Integrating Time-Triggered and Event-Triggered Traffic in a Hard Real-Time System

Sascha Einspieler
KAI Kompetenzzentrum Automobil- und Industrie-Elektronik GmbH
Villach, Austria
sascha.einspieler@k-ai.at

Benjamin Steinwender
KAI Kompetenzzentrum Automobil- und Industrie-Elektronik GmbH
Villach, Austria

Wilfried Elmenreich
Networked and Embedded Systems
Alpen-Adria-Universität
Klagenfurt, Austria
wilfried.elmenreich@aau.at

***Abstract*—In industrial communications, there is a high demand for flexibility as well as dependable and timely communication. There are two main paradigms for setting up a distributed real-time system, the time-triggered and the event-triggered approach. While the time-triggered approach is more rigorous and verifiable, the event-triggered approach is more flexible but harder to prove.**

In this paper, we present an approach of a distributed real-time system that follows the time-triggered paradigm but is able to operate in an event-triggered mode when the timing of the particular tasks does not stay within the initial assumptions. The system features an innovative on-the-fly monitoring mechanism based on slot violations. This way, the system stays operable while being able to inform about the mode change and therefore a possible problem.

***Index Terms*—Real-Time, Monitoring, Time-Triggered, Event-Triggered, Distributed System**

I. Introduction

To provide organized bus access in distributed real-time networks, time-triggered communication is mainly used. Thereby, each bus participant is granted bus access during defined time slots. The number of bus participants and the data interface configuration of time-triggered networks need to be defined *a priori* which allows us to create a time-triggered computation and communication schedule of the system. The provided determinism avoids the possibility of collisions on the shared bus. Such systems are usually very static in their configuration and require tasks and communication to fulfill limits on their Worst Case Execution Time (WCET) [1]. With a trend to use more complex software and hardware for embedded systems, determining a hardbound for a task WCET becomes a challenge [2]–[4]: Many state-of-the-art microprocessors have complex caching and clocking strategies that can significantly affect the performance of a single task. Processors that implement dynamic frequency scaling can have thermal effects causing the processor to change to an operation with a lower clock frequency [5], [6].

If it is not possible to avoid hardware with such non-deterministic performance or if a reliable WCET analysis is not feasible, an innovative solution is needed. Thus, we propose an approach with a fallback mechanism to recover from timing violations using an event-triggered communication style. This is in contrast to [7] where the recovery is based on a restart of the time-triggered mode. Further, the proposed approach differs significantly from existing approaches such as FlexRay [8], which implements two separate channels for either communication mechanism. Work by Obermaisser with focus on flexible time-triggered communication [9] uses virtual channels to add event-triggered traffic on a time-triggered system. Thereby, these two modes are tightly integrated with each other at protocol level and switched dynamically. To add a level of determinism to standard IEEE 802.1 and IEEE 802.3 Ethernet networks, a family of standards have been developed by the TSN Task Group [10] since 2012. This novel technology named Time-Sensitive Networking (TSN) applies various mechanisms to separate best-effort and real-time data frames using timing based prioritization.

The proposed system provides the following advantages:

1) It implements a time-triggered approach, allowing rigorous proofs concerning the time-triggered operation,
2) it is able to operate in an event-triggered, best-effort mode to solve problems with unpredicted timing violations,
3) and it comes with a monitoring system notifying an operator how well a system stays within its time-triggered operation or if there already have been some timing violations during system operation.

Possible application domains for the approach lie in real-time systems with the requirement for using state-of-the-art microprocessors with a good average performance but hard to prove WCETs. Examples could be applications with a high computational effort such as image processing or systems requiring an operating system such as embedded Linux.

The remaining paper is organized as follows: In Section II a brief overview of currently available bus setup and monitoring concepts is given. In Section III, the assumptions and requirements for the presented mechanisms are described. Section IV presents the slot violation fallback mechanism. Its operational behavior is shown using various communication scenarios. Based on this mechanism, Section V presents an additional on-the-fly monitoring mechanism. It enables an inherent analysis of the bus communication using a slot and process rating scheme. Section VI discusses the benefits and drawbacks of the presented mechanisms. Section VII concludes the paper and gives an outlook on future work.

II. State of the Art

In time-triggered networks, the possibility of a collision on the shared medium is minimized by granting every bus member access only for distinct time slots. This access scheme, also referred to as Time Division Multiple Access (TDMA), is usually organized by a time schedule which is partially or fully owned by the bus participants [11]. In popular time-triggered protocols like FlexRay [8] or TTP/C [12], the method for setting up the static schedule is named Message Descriptor List (MEDL). Within the MEDL, the TDMA rounds are organized. One round consists of multiple slots where each time slot is assigned to a bus participant. During its slot, the bus member must transmit a data frame that may contain several messages. Multiple rounds form the so-called *cluster cycle*.

As long as the slot boundaries are not violated, a flawless communication can be ensured. However, in case of a slot violation, a transmission overlap is possible which may lead to a data corruption of the following slots. TTP/C prevents this malfunction by use of a bus guardian [12] - i.e. bus members are not allowed to send data outside of their assigned slot to avoid information loss. Some protocol-specific mechanisms provide the possibility to grant preference to the transmission of important data. Such priority mechanism is used, for example, in TTCAN [13] where a message with higher priority interrupts a lower prior message. A retransmission of the interrupted message is not allowed since it may lead to further conflicts and thus, partial information loss. In TSN, best-effort and real-time data frames are separated using TDMA and Virtual-LAN (VLAN) priority slots **IEEE2016b** Each network routing device applies a time-aware scheduler which is responsible to forward real-time data frames at distinct points in time within a cycle. Guard bands are introduced as a mechanism to prevent best-effort frames from extending into the following real-time data slot. Consequently, to compensate the reduced transmission time for best-effort frames, a frame pre-emption **IEEE2016**, **IEEE2016a** mechanism is applied.

The reasons which lead to a slot violation are manifold. They may originate from the firmware or software implementation, the hardware, or the environment. Increased code execution times caused by changing environment or CPU temperatures are mainly attributed to a badly designed TDMA schedule. Also, a bad interrupt or thread handling may lead to unexpected execution times. Even if a WCET analysis is applied, not all scenarios might be covered. Despite this knowledge, the TDMA schedule is mostly designed through an iterative process. As a starting point, the slot widths are set to the expected task's WCET based on an estimation. Afterwards, the widths are adjusted by using a trial-and-error method where the setup is verified online and modified offline. This procedure is repeated until the application requirements (e.g., data update rate) are met or the minimum allowed slot width is reached. Thereby, the minimum allowed slot width $\Delta t_{S_{min}}$ is a function of the tasks execution time Δt_T where $\Delta t_{S_{min}} \geq \Delta t_T$. It is important, that the minimum slot width Δt_S must not undercut the lower task execution time $\Delta t_{T_{min}}$ since this inevitably causes a violation.

To overcome slot violations during run-time, the communication system may be restarted. However, this may not be a proper solution for a variety of applications [7]. Concerning the dynamic properties that lead to such timing violations - e.g. CPU frequency scaling or task switching overhead in modern operating systems - the violation may only be of a short duration and recover on its own.

Previous efforts have been combining event-triggered with time-triggered networks [14], [15]. Here, the event-triggered part is mostly an additional communication channel used to transmit configuration and general data. To monitor and analyze the bus communication, protocols such as [8], [12], [16] require additional external hardware [17]. During normal operation, these tools are not available and thus it is hard to track down possible errors.

III. Prerequisites

This paper presents a fallback mechanism which intends to diminish data loss caused by unpredicted slot violations. In the following subsections assumptions and requirements are presented which are needed to explain the basic functionality of this mechanism. To keep the explanations as simple as possible, a fault-free software and hardware are assumed. Thus, communication fault scenarios due to a wire break or a bad node (babbling idiot) are not considered in this paper.

A. From Messages to Tasks

The designation *message* is generally used to indicate a container which holds some data. A node which puts a message onto the bus distributes the containing information to all the other bus participants. In time-triggered communication, such transmissions start at defined points in time. Thereby, the usual requirement for a flawless communication is the timely reception of messages in their slots. Consequently, the slots must provide the necessary time to transmit their messages. Beside the transmission time also other parameters like the process execution time influence the slot length. This additional delay should be taken into account before the message transmission and is either assumed or obtained by using WCET techniques. However, in most explanations, this is left out or treated as a negligible component.

In the context of this paper, a node does not simply distribute a message. Rather it executes a *Task* T. The task is composed of two parts. The execution of a process P is the first part followed by the transmission of a message M. A graphical representation of such a task is shown in Figure 1. This representation is used for the following explanations.

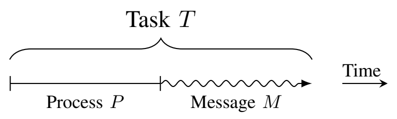

Figure 1. The representation of a task consisting of a process and a message.

As the time passes from left to right the task first invokes the process. The process execution is represented by a straight line enclosed by two vertical bars. Following, the winding line represents the message transmission. The message reception is indicated by the arrowhead. Their lengths qualitatively visualize the necessary time to perform the particular operation.

B. Deterministic Process Execution

Usually, the deterministic property of time-triggered communication relates to the reception time of messages. As long as all messages are received within their slots the communication is denoted to be deterministic. When a slot violation occurs, the communication loses its determinism at that specific point in time. For the realization of the fallback mechanism, this consideration is not useful since slot violations are assumed to be possible. Instead, a deterministic process execution is reasonable. By using this paradigm, the point in time when a process finishes its execution is of interest.

Suppose that a process on node B consumes data from a previously executed process on node A. The provided data consumed by node B is treated to be deterministic as long as the process of node A finishes its execution in its specific slot. This is valid, even if the message reception happened beyond the expected slot and node B hasn't started its process, yet. Hence, if node B starts its process in one of the subsequent slots the received data is valid and usable.

For this reason, the point in time when a specific information was gathered is important for the proposed mechanism and not its message reception time. As long as the subsequent processes are still able to finish in their slots the system remains deterministic.

C. Extended Slot Timing Information

In general, two parameters are sufficient to grant a node access to the shared medium. The first parameter is the global time t_G which is used as the common time base for all bus participants. The second parameter contains the slot starting points $t_S = \{t_{S_0}, t_{S_1} \ldots\}$ which are usually determined in advance during the TDMA slot allocation phase. With this set, each node is aware at which point in time it is allowed to access the shared communication medium. However, this set does not imply enough information to detect a slot violation caused by a previous node. In order to provide the missing information, the slot structure is separated into multiple sections as shown in Figure 2. Due to simplicity, the explanation assumes only one task execution per slot.
The time interval Δt_{S_n} of slot S_n can be determined based on the given slot boundaries:

$$\Delta t_{S_n} = t_{S_{n+1}} - t_{S_n} \text{ with } t_{S_{n+1}} > t_{S_n} \tag{1}$$

The internal slot structure consists of three main sections. These are the task T_n, the starting delay D_n, and the buffer B_n. The task execution time Δt_{T_n} results by

$$\Delta t_{T_n} = \Delta t_{P_n} + \Delta t_{M_n} \tag{2}$$

where Δt_{P_n} is the process execution time of process P_n and Δt_{M_n} is the message transmission time of message M_n.

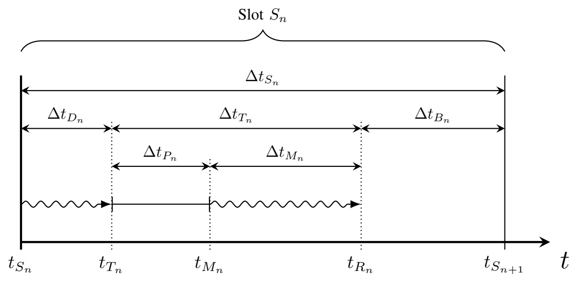

Figure 2. Slot composition including a task, the time segments and the corresponding labels.

The message transmission time Δt_{M_n} depends on the message size s_{M_n} in bits and the used data rate d measured in bits per second:

$$\Delta t_{M_n} = \frac{s_{M_n}}{d} \tag{3}$$

To determine the message sizes, the total number of bits b per message are summed up. The resulting size is a function of the communication protocol $\mathcal{P}$ and the message size in bits:

$$s_{M_n} = \mathcal{P}\Big(\sum_{i=0}^{m} b_i\Big) \tag{4}$$

Since the messages include some protocol-specific overhead the size of the data stream may vary considerably. Consequently, the required time to transmit a message may change if the communication protocol is changed.

The process execution time Δt_{P_n} is influenced by various parameters. These are the local clock speed, the code to process, and the interrupt and/or thread handling. The method, of how the data is acquired mainly influences the number of instructions to process and thus changes the entire execution time. As an example, while the required time for a simple memory access is rather negligible, complex calculations as needed for control loops may have a massive impact.

In general, the message transmission time Δt_{M_n} is static. It can be either determined in advance or derived from the incoming message using additional, protocol specific knowledge. As a result, only the process time Δt_{P_n} dominantly influences the uncertainty of the tasks execution time. However, the process execution time is treated to be unknown and therefore not available to calculate the task execution time. For this reason, Δt_{P_n} needs to be determined indirectly.

The only information a receiving node is able to determine from an executed task T_n is the message reception time t_{R_n}. With this and the message transmission time we are able to calculate the message transmission start

$$t_{M_n} = t_{R_n} - \Delta t_{M_n} \tag{5}$$

which is considered to be identical to the point in time when process P_n finished its execution.

Under normal conditions task T_n would start its execution at the beginning of its slot such that $t_{S_n} = t_{T_n}$. However, since the current slot may be violated by a delayed message of the previous task T_{n-1}, an additional delay $\Delta t_{D_n} \geq 0$ is introduced. This results in

$$t_{T_n} = t_{S_n} + \Delta t_{D_n} \tag{6}$$

Since we are aware of the reception time $t_{R_{n-1}}$ of the previous message M_{n-1} the caused delay can also be calculated with

$$\Delta t_{D_n} = t_{R_{n-1}} - t_{S_n} \tag{7}$$

Now we are able to determine the task's starting point based on the following conditions:

$$t_{T_n} = \begin{cases} t_{S_n} & \text{if } t_{R_{n-1}} \leq t_{S_n} \\ t_{R_{n-1}} & \text{otherwise} \end{cases} \tag{8}$$

The slot ending labeled with $t_{S_{n+1}}$ simultaneously marks the starting point of the subsequent slot S_{n+1}. The time between the slot ending and the message reception is the remaining buffer time $\Delta t_{B_n} = t_{S_{n+1}} - t_{R_n}$. It indicates how much time is left until the slot boundary is reached. If $\Delta t_{B_n} \geq 0$, the task was able to finish in time. Otherwise a slot violation occurred where $-\Delta t_{B_n} = \Delta t_{D_{n+1}}$.

IV. Slot Violation Fallback Mechanism

By using the presented prerequisites we are now able to apply the fallback mechanism. The mechanism extends the time-related task execution check by an additional, event-related validation. The basic functionality is shown in Figure 3. After the start, the first check relates to the validation of the time-triggered condition. Here, the node compares its local time t_G with the starting point of slot S_n. The mechanism checks this

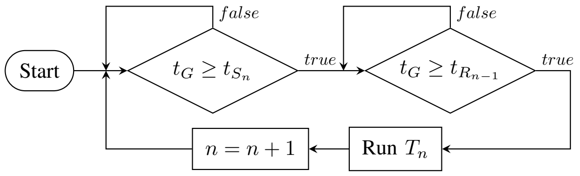

Figure 3. Structure of the fallback mechanism. The left case evaluates the time-triggered requirements whereas the right case evaluates the event-triggered requirement.

condition as long as it returns false. Otherwise, the mechanism advances to the event-related check. Here the arrival time $t_{R_{n-1}}$ of message M_{n-1} is examined. As long as the previous message is pending, the mechanism keeps checking this condition. If this check succeeds, task T_n is allowed to be invoked in its slot S_n. When task T_n terminated its execution, the following slot becomes active and the mechanism repeats.

The event-related check is the core feature of the proposed monitoring mechanism. As long as no slot violation occurs the expected messages are received in time such that the event-based condition will evaluate as true. Thus, the communication is fully time-triggered and the tasks are executed when their slots become active. Only if a delayed message violates the previous slot boundary the execution of the upcoming task is withheld by the event-based check. As soon as the message is received the pending task is executed immediately. Due to this feature, a communication overlap is prevented and thus no data corruption and data loss occur.

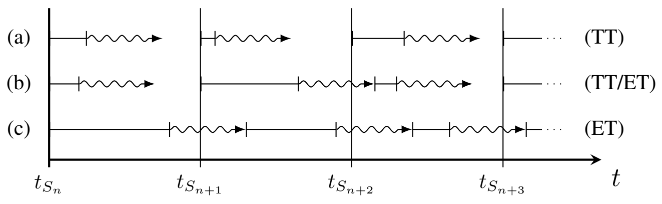

Figure 4. Communication scenarios representing the fallback mechanism.

Figure 4 shows how the described mechanism takes effect. Here, three different communication scenarios are depicted. Scenario *(a)* shows the default, time-triggered bus communication. All tasks are processed quickly and the messages are transmitted in time. Thus, the tasks are allowed to start at their slots starting point. In scenario *(b)* the time-triggered communication is disturbed by a random slot violation. The second task invokes immediately at its slots starting point since it already received the message from the previous task. Unfortunately, it is unable to finish in time which leads to a slot violation. This causes the subsequent node to delay its task since the event-based condition is not fulfilled. Nevertheless, despite this delay, the affected task is able to finish whitin its slot boundaries. This causes the timely task execution in slot S_{n+3} and the bus communication is time-triggered again. In scenario *(c)*, all the tasks need more processing time than expected which leads to multiple slot boundary violations. As a result, all tasks need to be delayed and thus are mainly started by the event-based condition.

As a result, a purely time-triggered communication setup must guarantee that all tasks terminate within their slots. Otherwise, a single slot violation leads to the loss of determinism and data. As shown, the fallback to the event-triggered communication diminishes the loss of determinism and data. The mechanism tries to recover the time-triggered communication by compensating delays within the slots.

However, a scenario as shown in (c) is critical even if all processes seem to terminate within their boundaries. If all tasks in the TDMA round are violating their slots the delay is propagated into the next round. In this case, the communication would quickly lose its determinism within the next few rounds. To overcome this issue, the fallback mechanism could be extended using a fail-silent mode. If a tasks process violates the slot boundary its message is not put onto the bus. Since the subsequent node detects no ongoing transmission at its slot start it is allowed to start its task on the time-triggered basis.

V. On-the-Fly Monitoring Mechanism

As presented, the fallback mechanism is able to reduce data loss caused by slot violations. However, additional information is required to facilitate the setup and verification of the time-triggered communication. Thus, an on-the-fly monitoring mechanism is presented which rates various slot specific qualities. Thereby, the bus communication is analyzed by tracking the message reception times. The gathered data is compared to the known slot boundaries and the quality ratings are determined. As a result, the additional information enables an implicit verification of the communication state which can further be used for an extended node control. Following, three different rating levels are described:

A. Slot Quality Rating

The slot quality rating R_{S_n} is a measure of the closeness between the task termination time and the upper slot boundary. It is determined with the help of the buffer-to-slot ratio $r_{BS_n} = \Delta t_{B_n}/\Delta t_{S_n}$ with $\Delta t_{B_n} \leq \Delta t_{S_n}$. As long as $\Delta t_{B_n} \geq 0$ the task is able to finish in its slot. Otherwise, the slot ratio becomes negative which indicates a slot violation. Based on this, the tasks can be classified within one out of three possible groups which are labeled as *NORM*al, *CRIT*ical, and *FAIL*ed. These groups are mapped onto the slots time-line as shown in Figure 5.

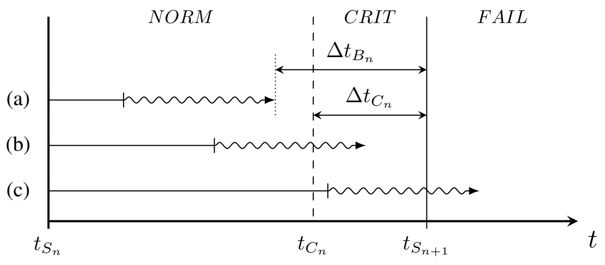

Figure 5. Slot rating using three different quality regions.

The *FAIL* region covers all the tasks which violated their slot boundary. The *NORM* region represents a timely task execution with sufficient space to its upper slot boundary. The *CRIT* region is added in order to provide information about the closeness between the task ending and the upper slot boundary. Thus, an additional boundary t_{C_n} is introduced which marks the critical region's starting point. Its slot ratio is determined with $r_{CS_n} = \Delta t_{C_n}/\Delta t_{S_n}$ where $\Delta t_{C_n} = t_{S_{n+1}} - t_{C_n}$. According to that, the tasks are classified using the following conditions:

$$R_{S_n} = \begin{cases} \textit{NORM} & \text{if } r_{CS_n} > r_{BS_n} \leq 1 \\ \textit{CRIT} & \text{if } 0 \geq r_{BS_n} \leq r_{CS_n} \\ \textit{FAIL} & \text{if } r_{BS_n} < 0 \end{cases} \tag{9}$$

In Figure 5 possible slot rating scenarios are shown. In scenario *(a)*, the slot quality is rated as normal since the task ending stays below the critical boundary t_{C_n}. In scenario *(b)*, the task finishes in the critical region. This still indicates a proper task execution but draws attention since the task ends near the slot boundary. The task shown in scenario *(c)* finishes beyond the upper slot boundary and is therefore rated as failed.

B. Task Quality Rating

The slot quality rating does not consider a slot violation due to a prior delayed task execution. Consequently, a delayed task which violates its slot results in a bad slot rating. This happens even if the task would usually fit into its slot. Therefore, the

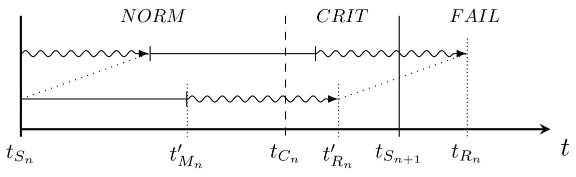

Figure 6. In addition to the slot rating the task rating considers a possible delay and rates the plane task execution time with the slot time.

task quality rating extends the slot quality rating by considering the task execution delay.

To achieve this, the task needs to be virtually shifted to the slot's starting point as shown in Figure 6. Hence, the execution delay is subtracted from the message reception time t_{R_n} to obtain the virtual reception time $t'_{R_n} = t_{R_n} - \Delta t_{D_n}$. The required delay time is calculated with (7) considering the conditions shown in (8). Afterwards, the task is rated by the same mechanism as used for the slot quality rating.

Compared to the slot quality rating the proposed task rating returns a different result if a task execution delay exists. Consequently, if the delay $\Delta t_{D_n} = 0$, both mechanisms return the same result.

C. Process Quality Rating

If a process fits into its slot is an important information since a badly selected process-to-slot ratio ($r_{PS} := \Delta t_{P_n} \cdot \Delta t_{S_n}{}^{-1} > 1$) inevitably leads to the loss of determinism. Therefore, the process quality rating extends V-B. First, in case of an existing delay the entire task is shifted to the slot's beginning. Following, the virtual process termination time is determined as $t'_{M_n} = t'_{R_n} - \Delta t_{M_n}$. Finally, the same procedure as described in V-A is applied. In contrast to the previous ratings, the ratio r_{PS} is computed by using the process execution time which is obtained with $\Delta t_{P_n} = t'_{M_n} - t_{T_n}$ using (5) and (8). With this, the process is assigned to one of the three rating groups.

D. Determinism Check

The process quality rating only yields information about the process-to-slot ratio. Thus, we are still not able to determine if we lost the deterministic property $\mathfrak{D}$ due to a process based slot violation. Therefore, this check provides the missing information using the following condition:

$$\mathfrak{D} = \begin{cases} \textit{1} & \text{if } t'_{M_n} \leq t_{S_{n+1}}\ \forall n \\ \textit{0} & \text{otherwise} \end{cases} \tag{10}$$

As long as the process is able to finish its execution within its slots boundaries ($\mathfrak{D} = 1$) the deterministic property of the real-time network is preserved.

VI. Discussion

The intention of the presented mechanisms is to simplify the setup and validation of a time-triggered bus communication. The fallback mechanism's core functionality is to prevent data loss caused by an unwanted slot violation. To provide this functionality, it uses features from the time-triggered and the event-triggered domain. Thereby, as in time-triggered communication, the bus communication is organized using message scheduling and the message exchange happens on a time-triggered basis. As a result, under normal conditions, the communication mechanism behaves as its purely time-triggered counterpart. Only in case of a slot-violation the event-triggered property becomes active. In such a case, the purely time-triggered communication would lose its determinism immediately. As presented in Section III-B, this does not hold for the fallback mechanism. As long as the process is able to finish in its slot, the fallback mechanism preserves determinism to a certain degree and is therefore *slot-violation tolerant*. Further, depending on the following task execution times, it is able to *recover from a slot-violation* such that the communication is time-triggered again.

In case of a slot violation, the time-triggered scheme is in danger to lose information if two message transmissions overlap. Due to its functionality, the fallback mechanism provides *data-loss reduction* since it avoids such communication based data loss. The loss of information based on severe faults like hardware errors or bad nodes is currently not handled by the basic fallback mechanism. Consequently, it is vulnerable against such error scenarios since the currently active node would simply wait for the missing message and the bus communication would stop. However, this drawback can be resolved by a higher layer which aborts the event-triggered part using a watchdog. After a certain amount of time, the message can be assumed as lost and the subsequent node runs its task. Instead of the expected data either the last or a default value could be used.

A bus monitoring tool is not required, since the proposed on-the-fly monitoring mechanism provides an *inherent communication and execution time monitoring* feature. Thereby, each slot, task, and process is rated at runtime which gives an overview of the state of the entire TDMA round. With this, it is possible to identify tight slot boundaries, violations, and the error causing node. Further, the information can be used to statistically determine the WCET of every single process during runtime and the currently used data rate can be derived. However, to use this feature, the participants need to hold a copy of the entire scheduling list. This list needs to be distributed whenever a new node is added to the bus.

As already mentioned, the required effort to set up a time-triggered communication is cumbersome. With the help of the monitoring feature, this step can be simplified since all required information is available during runtime. Due to the instant bus state feedback the overall communication can be evaluated and verified which leads to a *reduced setup and verification time*. This could also be automated such that the bus members find their slot configuration based on given requirements.

Due to its property of allowing confined slot violations it is not possible to integrate this mechanism into protocols which use bus guardians. Consequently, the fallback mechanism would not work since a delayed transmission would be mistakenly interrupted by the guard.

VII. Conclusion & Outlook

In this paper, a new distributed real-time bus communication scheme paired with an on-the-fly monitoring mechanism has been proposed. It combines time-triggered and event-triggered concepts to keep the bus communication functional even if the slot boundaries are violated by an ongoing message transmission.

The fallback mechanism prevents a possible data loss even if the time-triggered property is lost. The on-the-fly monitoring mechanism rates the single components of a TDMA round to provide detailed information about the overall bus state. In effect, no additional external monitoring hardware is required to verify the proper bus communication during runtime. However, the fallback mechanism is vulnerable to missing messages which can easily be caused by a broken wire or a bad node. To overcome this drawback, the fallback mechanism needs to be extended. Therefore, available approaches like watchdogs will be investigated. Further work comprises the simulation of the proposed mechanisms and the development of a distributed bus communication prototype to investigate the functionality using a small set of nodes. The gathered simulation and prototype results will be published in the future.

Acknowledgment

The authors would like to thank the reliability and product testing department of Infineon Technologies Austria AG for providing an interesting project topic. Fruitful discussions with KAI and Infineon colleagues are also greatly acknowledged.

This work was funded by the Austrian Research Promotion Agency (FFG, Project No. 860424).

References

[1] R. Kirner and P. Puschner, “Classification of WCET Analysis Techniques”, in *Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05)*, IEEE, 2005, pp. 190–199. DOI: 10.1109/ISORC.2005.19.

[2] S. Edgar and A. Burns, “Statistical analysis of WCET for scheduling”, in *Proceedings 22nd IEEE Real-Time Systems Symposium (RTSS 2001) (Cat. No.01PR1420)*, Dec. 2001, pp. 215–224. DOI: 10.1109/REAL.2001.990614.

[3] R. Kirner and P. Puschner, “Transformation of path information for WCET analysis during compilation”, in *Proceedings 13th Euromicro Conference on Real-Time Systems*, IEEE Comput. Soc, 2001, pp. 29–36. DOI: 10.1109/EMRTS.2001.933993.

[4] S. M. Petters, “How much worst case is needed in WCET estimation”, in *2nd International Workshop on Worst Case Execution Time Analysis*, 2002, pp. 111–112.

[5] G. Semeraro, G. Magklis, R. Balasubramonian, D. Albonesi, S. Dwarkadas, and M. Scott, "Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling", in *Proceedings of the 8th International Symposium on High Performance Computer Architecture*, IEEE, Aug. 2002. DOI: 10.1109/HPCA.2002.995696.

[6] E. Le Sueur and G. Heiser, "Dynamic voltage and frequency scaling: The laws of diminishing returns", in *Proceedings of the 2010 International Conference on Power Aware Computing and Systems*, ser. HotPower'10, 2010, pp. 1–8.

[7] W. Steiner and W. Elmenreich, "Automatic Recovery of the TTP/A Sensor/Actuator Network", in *Proceedings of the First Workshop on Intelligent Solutions in Embedded Systems*, 2003, pp. 25–37.

[8] FlexRay, *Communications System Protocol Specification*, Oct. 2010.

[9] R. Obermaisser, P. Peti, and H. Kopetz, "Virtual Networks in an Integrated Time-Triggered Architecture", in *10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems*, IEEE, 2005, pp. 241–253. DOI: 10.1109/WORDS.2005.55.

[10] T.-S. N. T. Group, *Tsn*, Mar. 2018. [Online]. Available: http://www.ieee802.org/1/pages/tsn.html (visited on 03/31/2018).

[11] H. Kopetz, M. Braun, C. Ebner, A. Kruger, D. Millinger, R. Nossal, and A. Schedl, "The design of large real-time systems: The time-triggered approach", in *Proceedings 16th IEEE Real-Time Systems Symposium*, IEEE Comput. Soc. Press, 1995, pp. 182–187. DOI: 10.1109/REAL.1995.495208.

[12] W. Elmenreich and R. Ipp, *Introduction to TTP/C and TTP/A*, 2003.

[13] G. Leen and D. Heffernan, "TTCAN: A new time-triggered controller area network", *Microprocessors and Microsystems*, vol. 26, no. 2, pp. 77–94, 2002.

[14] P. Pedreiras and L. Almeida, "Combining event-triggered and time-triggered traffic in FTT-CAN: Analysis of the asynchronous messaging system", in *Factory Communication Systems, 2000. Proceedings. 2000 IEEE International Workshop on*, IEEE, 2000, pp. 67–75.

[15] L. Almeida, P. Pedreiras, and J. A. G. Fonseca, "The FTT-CAN protocol: Why and how", *IEEE transactions on industrial electronics*, vol. 49, no. 6, pp. 1189–1201, 2002.

[16] C. Watterson, *Controller Area Network (CAN) Implementation Guide*, Application Note AN-1123, 2012. [Online]. Available: http://www.analog.com/media/en/technical-documentation/application-notes/AN-1123.pdf (visited on 11/02/2017).

[17] TTTech, *Monitoring node*, 2018. [Online]. Available: https://www.tttech.com/products/aerospace/development-test-vv/test-hardware/ttp-monitoring-node/ (visited on 01/10/2018).