# Mode-Based Scheduling with Fast Mode-Signaling — A Method for Efficient Usage of Network Time Slots

Tobias Braun, Reinhard Gotzhein, and Thomas Kuhn

***Abstract*—Today, communication in real-time systems is based on time-triggered or event-triggered protocols, or combinations thereof. Each of these solutions has its benefits and tradeoffs regarding timeliness and bandwidth usage. In this work, we present a new method called *mode-based slot scheduling with fast mode-signaling*, which can substantially improve bandwidth usage in many scenarios of industrial relevance, while preserving timeliness. With *mode-based slot scheduling*, a well-controlled amount of dynamic contention for network time slots is possible. *Fast mode-signaling* is used to reach consensus on the current transmission mode with the highest preference extremely fast and reliably among all network nodes. We present the implementation of our method with TTCAN – a time-triggered protocol in the automotive domain –, and evaluation results.**

***Index Terms*—Mode-based slot scheduling, fast mode-signaling, TDMA, slot assignment, TTCAN.**

## I. Introduction

In the real-time systems domain, it is much debated whether to use event-triggered or time-triggered communication protocols. With event-triggered protocols, messages are sent only when events occur, e.g., when a switch is pressed or a threshold is exceeded. With time-triggered protocols, messages are sent at prescheduled points in time, yielding deterministic timing behavior. In case of sporadic traffic, event-triggered protocols exhibit better performance, because bandwidth is only used when needed. However, message delays are less predictable in case of a set of events occurring (almost) simultaneously. In case of periodic traffic, time-triggered protocols are preferable, as they can be configured to match given traffic patterns, yielding predictable maximal message delays. However, extending predictability to non-periodic traffic requires considering worst cases (maximal response times, minimal event intervals), resulting in massive waste of bandwidth as result of the mostly unused, but exclusively reserved (network) time slots.

Consider a typical scenario in a car, where networked components of chassis, power train, safety, comfort, and infotainment compete for communication resources. Due to very different requirements with regard to bandwidth, delay, jitter, and reliability, the automotive industry uses several communication technologies together. E.g., Local Interconnect Network (LIN) [1] supports air conditioning and seat control, Controller Area Network (CAN) [2] connects components of engine control, Media Oriented Systems Transport bus (MOST) [3] is concerned with infotainment, and FlexRay [4] deals with x-by-wire-systems, where *x* stands for safety-relevant functionalities such as steering and braking.

To guarantee predictable message delays and communication bandwidths, precise scheduling and assignment of communication resources is done at development time, i.e. offline. For deterministic guarantees, worst-case situations must be considered. This results in a massive waste of resources in normal situations, which usually require far less communication bandwidth. E.g., to guarantee maximal response times for rare sporadic emergency events, time slots may be assigned on a strictly periodical basis. For these events, reserved time slots are often not used, but due the exclusive reservation policy of the time-triggered paradigm, they can not be used by other nodes.

To exploit these assignments for other purposes when not needed and thereby reduce oversizing of communication systems considerably, we propose a method called *mode-based slot scheduling with fast mode-signaling*, which supports the flexible usage of assigned time slots at runtime. The basic idea of *mode-based slot scheduling* is to permit a well-controlled amount of dynamic, deterministic contention for time slots, based on *transmission modes* (short: *modes*) representing purposes and mode preferences per time slot. For effective operation, this is combined with *fast mode-signaling*, a method to reach network-wide deterministic consensus on the mode with the highest preference reliably and extremely fast.

In Section II, we introduce a simplified car scenario and apply known scheduling techniques. In Section III, we present mode-based slot scheduling. In Section IV, we introduce fast mode-signaling and outline compatibility with TTCAN. Section V presents our prototype implementation of fast mode- signaling for mode-based scheduling for TTCAN. In Section VI, we point out related work. We summarize our findings, draw conclusions, and outline future work in Section VII.

## II. Simplified Car Scenario

To compare event-triggered scheduling, time-triggered scheduling, and combinations thereof with mode-based scheduling, we will consider a simplified car scenario with six nodes $v_1,\ldots,v_6$ sharing a communication bus. Table I lists, for each node, message types, message properties, and communication requirements. In the scenario, node $v_1$ sends periodic entertainment messages, e.g. fragments of an audio

Manuscript received July 8, 2013; revised November 10, 2013.

T. Braun and R. Gotzhein are with the Department of Computer Science, University of Kaiserslautern, 67663 Kaiserslautern, Germany (e-mail: tbraun©cs.uni-kl.de, gotzhein©cs.uni-kl.de).

T. Kuhn is with the Fraunhofer Institute for Experimental Software Engineering IESE, 67663 Kaiserslautern, Germany (e-mail: Thomas.kuhn@iese.fraunhofer.de).

DOI: 10.7763/JACN.2014.V2.81

or video stream, producing relatively high data volume and requiring small and constant transmission delays. Nodes $v_2$ and $v_3$ notify about mirror movements and seat positioning, respectively, which are rare sporadic events requiring bounded transmission delays. Nodes $v_4$ and $v_5$ are responsible for x-by-wire events, which we consider as temporary periodical event streams requiring small and constant transmission delays with high reliability. Finally, node $v_6$ signals airbag events, which are extremely rare, but require very small transmission delays and very high reliability.

We now assume an event-triggered communication bus shared by nodes $v_1,\ldots,v_6$, supporting global priority-based medium arbitration. Furthermore, let each message type be statically associated with a unique priority. These assumptions hold, e.g., for the Controller Area Network (CAN) [2]. Table II shows the priorities of message types, which reflect the requirements of Table I, where smaller values denote higher priorities. This priority assignment suffers from the problem that one of the two x-by-wire event types has always preference over the other (in the example, brake-by-wire over steer-by-wire events). Another problem is that entertainment traffic is dominated by all other traffic, resulting in loss of isochronicity.

TABLE I: MESSAGE TYPES OF THE CAR SCENARIO

| node | message type | message properties | communication requirements |
|---|---|---|---|
| $v_1$ | entertainment | isochronous data stream | small and constant delay, high volume |
| $v_2$ | mirror movement | rare sporadic events | bounded delay |
| $v_3$ | seat positioning | rare sporadic events | bounded delay |
| $v_4$ | steer-by-wire | temporary event streams | small and bounded delay, high reliability |
| $v_5$ | brake-by-wire | temporary event streams | small and bounded delay, high reliability |
| $v_6$ | airbag command | very rare sporadic events | very small and bounded delay, very high reliability |

TABLE II: EVENT-TRIGGERED SCHEDULING

| node | message type | priority |
|---|---|---|
| $v_1$ | entertainment | 10 |
| $v_2$ | mirror movement | 8 |
| $v_3$ | seat positioning | 7 |
| $v_4$ | steer-by-wire | 2 |
| $v_5$ | brake-by-wire | 1 |
| $v_6$ | airbag command | 0 |

Next, we assume a time-triggered communication bus that supports exclusive slot reservations, such as FlexRay [4]. Time is divided into cycles of fixed length consisting of $2^4$-1 slots each. Table III shows, for each message type, the number of slots required per cycle. E.g., entertainment generates high load and therefore needs 8 slots. Seat positioning and mirror movements are rare events, but still need the minimal number of slot assignments, which is one. Steer-by-wire and brake-by-wire related messages need a small and bounded delay; hence we reserve 4 slots for each message type. Finally, airbag commands require a very small and bounded transmission delay and therefore are assigned 2 slots. Adding up, 20 slots per cycle are required in this scenario; however, only 15 slots are available. As a solution, a second communication bus could be installed. However, this would mean massive waste of bandwidth, as slots assigned to rare sporadic events or temporary event streams are often not used.

To improve the situation, we now assume a communication bus that supports both exclusive slot reservations and global priority-based medium arbitration, such as TTCAN [5]. As before, time is divided into cycles of $2^4$-1 slots. As shown in Table IV, slots are assigned exclusively to periodical traffic (entertainment) and temporary event streams (x-by-wire). Rare sporadic events (airbag command etc.) share the same slot(s) and are scheduled according to their priority. To achieve very fast reaction times of airbag commands, 2 slots per cycle are assigned. Adding up, 18 slots per cycle are required, which is a slight improvement over pure time-triggered scheduling, but still needs the installation of a second communication bus.

TABLE III: TIME-TRIGGERED SCHEDULING

| node | message type | slots/cycle |
|---|---|---|
| $v_1$ | entertainment | 8 |
| $v_2$ | mirror movement | 1 |
| $v_3$ | seat positioning | 1 |
| $v_4$ | steer-by-wire | 4 |
| $v_5$ | brake-by-wire | 4 |
| $v_6$ | airbag command | 2 |

TABLE IV: TIME- AND EVENT-TRIGGERED SCHEDULING

| node | message type | priority / slots per cycle (event-triggered) | slots per cycle (time-triggered) |
|---|---|---|---|
| $v_1$ | entertainment | | 8 |
| $v_2$ | mirror movement | 8/1 | |
| $v_3$ | seat positioning | 7/1 | |
| $v_4$ | steer-by-wire | | 4 |
| $v_5$ | brake-by-wire | | 4 |
| $v_6$ | airbag command | 0/2 | |

We will now introduce mode-based slot scheduling, an alternative scheduling technique that, while meeting all hard real-time requirements of the simplified car scenario, needs one communication bus only.

## III. Mode-Based Slot Scheduling

In this section, we introduce and formally define *mode-based slot scheduling*, a global scheduling technique for slots with multiple assignments for communication networks. For this purpose, we define an abstract communication model, which can be instantiated in existing time-triggered communication protocols such as TTCAN or FlexRay.

**Def. 1**: Time is divided into an infinite set of (consecutively numbered) time intervals of equal length $\mathcal{S} = \{S_1, S_2, \ldots\}$ called *macro slots*. Each macro slot is subdivided into a finite set $S = \{s_1,\ldots,s_n\}$ of (consecutively numbered) *micro slots*, which may have different length. Each micro slot has one occurrence per macro slot. The occurrence of micro slot $s_j$ in macro slot $S_i$ is referred to as $s_{i,j}$.

**Def. 2**: A *single-hop network* consists of a finite set $V = \{v_1,\ldots,v_s\}$ of nodes that are pairwise connected by links.

At this point, we consider single-hop networks. This simplifies our model, as we do not have to make connectivity explicit. In Section VI, we will address multi-hop networks, too.

Next, we introduce *transmission modes* (or *modes* for short), a high-level design concept. Modes may be derived from system operation modes, or from transmission purposes.

**Def. 3**: *Transmission modes* are modeled as a non-empty, finite set $M = \{m_1,\ldots,m_r\}$.

As an example, consider the simplified car scenario from Section II. Table V shows message types and their associated transmission modes. E.g., entertainment messages and x-by-wire messages are associated with modes *stream* and *safety*, respectively. In a more refined car scenario, individual messages of the same message type may be associated with different transmission modes in different slots. E.g., to cover critical situations, some x-by-wire messages may also be associated with mode *emergency*.

TABLE V: Modes in the Car Scenario

| node | message type | mode |
|---|---|---|
| $v_1$ | entertainment | stream |
| $v_2$ | mirror movement | regular |
| $v_3$ | seat positioning | regular |
| $v_4$ | steer-by-wire | safety |
| $v_5$ | brake-by-wire | safety |
| $v_6$ | airbag command | emergency |

For each mode, micro slots can be assigned to nodes, which can use these slots to send frames. This means that given $r$ modes $m_1,\ldots,m_r$, a micro slot can be assigned up to $r$ times, possibly to different nodes. If micro slot $s$ is assigned to some node $v$ in mode $m$, this implicitly applies to the occurrences of this micro slot in all macro slots.

**Def. 4**: Let $V$ be a single-hop network, $M$ be a set of modes, and $S$ be a set of micro slots. Then $SA$: $S\times M \rightarrow_p V$ is a partial function called *slot assignment*.

TABLE VI: Slot Assignment in the Car Scenario (4 Micro Slots)

| $SA$ | $s_1$ | $s_2$ | $s_3$ | $s_4$ |
|---|---|---|---|---|
| emergency | $v_6$ | - | $v_6$ | - |
| safety | $v_5$ | $v_4$ | $v_4$ | $v_5$ |
| regular | - | $v_2$ | - | $v_3$ |
| stream | $v_1$ | $v_1$ | $v_1$ | $v_1$ |

Slot assignment is determined *statically* as part of the development process, and defines all bandwidth allocations. If $SA$ is defined for some pair ($s$, $m$), this uniquely defines a single node to which all occurrences of slot $s$ are assigned in mode $m$. If $SA$ is undefined for some pair ($s$, $m$), the occurrences of slot $s$ are not used in mode $m$.

As an example, consider the slot assignment $SA$ of a reduced car scenario shown in Table VI, with 4 micro slots per macro slot only, and 4 modes. Obviously, $SA$ satisfies the constraint that for each mode, a slot is assigned at most once.

As a special case, a slot is assigned exclusively if there is only one mode for which $SA$ is defined. In general, slots may be assigned to several nodes; therefore, the question of how a slot is arbitrated in case of dynamic contention is to be addressed. For mode-based slot scheduling, we resolve contention by introducing, for each slot and mode, unique mode preferences, which determine the priority of messages scheduled in a slot.

**Def. 5**: Let $S$ be a set of micro slots, $M$ a non-empty set of modes, $V$ a single-hop network, $SA$ a slot assignment. The *mode preference* $mp$: $S\times M \rightarrow_p \mathbb{N}_0$ is a partial function, assigning, to each mode and slot for which $SA$ is defined, a unique priority, where lower values represent higher priorities. As an example, consider the mode preferences of the reduced car scenario shown in Table VII. The preferences are chosen such that messages associated with mode *emergency* get the highest priority 0 if scheduled. Messages in mode *safety* get priority 1 if scheduled together with an *emergency* message and the highest priority 0 otherwise. Finally, messages in mode *stream* get the lowest priority, which is 2 in the example.

Here, the question why to distinguish modes and mode preferences may arise. We consider modes to be a high-level design concept, and mode preferences to be a low-level scheduling concept expressing message urgencies. In our opinion, the distinction thus leads to more conceptual clarity.

According to Def. 5, we may, e.g., associate different messages of the same mode and message type with different mode preferences ($mp$) instead of associating message types with unique priorities. This is exploited in the example, where steer-by-wire messages (mode *safety*) have different preferences in slots $s_2$ and $s_3$ (see Tables VI and VII). This flexibility supports the definition of temporal message urgencies, and also the minimization of preference levels per slot, which is crucial for the efficient implementation of mode-based slot scheduling (see Section IV).

TABLE VII: MODE PREFERENCES IN THE CAR SCENARIO (4 MICRO SLOTS)

| *mp* | $s_1$ | $s_2$ | $s_3$ | $s_4$ |
|---|---|---|---|---|
| emergency | 0 | - | 0 | - |
| safety | 1 | 0 | 1 | 0 |
| regular | - | 1 | - | 1 |
| stream | 2 | 2 | 2 | 2 |

In the example, the messages associated with the highest mode preference within a slot may block or delay messages with a lower mode preference (e.g. *safety* messages may be delayed by *emergency* messages; *regular*/*stream* messages can be blocked by these kind of messages for an arbitrary time). However, messages of the modes *emergency* and *safety* are considered to be temporary event streams, and since mirror and seat positioning are sporadic events, a competition for the same slot is not very likely and causes only a slight delay, not restraining the proper functionality of these components. For messages of mode *stream*, the shown schedule assigns the remaining bandwidth (Table VII).

Message types (airbag, x-by-wire) sent by safety-critical components have the highest mode preferences assigned in at least one slot per cycle (slots $s_1$, $s_2$, and $s_4$, respectively). The described schedule is optimized to a minimal number of needed slots. Aspects such as alive messages or QoS constraints can also be considered as outlined in our next example.

**Def. 6**: Let $F$ be a set of frames, i.e. messages to be sent, $\mathfrak{S}$ be a set of macro slots, $S$ be a set of micro slots, $M$ be a set of modes, and $V$ be a single-hop network. Then $FA$: $\mathfrak{S}\times S\times M\times V \rightarrow_p F$ is a partial function defining a *frame assignment*.

TABLE VIII: EXAMPLE OF A FRAME ASSIGNMENT *FA* (4 MICRO SLOTS, MACRO SLOTS $S_1$, $S_2$, $S_3$)

| *FA* | $s_{1,1}$ | $s_{1,2}$ | $s_{1,3}$ | $s_{1,4}$ | $s_{2,1}$ | $s_{2,2}$ | $s_{2,3}$ | $s_{2,4}$ | $s_{3,1}$ | $s_{3,2}$ | $s_{3,3}$ | $s_{3,4}$ | … |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| emergency, $v_6$ | - | | - | | - | | - | | - | | - | | … |
| safety, $v_4$ | | - | - | | | $f_{4,1}$ | - | | | - | - | | … |
| safety, $v_5$ | - | | | - | - | | | $f_{5,1}$ | - | - | | - | … |
| regular, $v_2$ | | - | | | | - | | | | $f_{2,1}$ | | | … |
| regular, $v_3$ | | | | - | | | | $f_{3,1}$ | | | | $f_{3,1}$ | … |
| stream, $v_1$ | $f_{1,1}$ | $f_{1,2}$ | $f_{1,3}$ | $f_{1,4}$ | $f_{1,5}$ | $f_{1,6}$ | $f_{1,7}$ | $f_{1,8}$ | $f_{1,9}$ | $f_{1,10}$ | $f_{1,11}$ | $f_{1,12}$ | … |

TABLE IX: MODE-BASED SLOT SCHEDULING (15 MICRO SLOTS)

| *SA* | $s_0$ | $s_1$ | $s_2$ | $s_3$ | $s_4$ | $s_5$ | $s_6$ | $s_7$ | $s_8$ | $s_9$ | $s_{10}$ | $s_{11}$ | $s_{12}$ | $s_{13}$ | $s_{14}$ | $s_{15}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| emergency | | | | $v_6$ | | | | | | | | $v_6$ | | | | |
| safety | | $v_5$ | $v_4$ | | | $v_5$ | $v_4$ | | | $v_5$ | $v_4$ | | | $v_5$ | $v_4$ | |
| regular | | | | | $v_2$ | | | | | | | | $v_3$ | | | |
| stream | | $v_1$ | | $v_1$ | $v_1$ | | | $v_1$ | $v_1$ | | | | $v_1$ | | $v_1$ | $v_1$ |

A frame assignment determines which frames are scheduled for which slot, mode, and node. In practice, this will be decided *dynamically*, as the frames to be sent are not known offline. E.g., brake-by-wire frames may only be scheduled when the driver uses the brake pedal, thus their occurrence is not deterministic. Also, frames may have to be scheduled more than once, if they compete with other frames associated with a mode of higher preference, or be discarded. In other words, if a frame is scheduled for a given micro slot, this does not imply that it will actually be sent in that slot.

As a constraint, a (dynamic) frame assignment has to respect the (static) slot assignment:

**Def. 7**: Let *SA* be a slot assignment, *FA* be a frame assignment. *FA is consistent with SA* iff $\forall S\in\mathfrak{S}, s\in S, m\in M, v\in V, f\in F.\ (\ FA(S, s, m, v) = f \Rightarrow SA(s, m) = v\ )$.

Table VIII shows a frame assignment *FA* that is consistent with the slot assignment in Table VI. Scheduled frames are represented by an entry $f_{i,j}$, with index 1 corresponding to the node index, and index 2 being a consecutive numbering. Slot assignments without a scheduled frame are represented by an entry "-". Note that node $v_1$ is ready to use all possible bandwidth, whereas the other nodes have only temporary or sporadic needs.

With these preparations, we can now define mode-based slot scheduling as follows:

**Def. 8**: Let $FA$: $\mathfrak{S}\times S\times M\times V \rightarrow_p F$ be a frame assignment, $tx \subseteq \mathfrak{S}\times S\times V\times F$ be a relation defining whether a frame is sent by a node in a given slot occurrence. Then *mode-based slot scheduling* is the strategy where, for each slot occurrence, the scheduled frame with highest mode preference is sent. Formally: $\forall S\in\mathfrak{S}, s\in S, m\in M, v\in V.\ \exists\, f\in F.\ (\ f = FA(S, s, m, v) \wedge \forall m'\in M, v'\in V, f'\in F.\ (m' \neq m \wedge\ f' = FA(S,s,m',v') \wedge mp(s, m) < mp(s,m'))\ \Rightarrow tx(S, s, v, f)\ )$.

From the definitions above, it follows that in each slot occurrence, at most one node can transmit a frame. In the example in Table VIII, shaded entries denote the frames that are actually sent. In some slots, there is dynamic competition among nodes, which is resolved by mode preferences. Nodes

losing competition either reschedule or discard their frames, which are application-specific decisions. In Table VIII, frames in mode *stream* are discarded (e.g., frames $f_{1,5}$, $f_{1,8}$), while frames in mode *regular* are rescheduled (e.g., frame $f_{3,1}$). If there are no frames in mode *emergency*, frames in mode *safety* win competition, so, no rescheduling is required in these cases.

At this point, we revisit our simplified car scenario outlined in Section II. We assume that time is divided into cycles of fixed length consisting of $2^4$ slots each. Slot $s_0$ is not assigned and may be used to implement protocol-specific demands, such as synchronization, which leaves $2^4$-1 slots for assignment. Bandwidth and latency requirements are expressed in slots per cycle (Table III), transmission modes of message types are shown in Table V. Mode preferences are 0 for emergency, safety, and regular where scheduled, and 1 for stream. Table IX shows a feasible slot assignment, with at most two nodes scheduled per slot. All safety critical components have at least one slot per cycle exclusively assigned, enabling the transmission of an alive message once per cycle, if no other events have to be communicated (highlighted in the table). Even for messages assigned to mode *stream*, a minimal bandwidth of 3 slots per cycle is guaranteed. Average bandwidth for mode *stream* could be increased by adding additional slot assignments for non-exclusively used slots with the lowest mode preference 1 (e.g. slots $s_2$, $s_5$, $s_9$). We note that different from the schedules discussed in Sect. II, one communication bus is sufficient when applying mode-based scheduling. Comparing our mode-based schedule with a schedule using exclusive slot reservations, mode-based scheduling reduces the required bandwidth by 25%.

## IV. Fast Mode-Signaling

In this section, we examine different ways to efficiently implement mode-based slot scheduling, introducing an approach called *fast mode-signaling*. We start by discussing conceptual aspects, and then outline a concrete solution with TTCAN [5], a time-triggered protocol from the automotive domain. Further solutions, e.g., with a slightly modified version of FlexRay, are conceivable.

### A. The Concept of Fast Mode-Signaling

By *fast mode-signaling*, we refer to techniques that propagate the current mode with the highest preference fast and reliably through the network. *Fast* means that at the beginning of each time slot or with a very short (negligible) delay, the current transmission mode is known to all nodes. This is a prerequisite for the efficient implementation of mode-based slot scheduling in a distributed system, which requires, for each time slot, that the frame associated with the mode of highest preference be sent.

Conceptually, there are different ways of fast mode-signaling, related to the scope of transmission modes. E.g., the scope can be system-wide, i.e. transmission modes are global, (see, e.g., TTP/C [6]). In this case, only slots associated with the current global transmission mode can be used. Global modes can be signaled, for instance, at the beginning of macro slots, or at the beginning of each micro slot. An advantage is that frames associated with the current mode can directly be sent, without further competition. A disadvantage is that if for a given slot, there is no such frame, this slot is not used.

Another style of fast mode-signaling is to reduce the scope of transmission modes to individual nodes, and then have a well-controlled, highly efficient competition among them. This means that for each slot assigned to a particular node, the node determines its local frame assignments. Given a non-empty set of scheduled frames, the corresponding mode with highest preference is the local mode applicable to the current slot. To send the frame, the node then has to compete with other nodes by signaling its mode in a collision-protected way. As this is in line with mode-based slot scheduling, we will now use it as a basis for outlining a concrete solution. This solution is highly efficient, as no additional overhead is produced.

### B. Fast Mode-Signaling with TTCAN

Today's communication in cars is widely based on the Controller Area Network (CAN) fieldbus technology [2], which marks a breakthrough in modern car manufacturing. CAN applies a CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) scheme for collision-protected medium arbitration. If several nodes start a MAC frame transmission (almost) simultaneously, the node sending the frame of highest priority will always win the arbitration, provided CAN message identifiers are unique.

A property of CAN is that high-priority frames can defer low-priority frames indefinitely. To achieve deterministic operation, a time-triggered version of CAN called TTCAN [5] has been standardized. In TTCAN, time is divided into macro slots called *basic cycles*. Relative to the reception times of a beacon frame (called *reference message*) at the beginning of each macro slot, time slots (called *time windows*) that may have different length are defined. Time windows are *exclusive*, *arbitrating*, or *free*. Exclusive time windows are assigned to one predefined message of a particular node. If the system is properly configured, no contention with other nodes will occur. For arbitrating time windows, which can be used for event-triggered messages, regular CAN arbitration is applied.

Neither mode-based slot scheduling nor fast mode-signaling is addressed in the TTCAN literature (e.g., see [5] and [7]). However, using the mechanisms of TTCAN, it is straightforward to implement fast mode-signaling to support mode-based slot scheduling. Fast mode-signaling can be achieved by applying regular CAN arbitration in exclusive time windows, by associating, with each mode, a CAN message identifier in such a way that mode preferences (see Section III) are respected. To realize mode-based slot scheduling for the slot assignment and mode preferences in Tables VI and VII, respectively, three CAN message identifiers are sufficient. Thus, whenever several nodes attempt to send frames in the same exclusive time window, the frame with highest mode preference will win the contention. Contention in exclusive time windows is feasible, because according to [5], regular CAN arbitration remains active. Thus, our approach does not increase the overhead or reduce the usable bandwidth compared to standard TTCAN.

In comparison to regular TTCAN operation, our approach has two crucial differences. First, contention in each exclusive time window is limited to a well-defined,

controllable number of nodes – up to one node per mode. If an exclusive time window is assigned to one node and one mode only, the time window usage is exclusive, and operation is fully deterministic. Second, CAN message identifiers that encode modes can be used for different messages or message types sent by different nodes, as long as they are sent in different time windows. This would violate regular CAN operation, but is not harmful under mode-based slot scheduling, if each mode is used at most once in each time window, which can easily be checked at configuration time.

To illustrate the differences between pure TTCAN and mode-based slot scheduling, we consider the scenario in Table VI with a total of 4 time windows per basic cycle. Using mode-based slot scheduling, solutions with one TTCAN bus are feasible (see Section III). If time windows are used only exclusively, a total of three TTCAN buses would be required since 12 slots are needed per basic cycle, but only 4 time windows are available in this configuration.

In a combined solution using only one TTCAN bus, exclusive time windows could be assigned to periodic data (mode *stream*), while arbitrating time windows are used for sporadic data (modes *emergency*, *safety*, *regular*), with suitable preferences implemented by assigning CAN identifiers. To obtain some bandwidth for these sporadic data, we assign one time window as arbitrating time window per basic cycle, thereby reducing bandwidth for periodic data. With this arrangement, *emergency* messages compete with *safety* messages, which in turn compete with *regular* messages in the same time windows, and therefore could delay them indefinitely. Even worse, if we assume that there are different safety message types, then according to CAN rules, we would have to use different CAN identifiers. This could result in scenarios where messages of one type always dominate messages of the other one, which would be unacceptable in the x-by-wire scenario.

In summary, solutions with one bus are feasible with mode-based slot scheduling (see Section III) implemented by fast mode-signaling, but not with pure TTCAN if the real-time requirements of the scenario are to be considered.

## V. Implementation

To prove the practicability of fast mode-signaling with TTCAN, we have implemented a prototype using a standard CAN controller together with a software stack realizing fast mode-signaling in conjunction with mode-based scheduling. Our prototype is based on the STM32F407VGT6 microcontroller by STMicroelectronics (Cortex M4) with two integrated CAN controllers [8].

In the first subsection, we will derive requirements on the TTCAN configuration, necessary to enable the implementation of fast mode-signaling for mode-based scheduling. Subsection B describes details of our implementation on the STM32F407VGT6 microcontroller. We evaluate our prototype implementation in Subsection C and discuss our results in Subsection D.

### A. Requirements on the TTCAN Configuration for Implementing Fast Mode-Signaling

In TTCAN, time is divided into basic cycles. Each basic cycle starts with a reference frame, used for the synchronization of all nodes. The basic cycle is further divided into time windows relative to the start of the reference frame. TTCAN supports three types of time windows: *exclusive*, *arbitrating* and *free* windows. We add the type *mode window* for time windows used for mode-based scheduling and derive constraints regarding the minimal required window length. Furthermore, we derive additional constraints regarding the needed synchronization accuracy resp. maximal resynchronization interval, i.e., the maximal length of the basic cycle for realizing a stable working fast mode-signaling.

**Def. 9**: Every mode window has a determined *transmission starting point* (TSP). Nodes with slot assignments for a window start their frame transmission at this point in time according to their local clock. The duration $d_{TSP}$ is defined as duration between the start of the mode window and its TSP (see Fig. 1).

Obviously, to ensure that every node associates the start of a frame transmission to the correct time window, $d_{TSP}$ must be larger than the worst case accuracy achieved by the TTCAN synchronization. The correct association of frame to time window is essential for interpreting and processing the received frame.

**Constraint 1:** Let $accuracy_{worst}$ be an upper bound for the synchronization accuracy achieved by TTCAN (re-) synchronization, then $d_{TSP} \geq accuracy_{worst}$ must hold.

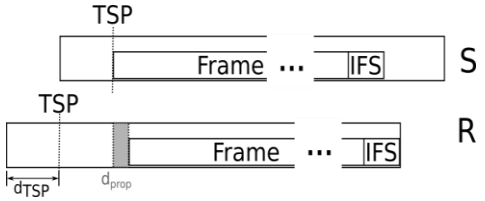


Fig. 1. Frame transmission with maximal deviation and propagation delay.

Fig. 1 shows the structure of a mode window, with $d_{TSP} = accuracy_{worst}$. In this example, sender S and receiver R have a maximal allowed deviation of $accuracy_{worst}$, and the length of the time window is minimal. The propagation delay between sender and receiver is $d_{\mathrm{prop}} = d_{\mathrm{propMax}}$, which is equal to the maximal allowed propagation delay according to the CAN standard for the chosen data rate and bus length (for details see propagation segment in [2] and [9]). To ensure, that transmissions in adjacent time windows are not disturbed, the time window length has to be chosen in a way that a complete frame can be received within the window even if the clocks of sender and receiver clock have the maximal allowed deviation. Increasing the length of $d_{TSP}$ can be used to add extra guard time to make the protocol less susceptible.

**Constraint 2:** Let $d_{frame}$ be the duration of the complete frame transmission including the final interframe spacing (IFS), $d_{propMax}$ the maximal allowed propagation delay and $d_{TSP}$ the length of the transmission starting point duration. For the length of the mode time window $d_{mtw}$, $d_{mtw} \geq 2 \cdot d_{tsp} + d_{propMax} + d_{frame}$ must hold.

Since CAN uses an automatic error signaling mechanism based on overlapping error frames and bit stuffing, $d_{frame}$ must be chosen such that even under worst conditions (bit

stuffing for the original frame and worst case overlapping of error frames) the transmission (including IFS) can be finished within the bounds of the time window. The automatic retransmission of frame has to be disabled. Assuming a CAN frame with 8 byte payload and an overload frame starting at the last bit of the IFS resp. an error frame starting in the 6th bit of the EOF delimiter due a flipping bit together with worst case bit stuffing and overlapping error resp. overload frames may block the bus for maximal 153 bit times.

**Def. 10**: The duration $d_{frame}$ is the duration in number of bit times, required to transfer a standard can frame under worst case conditions (bit stuffing and overlapping error frames) including the final interframe spacing (IFS).

Definition 10 and Constraint 2 are very strict, e.g., by adding $d_{TSP}$ two times to the window length. This ensures that even a receiving node with maximal clock deviation can associate the transmission end with the correct time window. This completely prevents affecting adjacent time windows and their transmissions even under worst case assumptions including erroneous transmissions.

To perform fast mode-signaling and to enable mode based-scheduling with TTCAN, all nodes competing for the same mode window must start transmitting their SOF bit of competing frames at (almost) the same time. The SOF and the subsequent bits of the identifier collide, and the CAN arbitration mechanism is used to solve the conflict. Since the CAN nodes must start their frame transmission based on their local clock, the maximal allowed deviation is limited to guarantee a valid overlapping of the transmission starts and identifier bits.

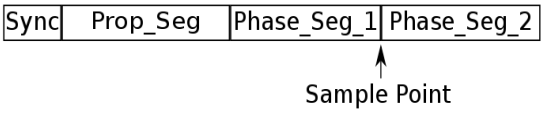


Fig. 2. CAN bit timing and sample point.

Fig. 2 shows the CAN bit timing in more detail. CAN divides a bit time into time quanta grouped into segments [9]. To ensure that the SOF bit and the identifier bits overlap, competitive nodes must receive the transmitted bit of all other competitive nodes in a mode window before their local sample point occurs.

**Constraint 3**: Let $d_{Sync}$, $d_{PropSeg}$ and $d_{PhaseSeg1}$ be the durations of the corresponding segments according to the CAN bit timing definitions. $d_{tq}$ is the duration of a time quantum. For the worst case accuracy $accuracy_{worst}$, it must hold that $accuracy_{worst} < d_{Sync} + d_{PropSeg} + d_{PhaseSeg1} - d_{propMax} - d_{tq}$.

Since $d_{PropSeg} \geq 2 \cdot d_{propMax}$ is a required property for the CAN arbitration mechanism, we can also write constraint 3 as $accuracy_{worst} < 0.5\, d_{PropSeg} + d_{PhaseSeg1}$ (since $d_{Sync} = d_{tq}$). This simplified constraint variant is more strict than Constraint 3, but does not require the knowledge of the maximal propagation delay of the concrete bus. The expression $- d_{tq}$ in Constraint 3 has to be added, since the edge detection of CAN has a worst case delay of one time quantum resulting from the phase shift between the local clocks of two nodes and their discrete sampling at the end of time quanta to detect edges.

The worst case accuracy depends on the accuracy after the synchronization (performed on the reception of the reference frame), the length of the basic cycle (resynchronization interval) and the clock skew of the local nodes. Definition 11 can be used to derive a value for the worst case accuracy $accuracy_{worst}$ and helps to configure a TTCAN bus for fast mode-signaling and mode-based scheduling.

**Def. 11**: Let $d_{cycle}$ be the length of the basic cycle, $d_{maxOffset}$ the maximal deviation after (re-)synchronization, and $r_{maxClockSkew}$ the maximal clock skew of a node's crystal. The worst case accuracy is $accuracy_{worst} \equiv d_{maxOffset} + 2 \cdot r_{maxClockSkew} \cdot d_{cycle}$.

### *B. Implementation of Fast Mode-Signaling for Mode-Based Scheduling*

To validate our assumptions and constraints, we have implemented a prototype using the Cortex M4 microcontroller STM32F407VGT6 by ST. The integrated CAN controller fulfills the requirements of TTCAN level 1, but lacks support for time mark interrupts and a trigger memory [8], [10]. The TTCAN mode of the CAN controller supports the *singleshot* transmission mode, preventing automatic retransmission on transmission errors and lost arbitrations. This mode also supports timestamping of received and transmitted frames on bit time granularity. However, the used timer for creating the timestamps can not be used as interrupt source or to trigger a transmission start.

Therefore, we have implemented the synchronization based on the CAN transmission received interrupt and regular timers to perform time critical, transmission related tasks. The transmission received interrupt is also used to associate a received frame with the corresponding time window. On the successful reception of a reference frame, all nodes reset their local timer; the start of all time windows in the basic cycle is defined relative to this event.

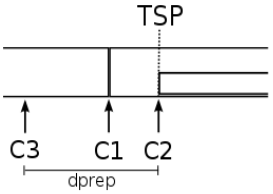


Fig. 3. Frame transmission in mode windows using timer compare interrupts.

Fig. 3 illustrates the process of transmitting a frame in a mode window. Before a frame can be transmitted, the frame has to be transferred to the CAN transmission mailbox, a set of registers managed by the CAN controller for outgoing frames. This step involves the configuration of the identifier of the frame and also the storage of the payload in these registers. The CAN identifier depends on the time window, transmission mode and the associated mode preference. If multiple frames (of one node) are associated with a mode window, the frame with the highest mode preference is selected and loaded into the mailbox. Since these operations require additional runtime, they are performed before the TSP, using a timer compare interrupt (see C3 in Fig. 3). C1 marks the start of the mode window the frame should be transmitted in. The timer compare 2 (C2) interrupt is triggered at the TSP and initiates the transmission of the CAN mailbox by setting the transmission start bit. The duration

$d_{prep}$ between frame preparation and TSP can be configured to reflect the complexity of the setup. Timer compare interrupt C1 is only used to update internal management structures if no frame is going to be transmitted; otherwise these structures are updated after initiating the frame transmission at point C2.

To decouple application logic and time critical frame handling, we have introduced the concept of mode-based transmission buffers. A mode-based transmission buffer consists of a FIFO queue and a static configuration associating the buffer with a transmission mode, a list of mode windows and additional parameters such as the maximal number of retries. The application stores the payload to be transmitted in the corresponding transmission buffer. In the next mode window, the buffer with the highest mode preference associated with this window and containing a frame is selected. This frame is prepared, loaded into the CAN transmission mailbox, and transmitted. If the transmission was not successful, either because of an error or another node transmitting a frame with higher mode preference, the transmission attempt is repeated in the next associated mode window until the maximal numbers of retries has been reached. If the transmission was successful, the frame is removed (using the singleshot semantic) or is retransmitted until the frame is overwritten with a new frame or deleted from the buffer (continuous semantic). Since the described buffer concept is very generic, we also provide similar buffers for exclusive and arbitrating windows.

Incoming frames are stored within a FIFO buffer. Depending on the mode window and the frame identifier, the associated transmission mode is determined and stored together with the payload.

### C. Evaluation

Our evaluation setup consists of 4 STM32F4 development boards with CAN transceivers, $v_1$, $v_2$, $v_3$, and $v_4$. Node $v_1$ transmits the reference frame and works also as sniffer, comparing the expected schedule of frames and their transmission modes with the received sequence of frames and counts divergences.

We are using two different configurations to validate our derived constraints; a CAN bus with 250 kb/s data rate with 14 time windows and a CAN bus with 500 kb/s with 9 time windows. Both configurations use time window 0 as exclusive window for the transmission of the reference frame. Our scenario uses 3 mode preferences and up to 3 competing modes per slot. From 14 resp. 9 time windows, 4 windows are use as mode windows. Tables X and XI are showing the mode preferences and slot assignments for the mode windows of our evaluation scenario for 250 kb/s. To obtain the resp. tables for a data rate of 500 kb/s, replace $s_{13}$ by $s_8$ in table X and XI.

TABLE X: SLOT ASSIGNMENT FOR THE MODE WINDOWS OF OUR EVALUATION

| *SA* | $s_2$ | $s_4$ | $s_6$ | $s_{13}$ |
|---|---|---|---|---|
| *emergency* | $v_3$ | - | - | $v_2$ |
| *safety* | $v_2$ | $v_3$ | $v_4$ | $v_3$ |
| *regular* | - | $v_4$ | $v_2$ | $v_4$ |

TABLE XI: MODE PREFERENCES IN THE EVALUATION SCENARIO

| *mp* | $s_2$ | $s_4$ | $s_6$ | $s_{13}$ |
|---|---|---|---|---|
| *emergency* | 0 | - | - | 0 |
| *safety* | 1 | 0 | 0 | 1 |
| *regular* | - | 1 | 1 | 2 |

During our evaluation, node $v_1$ logged the transmitted frames and calculated the error rate for the actual run. Using a CAN bus with 250 kb/s, we obtained an error rate of 0% while transmitting 250.000 frames. The same experiment has been repeated with 500 kb/s. While during different runs, the error rate for 250 kb/s was stable, the error rate for 500 kb/s was wavering with an average of 37.3%.

By applying Constraint 3, we derive that an accuracy of 2 µs is required for the 250 kb/s CAN bus, whereas the 500 kb/s variant needs at least an accuracy of 0.952 µs. To check these results, we have used a logic analyzer to measure the offset *tickOffset* between all nodes after receiving the reference frame and the deviation *resyncOffset* before a new resynchronization is performed.

Fig. 4 shows the results for both data rates. The bars indicate the measured average tick offset (*tickOffset*) after receiving the reference frame and the average offset before a new synchronization is performed (*resyncOffset*). Every bar has a range indicator attached, indicating the variance between minimal and maximal measured offsets during the experiment. The dashed line resp. dotted line show the calculated needed accuracy for 250 kb/s resp. 500 kb/s based on constraint 3.

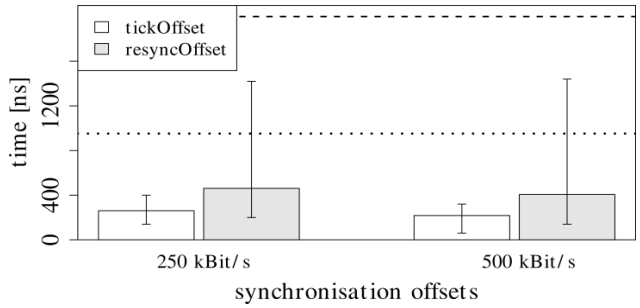


Fig. 4. Measured offsets for 250 kb/s resp. 500 kb/s.

The measured average offsets support our findings of the measured error rate. For a data rate of 250 kb/s, the allowed maximal offset according to constraint 3 (dashed line) has never been exceeded in our measurements resulting in an error rate of 0% in our experiment. In case of a data rate of 500 kb/s, the allowed maximal offset (dotted line) was exceeded several timesby the measured maximal synchronization offset (cf. the range indicator for *resyncOffset* using 500 kb/s).

Remarkable is the effect that the average offsets and even the maximal observed tick offset for 500 kb/s are lower than those for 250 kb/s. This difference could be affected by the different length of the time quanta for both data rates, resulting from the edge detection implementation CAN uses and the phase shift of the nodes (see Constraint 3). The observed delays and jitter, especially regarding the measured tick offset after receiving a reference frame are delays caused

by interrupt handling and processing of incoming frames. We also observed that different optimization options of the compiler affect these offset values, illustrating the problems caused by realizing real-time critical components in pure software without sufficient hardware support.

### D. Discussion

The key requirement to implement fast mode-signaling for mode-based scheduling is an accurate synchronization of all nodes and a minimal clock skew between the nodes. As our evaluation of our prototype implementation shows, a software realization of fast mode-signaling for mode-based scheduling using a standard CAN controller is possible, but limited regarding the maximal achievable data rate. To improve the performance and to realize higher data rates, synchronization and presented buffer structure could be implemented in hardware as part of a specialized TTCAN controller (offering TTCAN level-2 support) with integrated support for fast mode-signaling for mode-based scheduling.

Another way to improve the performance would be to use the upcoming CAN FD standard to transfer mode-based frames [11]. CAN FD supports the usage of two different data rates during frame transmission, one slower data rate for arbitration and acknowledgement and a higher data rate for the transfer of the payload after a single sender has been determined. The payload of CAN FD frames has been increased, so a single frame can transmit up to 64 bytes payload to further reduce the overhead and improve the performance of the protocol compared to standard CAN.

## VI. Related Work

Mode-based slot scheduling with fast mode-signaling has the objective to substantially improve bandwidth usage of TDMA networks in real-time scenarios. This is of particular interest in the automotive domain, where a trend towards time-triggered protocols can be observed. Readily available communication technologies are, e.g., TTCAN [5] and FlexRay [4].

Work that is to some degree related to our approach concerns the construction of feasible and/or optimal message schedules for a given set of messages with real-time requirements. E.g., in [12] and [13], solutions to determine optimal schedules for TTCAN are reported. All approaches assume that access to a time slot is either exclusive (one station) or priority-based (all stations), where given real-time requirements are to be satisfied. The restrictions (exclusive *xor* priority-based access) reduce the search space, such that further improvements of bandwidth usage can not be expected. Another drawback is the priority-based medium contention, where all stations are allowed to participate. This reduces possible real-time guarantees in case of several (almost) simultaneous events. LIN, another time triggered protocol, offers non-exclusive assignments of network time slots to multiple frames (called sporadic frames [1]). Different from our approach, this enhancement is limited to the master node.

TTA (Time-Triggered Architecture [6], [14], [15]) also introduces the concept of different modes in the TDMA-based TTP/C protocol; however, these modes differ in objective and flexibility from our approach. Modes in TTP/C are defined as global *system operating modes* shared by all nodes and not related to our local *transmission modes* defined per slot and assigned message. Even the switching of modes is not compliant with our definition of fast mode-signaling.

Using TTP/C (Time-Triggered Protocol), nodes may request the switching of modes within the header of an arbitrary frame. Nevertheless, even if a node is allowed to change the used operating mode by the message descriptor list, the frame requesting the mode switch will be transmitted and interpreted according to the current active operating mode. As stated in [6], the maximum guaranteed delay interval before a mode change can be activated corresponds to the maximum interval between two network time slots. Since the mode changes are not performed immediately, but at the end of the TDMA round (*Deferred Mode Change*, [14]), a mode change request may be overwritten by subsequent nodes in the current round requesting a different change.

Different from our approach, the operating modes of TTP/C are based upon the same TDMA slot sequence, i.e. the operating mode may change the recipients and the interpretation of the message, but the association between slot and sender remains unchanged [15]. Using multiplexed nodes reduces this problem, but in general does not solve the described drawbacks. With our combined approach of mode-based slot scheduling and fast mode-signaling, we have mitigated these limitations. In particular, we allow for a well-controlled amount of contention while preserving real-time guarantees, if, for all time slots, applicable modes are orthogonal and/or only the slot assignment of highest priority has to satisfy strict real-time requirements. To exploit mode-based slot scheduling practically, we have devised an approach for fast signaling of transmission modes.

## VII. Conclusion

In this paper, we have introduced a method called *mode-based slot scheduling with fast mode-signaling*, supporting the flexible usage of assigned network time slots at runtime. With mode-based slot scheduling, a well-controlled amount of dynamic contention for time slots is permitted. Fast mode-signaling is used to propagate the current system mode extremely fast and reliably through the network. We have outlined an implementation of our method and presented a working prototype for TTCAN, a time-triggered protocol in the automotive domain.

Our approach supports a more efficient usage of communication technologies. For instance, in today's cars, up to 6 field buses of different types are installed. A reduction of the number of field buses by a more efficient and intelligent use of bandwidth would help to reduce cost and weight of cars, which is a topic in the automotive industry.

Mode-based slot scheduling with fast mode-signaling is fully compliant with scheduling strategies in the automotive industry that use (strictly) exclusive and arbitrating time windows only. In fact, these strategies are special cases of mode-based slot scheduling, where it is possible to assign exclusive time windows to at most one node, too. Therefore, current strategies can be used as a starting point to a more efficient usage communication bandwidth, by allowing for well-controlled competition in a suitable subset of exclusive time windows.

So far, we have assumed a single-hop network, which is a valid assumption in case of individual field bus systems. If several field buses are interconnected via gateway nodes, we obtain a multi-hop network, where higher level addressing schemes and routing protocols are needed. To provide deterministic behavior in multi-hop networks, time slots in different field buses can be arranged in a way that messages received by gateway nodes can be forwarded in a timely fashion over the next hop. The challenge here is to find suitable multi-hop mode-based slot schedules; however, the principles are the same as in the single-hop case. Here, fast mode-signaling as described for single-hop networks can be applied in case of wired technologies, too.

In our future work, we will transfer our concept of mode-based slot scheduling with fast mode-signaling to other technologies such as FlexRay and also study its applicability in wireless networked control systems. Furthermore, we are planning to apply our technique to a realistic automotive communication schedule to measure the efficiency of our method regarding the used bandwidth in comparison to established techniques.

## REFERENCES


[1] *LIN Protocol Spec. v2.2, Revision A*, LIN Consortium, Dec. 2012.

[2] K. Etschberger. *Controller Area Network – Basics, Protocols, Chips, and Applications*, IXXAT Press, 2001.

[3] MOST (Media Oriented Systems Transport) Cooperation. [Online]. Available: http://www.mostcooperation.com

[4] *FlexRay Communications System – Protocol Specification v3.0.1*, FlexRay Consortium, 2011.

[5] *Road Vehicles – Controller Area Network (CAN) – Part 4: Time-triggered Communication, ISO 11898-4*, International Standardization Organization, 2004.

[6] H. Kopetz and G. Grünsteidl, "TTP – A Time-Triggered Protocol for Fault-Tolerant Real-Time Systems," in *Proc. FTCS-23, The 23rd Int. Symposium on Fault-Tolerant Computing*, Digest of Papers, 1993.

[7] T. Führer, B. Müller, W. Dieterle, F. Hartwich, R. Hugel, and M. Walther, "Time triggered communication on can (time triggered CAN – TTCAN)," in *Proc. 7th Int. CAN Conf.*, Amsterdam, 2000.

[8] *RM0090: STM32F40xxx, STM32F41xxx, STM32F42xxx, STM32F43xxx advanced ARM-based 32-bit MCUs*, 4.0, STMicroelectronics.

[9] *CAN Specification Version 2.0*, Robert Bosch GmbH, 1991.

[10] F. Hartwich, B. Müller, T. Führer, and R. Hugel, *Timing in the TTCAN Network*, Robert Bosch GmbH, 2002.

[11] *CAN FD Procotol Specification Version 1.0*, Robert Bosch GmbH, April 2012.

[12] M. Naughton and D. Heffernan: "SMART-Plan – A New Message Scheduler for Real-time Control Networks," in *Proc. Irish Signals and Systems Conference*, 2005, pp. 302-307.

[13] K. Schmidt and E. G. Schmidt, "Systematic Message Schedule Construction for Time-Triggered CAN," *IEEE Transactions on Vehicular Technology*, vol. 56, no. 6, part 1, pp. 3431-3441, 2007.

[14] *TTP – Time-Triggered Protocol TTP/C, Protocol Version 1.1*, TTTech, Jan. 2004.

[15] J. Rushby, *A Comparison of Bus Architectures for Safety-Critical Embedded Systems*, SRI Int. Comp. Sci. Lab., CA, 2001.


**Tobias Braun** received his diploma degree in Computer Science from the University of Kaiserslautern in 2009. He is currently member of the Networked Systems Group of the University of Kaiserslautern and works on his PhD thesis. His research interests include real-time communication protocols especially in the context of the automotive domain and the simulation of heterogeneous communication networks.

**Reinhard Gotzhein** is heading the Networked Systems Group at the University of Kaiserslautern. He received his PhD from the University of Erlangen-Nuremberg in 1985. In 1993, he joined the University of Kaiserslautern, where he is holding the full professor position for system software. His areas of interest include networked systems, communication systems, ad-hoc networks, protocol engineering, performance modeling, software reuse technologies, and formal description techniques. Dr. Gotzhein has published about 100 reviewed scientific papers. He has chaired and co-chaired several international and national symposia and workshops, has served on the editorial board and as guest editor of several scientific journals, and is vice-speaker of the research center "Ambient Systems".

**Thomas Kuhn** received his PhD from the University of Kaiserslautern in 2009. Since 2009, he is working at the Fraunhofer Institute for Experimental Software Engineering (IESE). His research interests include medium access control protocols in wireless Ad-Hoc networks, collision resistant transmissions, and model-driven development of embedded systems.