\_Req$ and $Pdelay\_Resp$ messages should have the same size of 76 Bytes. Bit padding has to be done in $Pdelay\_Req$ and $Pdelay\_Resp$ frames. For the two-step mode, the $Pdelay\_Resp$ frame size is 54 Bytes. As a $Pdelay\_Resp$ frame contains two time information in the one-step mode, its size will be 64 Bytes (a time information consists of 10 Bytes).

Inserting time information on the fly into a synchronization message might cause some synchronization errors when the TSU is not able to generate a time-stamp within its deadline. This time interval depends on the location of the TSU (in the MAC or in the PHY). In Figure 8, when time-stamping is done on the media independent interface (MII) between the MAC and the PHY (TSU is in the MAC), $t\_MAC1$ is the time when the first bit leaves the MAC layer, and time $t\_MAC2$ is the time when the first bit of time-stamp information should be leaving the MAC. This will work if it is inserted in the packet before time $t\_MAC2$. The time-stamp calculation has to be performed within $\Delta t_{MAC}$ thus.

In Figure 9, when time-stamping is done in the PHY (TSU is in the PHY), $t\_PHY1$ is the time when the first bit leaves the PHY layer, while $t\_PHY2$ is the time when the first bit of time-stamp information should be leaving the PHY.

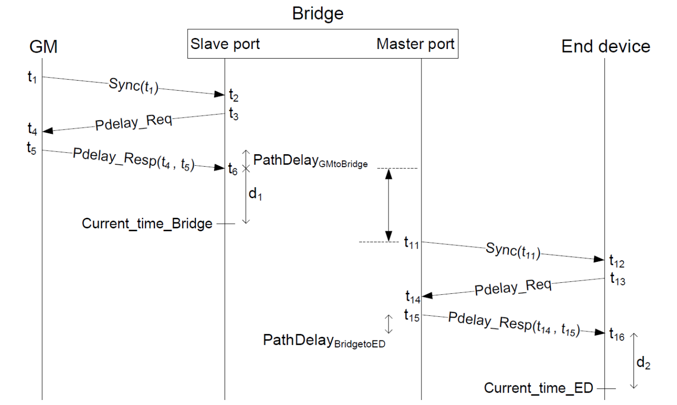

Figure 7: gPTP with one-step messaging and $Pdelay\_Resp$ frame size modification

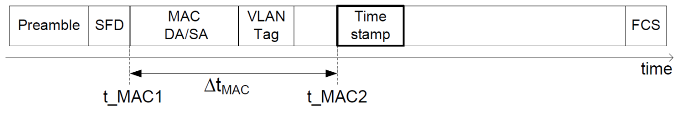

Figure 8: One-step messaging with TSU in MAC

The requirement on using one-step messaging without causing inaccuracies is to insert it in the packet before time $t\_PHY2$. Time-stamp calculations have to be performed within $\Delta t_{PHY}$ therefore.

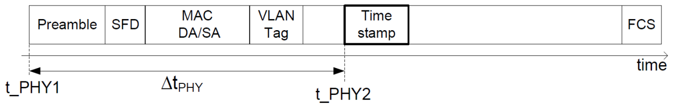

Figure 9: One-step messaging with TSU in PHY

A dedicated hardware should be used for such a one-step messaging because otherwise, if it is done in software, there are chances of unexpected delays due to CPU interruptions. When MII and PHY have the same link speed, we have: $\Delta t_{PHY} > \Delta t_{MAC}$, and thus a solution at the PHY layer has an easier timing constraint than one at the MAC layer.

## C. Peer delay mechanism omission

To further minimize the synchronization start-up time, we suggest to omit the peer delay mechanism at start-up. The necessary information could be reused from the previous active time of the vehicle by storing it in a flash drive, and used for clock deviation correction during start-up. This will be less accurate than automatically measured via the peer delay mechanism, because in addition to the length of wire, the path delay may also depend on many other factors such as temperature, purity and age of conductor etc.

Another idea could be to perform peer delay mechanism during the system design phase, save the measured propagation delay in a flash drive, and use it for time synchronization. If the length of the wire is known, an approximate path delay can be calculated by:

$$PathDelay = cable\_length/propagation\_speed$$

According to IEEE 802.1 AS, the peer delay mechanism has to be performed periodically once per second when the system is running, which would calibrate the value without delaying the initial setup. The previous subsections presented several ideas to shorten the start-up time, which can be combined in different ways presented in Table I.

Table I: Synchronization start-up time reduction methods

| | **Peer delay mechanism, Neighbor rate ratio** | **No peer delay mechanism, no neighbor rate ratio** |
|---|---|---|
| Two-step | Variant A | Variant B |
| One-step | Variant C | Variant D |

## VI. Synchronization Start-Up Methods and Theoretical Analysis

The overall synchronization start-up time obviously depends on the maximum number of hops between a GM and slave nodes for any network architecture. For our target system described in Figure 1, the maximum number of hops is only three, while the maximum number of hops in a general in-vehicle network is usually bigger than that. For a more realistic analysis we consider a network with double-star topology and five hops in the following (c.f. Figure 10). The network represents a backbone architecture of a typical in-vehicle network and is examined for its synchronization start-up time. The results for a smaller network as the one presented earlier in Figure 1 will then be better, i.e., start-up times will be shorter than the results calculated here.

For each variant mentioned above, the synchronization start-up time will be formally analyzed without considering interfering cross traffic during the common time base establishment. For simplification we consider that all links have the same length and that bridge 3 serves as the grand master (GM).

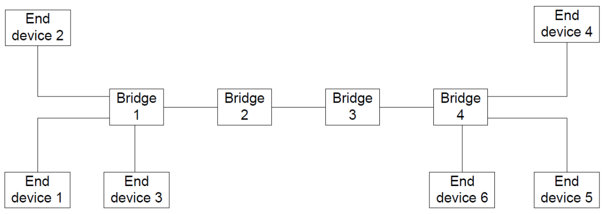

Figure 10: Network overview

### A. Variant A: Two-step messaging with peer delay mechanism and neighbor rate ratio calculation

Time synchronization is initiated by bridge 3 sending its timing information to Bridges 2 and 4 which will further synchronize subsequent connected nodes. The same process is then repeated until the end nodes receive synchronization messages (c.f. Figure 11). The synchronization start-up time then equals the time between the transmission of the first $Sync$ message by the GM and the arrival of the $Pdelay\_Resp\_Follow\_Up$ message at the end device which received it as the latest. For the given topology, it can be formulated as:

$$start\_up\_time = max\_hops \cdot (PathDelay + sync\_to\_FU + PD1 + PDelay\_calculation\_time) + (max\_hops - 1) \cdot PD2$$

$$PDelay\_calculation\_time = 2 \cdot PathDelay + P\_Req\_Res\_time + P\_Res\_FU\_time$$

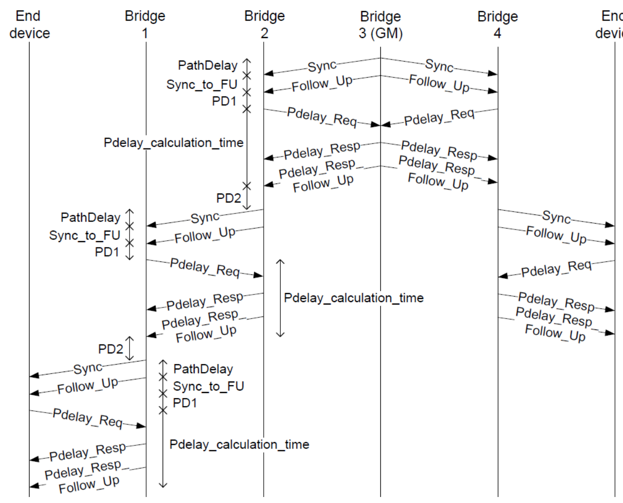

Figure 11: Variant A

### B. Variant B: Two-step messaging without peer delay mechanism and without neighbor rate ratio calculation

The effect of pre-configured path delay for each link is being analyzed here with two-step messaging (c.f. Figure 12). The synchronization start-up time in this case is the time interval between the transmission of the first $Sync$ message by the GM and the arrival of the $Follow\_Up$ at the end device which received it as the latest:

$$start\_up\_time = (max\_hops - 1) \cdot (PathDelay + resTime) + PathDelay + Sync\_to\_FU$$

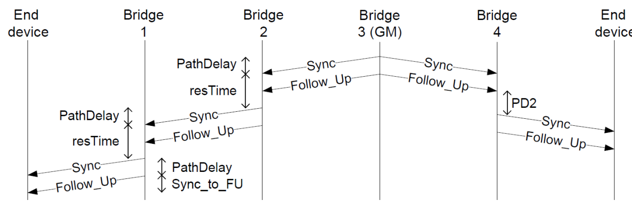

Figure 12: Variant B

### C. Variant C: One-step messaging with peer delay mechanism and neighbor rate ratio calculation

The synchronization start-up time is then the time between the transmission of the first $Sync$ message by the GM and the las arrival of a $Pdelay\_Resp$ message at an end device (c.f. Figure 13):

$$start\_up\_time = max\_hops \cdot (PathDelay + PD1 + PDelay\_calculation\_time) + (max\_hops - 1) \cdot PD2$$
$$PDelay\_calculation\_time = 2 \cdot PathDelay + P\_Req\_Res\_time$$

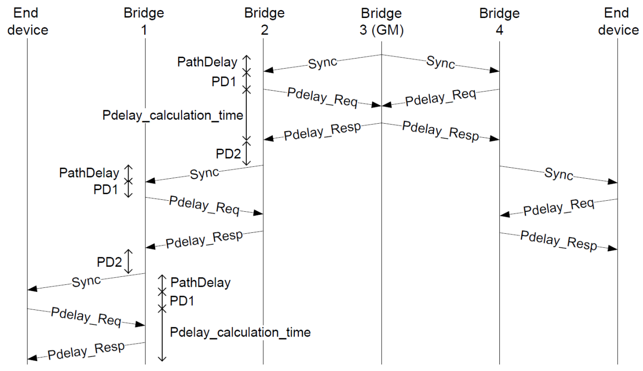

Figure 13: Variant C

### D. Variant D: One-step messaging without Peer delay mechanism and without neighbor rate ratio calculation

This is the simplest and the quickest way to synchronize the nodes at start-up, but also the least accurate (c.f. Figure 14). The resulting start-up time equals the time between the transmission of the first $Sync$ message by the GM and the arrival of the last $Sync$ message at an end device. With the prior knowledge of path delays, and using the one-step method for time synchronization, the start-up time will be further reduced to:

$$start\_up\_time = max\_hops \cdot PathDelay + (max\_hops - 1) \cdot PD2$$

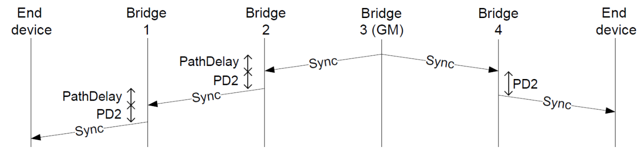

Figure 14: Variant D

## VII. Simulation results

This section is dedicated to the implementation of the IEEE 802.1 AS protocol with different optimization methods in OMNeT++ (see [15]). The INET framework has been extended to model our components. Simulations was carried out for the network topology described in Figure 10.

The assumed parameters (delays) are shown in Table II. The considered link speed is 100 Mbps full-duplex and cable length is 20 meters. During the synchronization start-up period, best effort cross traffic is transmitted. End devices 4, 5, and 6 are the receivers for end devices 1, 2, and 3 respectively (bidirectional). Transmission bandwidth consumption of each end device ranges from 0 to 40 Mbps. Node clock drifts are assumed to be between $1\,\mu s/sec$ and $1.3\,\mu s/sec$. As stated in [16], this is the clock drift range of a typical ordinary quartz oscillator.

Table II: Traffic handling mechanisms

| Delays | values (ms) |
|---|---|
| $Sync\_to\_FU$ | 1 |
| $PD1$ | 2 |
| $P\_Req\_Res\_time$ | 2 |
| $P\_Res\_FU\_time$ | 1 |
| $PD2$ | 4 |

Table III: Synchronization start-up time based on theoretical analysis

| variant A (ms) | variant B (ms) | variant C (ms) | variant D (ms) |
|---|---|---|---|
| 26.0004 | 11.0003 | 20.0009 | 8.0003 |

Table IV: Synchronization start-up time comparison based on simulation

| bandwidth occupation per source (Mbps) | variant A (ms) | variant B (ms) | variant C (ms) | variant D (ms) |
|---|---|---|---|---|
| 0 | 26.063 | 11.024 | 20.074 | 8.024 |
| 5 | 26.092 | 11.04 | 20.114 | 8.03 |
| 10 | 26.132 | 11.058 | 20.114 | 8.041 |
| 20 | 26.194 | 11.075 | 20.195 | 8.064 |
| 30 | 26.207 | 11.086 | 20.251 | 8.081 |
| 40 | 26.246 | 11.111 | 20.311 | 8.12 |

Tables III and IV show that there are minor differences between theoretical calculated synchronization start-up times and the ones obtained in simulation. One reason of the differences is the consideration of cross traffic caused by non time-critical traffic during synchronization start-up phase. That cross traffic is taken into account in the simulation while it is not case in the theoretical analysis. However, the results provided by simulation in absence of cross non-time-critical traffic are not identical to the theoretical ones. Nevertheless the differences due to the inaccuracy of the simulation model are not more than 8 $\mu s$. The performance evaluation results of our simulations are shown in Table IV. The synchronization start-up time reduction variant A provides the maximum value which is around 26 ms. This start-up time is reduced by about 50 % (11 ms) with variant B. This reduction is due to the omission of the peer delay mechanism in the time synchronization process at start-up time. Variants A and B are based on a two-step messaging method and fulfill synchronization start-up time requirements for in-vehicle networks. Variant B probably provides a worse synchronization error than variant A because of the omission of the peer delay mechanism. According to simulation results, synchronization error is $1.7 \cdot 10^{-11}$ seconds after start-up time by omitting the peer delay mechanism, while performing it reduces it to $3 \cdot 10^{-12}$ seconds after start-up time. The synchronization error obtained with variant B is extremely low and much better than required for automotive applications. For this reason, it is in this specific case preferable over variant A because of its lower synchronization start-up time and the lower network load it generates. The choice between variant A and variant B by system designers will be determined by synchronization errors after start-up times. When variant B provides a synchronization error satisfying the requirements, it will be obviously the one to choose. Using variants C and D requires fast hardware time-stamp units. More expensive equipment is thus needed in comparison to variants A and B. Referring to simulation results in Table IV, variant C is less advantageous than variants A and B in terms of relationship between synchronization start-up time and cost. It provides higher synchronization start-up time than variant B, reduces variant A's synchronization start-up time by approximately 6 ms, and requires more expensive equipment. However, its advantage is to reduce network load generated at synchronization start-up time. Variant D reduces the synchronization start-up time by around 75 % compared to variant A, and is 3 ms faster than B. The performance relationship between variants C and D is similar to the one between variants A and B. When the obtained synchronization error by skipping peer delay mechanism meets the requirements defined for an in-vehicle network, the two best variants are B and D. To choose the most appropriate one, a system designer has to compare both methods in terms of start-up time, cost, and network load. Table V gives an overview for both variants in our use-case. Figure 15 compares the synchronization start-up times graphically, which can be reduced by 93 % to 97.5 % with the techniques proposed here.

Table V: Performance Comparison

| | **variant B** | **variant D** |
|---|---|---|
| synch start-up time | very good | very good |
| Hardware costs | lower | higher |
| Network load | higher | lower |

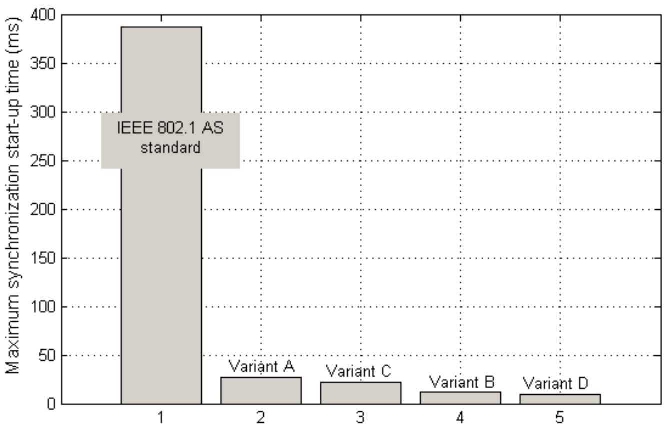

Figure 15: synchronization start-up time comparison between the four variants

## VIII. Conclusion

The paper proposed several ways to reduce synchronization start-up time for future Ethernet-based automotive networks. A theoretical analysis is given and a detailed simulation has been carried out. The fastest method only requires 2.5% of the time compared to the original standard. For a sample network topology with five hops, the highest synchronization start-up time is around 26 ms (about one third of the usually required time in today's CAN-based networks). Simulation showed that the achieved synchronization error is sufficient with the proposed changes. The choice of the best synchronization start-up time method is application-dependent.

## References

[1] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, Jul. 1978.

[2] H. Kopetz and W. Ochsenreiter, "Clock synchronization in distributed real-time systems," *Computers, IEEE Transactions on*, vol. C-36, no. 8, pp. 933–940, Aug 1987.

[3] H. Zinner, J. Noebauer, J. Seitz, and T. Waas, "A comparison of time synchronization in AVB and FlexRay in-vehicle networks," in *Proc. 9th Workshop on Intelligent Solutions in Embedded Systems (WISES)*, July 2011, pp. 67–72.

[4] R. Zurawski, *Networked embedded systems*. CRC press Boca Raton, FL, USA, 2009.

[5] K. Matheus and T. Königseder, *Automotive Ethernet*. Cambridge University Press, 2014.

[6] H.-T. Lim, D. Herrscher, L. Völker, and M. J. Waltl, "IEEE 802.1AS time synchronization in a switched Ethernet based in-car network," in *IEEE Vehicular Networking Conference (VNC)*, Nov 2011, pp. 147–154.

[7] A. Diarra and A. Zimmermann, "System design issues for future in-vehicle Ethernet-based time- and safety-critical networks," in *Proc. IEEE Int. Systems Conference (SysCon 2015)*, Vancouver, Canada, Apr. 2015, accepted for publication.

[8] S. Schneele and F. Geyer, "Comparison of IEEE AVB and AFDX," in *Digital Avionics Systems Conference (DASC), 2012 IEEE/AIAA 31st*, Oct 2012, pp. 7A1–1–7A1–9.

[9] IEEE, *IEEE Std 802.1AS-2011/Cor 1-2013 - IEEE Standard for Local and metropolitan area networks — Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks — Corrigendum 1: Technical and Editorial Corrections*, IEEE Std., Sept 2013.

[10] M. Johas Teener, A. Fredette, C. Boiger, P. Klein, C. Gunther, D. Olsen, and K. Stanton, "Heterogeneous networks for audio and video: Using IEEE 802.1 audio video bridging," *Proceedings of the IEEE*, vol. 101, no. 11, pp. 2339–2354, Nov 2013.

[11] Broadcom, "Ethernet time synchronization," Broadcom, White paper, 2008. [Online]. Available: http://www.broadcom.com/collateral/wp/StrataXGSIV-WP100-R.pdf

[12] A. Kern, H. Zinner, T. Streichert, J. Nobauer, and J. Teich, "Accuracy of Ethernet AVB time synchronization under varying temperature conditions for automotive networks," in *48th ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2011, pp. 597–602.

[13] Y. Liu and C. Yang, "OMNeT based modeling and simulation of the IEEE 1588 PTP clock," in *Int. Conf. on Electrical and Control Engineering (ICECE)*, Sept 2011, pp. 4602–4605.

[14] *Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems (IEEE Std. 1588-2008)*, IEEE Std., 2008.

[15] OMNeT++ simulation tool, version 4.0. [Online]. Available: http://www.omnetpp.org/

[16] N. Brian. (2012) Lecture on distributed systems, time in distributed systems. Aalborg University, Denmark. [Online]. Available: http://people.cs.aau.dk/bnielsen/DS-E08/material/clock.pdf