Generalized Asynchronous Time-Triggered Scheduling for FlexRay

Florian Sagstetter, *Member, IEEE*, Martin Lukasiewycz, *Member, IEEE*, and Samarjit Chakraborty, *Senior Member, IEEE*

***Abstract*—FlexRay is a hybrid communication protocol tailored to the requirements in the automotive domain, supporting both time-triggered and event-triggered communication with high data-rates. The time-triggered static segment is commonly used for in-vehicle communication while the event-triggered dynamic segment is used for diagnostics and configuration. This paper addresses the problem of synthesizing schedules for the static FlexRay segment for asynchronous scheduling, following the design approach of current automotive architectures. Previous approaches largely focused on FlexRay 2.1 while the few existing approaches for FlexRay 3.0 are nonoptimal in several aspects. As a remedy, our framework makes use of all new features of version 3.0 while supporting the still predominantly used FlexRay 2.1, making it backward compatible. The following approaches are proposed: 1) a single-stage integer linear programming (ILP) approach that determines an optimal solution but does not scale; 2) a multistage ILP for combining previously generated subsystem schedules to a global schedule. It clearly improves the scalability but is not optimal. The multistage approach allows to integrate and convert existing FlexRay 2.1 schedules into a FlexRay 3.0 schedule which is important for legacy reasons, i.e., to reduce testing and certification efforts; 3) a greedy heuristic which scales well and obtains high quality solutions in comparison with the optimal solution, but is unsuitable to integrate existing schedules; and 4) metaheuristic approaches based on genetic algorithms or simulated annealing to evaluate the benefits of the proposed approaches.**

***Index Terms*—Automotive, FlexRay, schedule synthesis, time-triggered.**

I. Introduction

RECENT years have seen a strong increase in the number of software-based and electronics-based functionality in modern vehicles. Active safety systems such as adaptive cruise control, lane departure warning, or precrash belt pretensioners make vehicles safer and more comfortable [1]. These new functions are implemented on a spatially distributed electrical and electronic architecture with high demands on the in-vehicle network in terms of bandwidth and deterministic behavior.

To overcome the drawbacks of the predominant event-triggered control area network [2] for automotive in-vehicle communication, a consortium of car manufacturers and suppliers developed the FlexRay protocol. It offers a hybrid topology layout with a high bandwidth, supporting time-triggered communication in the static segment, and event-triggered communication in the dynamic segment. With these characteristics, the FlexRay protocol is perfectly suitable for functions with very high safety demands such as drive-by-wire applications [3]. Car manufacturers like Audi or BMW already introduced the FlexRay bus in top-of-the-range series vehicles using version 2.1 of the protocol [4], [5]. However, with the release of the FlexRay 3.0 standard in 2010 [6], the FlexRay consortium introduced major changes, allowing a higher utilization of the bus. At the same time, the complexity of the protocol has increased and as a result new schedule synthesis techniques became necessary.

A. FlexRay Protocol

The FlexRay bus allows data rates of up to 10 Mbit/s per channel and supports both time- and event-triggered message transmission. A FlexRay schedule is organized in a number of communication cycles. While all cycles have the same structure, the messages transmitted in each cycle might change. These cycles are repeated as shown in Fig. 1. A FlexRay communication cycle consists of four different segments as illustrated.

1) A static segment where messages are transferred in a predefined schedule.
2) An optional dynamic segment to transfer less critical and event-triggered data of variable length.
3) An optional symbol window for transferring special symbols, e.g., for bus initialization.
4) The network idle time for the synchronization of network nodes.

The static segment consists of a fixed number of slots of equal size. Each slot has a header, trailer and a payload segment, carrying data between 0 and 254 bytes. By a predefined schedule, the slots are filled with the communication data of applications. Multiple applications might be distributed on electronic control units (ECUs) transmitting messages over the FlexRay bus. The applications exchange

Manuscript received July 27, 2015; revised October 31, 2015 and February 6, 2016; accepted April 28, 2016. Date of publication May 18, 2016; date of current version January 19, 2017. This work was supported by the Singapore National Research Foundation under Its Campus for Research Excellence and Technological Enterprise Programme. This paper was recommended by Associate Editor L. P. Carloni.

F. Sagstetter and M. Lukasiewycz are with the Technical University of Munich Campus for Research Excellence and Technological Enterprise Centre for Electromobility, Singapore (e-mail: florian.sagstetter@tum-create.edu.sg).

S. Chakraborty is with the Technical University of Munich, Germany (e-mail: samarjit@tum.de).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2016.2570421

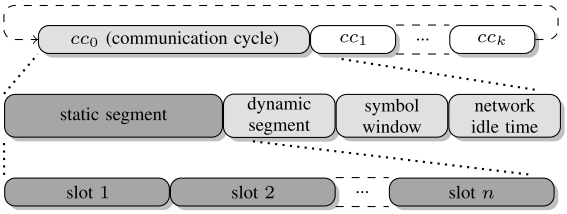


Fig. 1. Basic structure of FlexRay protocol organized in communication cycles with a detailed illustration of the static segment.

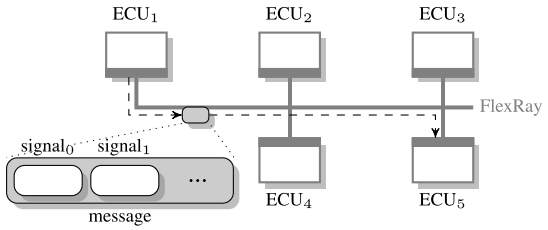


Fig. 2. FlexRay communication of two ECUs. Signals are packed into messages before being transfered over the bus.

signals as shown in Fig. 2. A signal could be physical data like the vehicle speed or a single control instruction. Several signals of an ECU are then packed into a message which is transmitted on the FlexRay bus. Messages are measured in bytes as the smallest unit. A message might only be sent in a predefined time slot. Fig. 3 gives an overview of the data structure of a FlexRay slot and the common terminology. Messages are packed into frames which equal the payload of a slot. Different frames may be transmitted in the same slot at different cycles. FlexRay supports cycle multiplexing such that for each cycle, a slot may contain different messages. In the work at hand, all signals are assumed to be packed into messages. This is a common scenario, since messages are predefined by the ECUs.[1] For FlexRay synthesis generally two approaches exist.

1) A two-stage approach, first packing messages into frames, before packing the frames into slots.
2) A direct packing of messages to slots, implicitly defining the frame packing.

The two-stage approach reduces the problem complexity, while it might lead to suboptimal solutions with respect to the number of required slots. This paper presents a single-stage integer linear programming (ILP) and a greedy heuristic providing a direct packing of messages to slots for a minimal bandwidth requirement. Furthermore, a multistage ILP applying the two-stage approach, supporting the integration of legacy systems, is presented.

B. Contributions of this Paper

This paper presents a general, version-independent, scheduling framework for FlexRay. Various work has been done for FlexRay 2.1 scheduling [7]–[9], however, these approaches are not applicable to FlexRay 3.0. Existing FlexRay 3.0

[1]Note that a direct packing of signals to frames compared to packing of messages to frames has no influence on the bandwidth requirements. A message solely groups signals, but does not introduce an overhead.

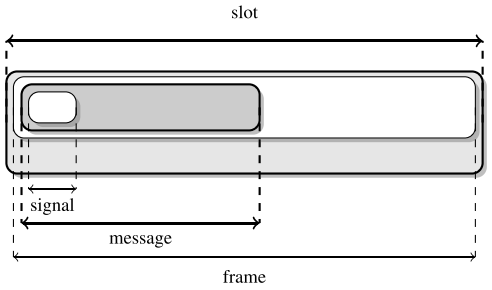


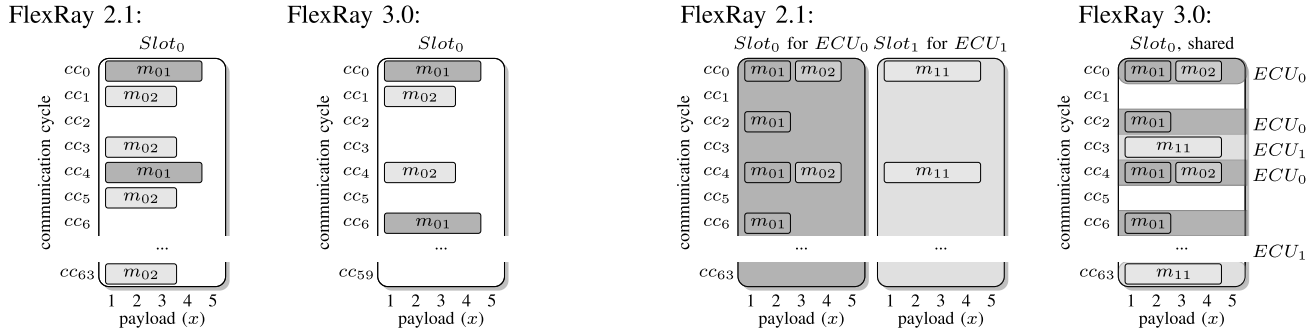
Fig. 3. Basic data structure of a FlexRay schedule. Signals are packed into messages before being sent by an ECU. Messages are sent in a frame assigned to a predefined time slot.

approaches [10], [11] only consider frame to slot or message to frame packing, respectively, and omit a message offset assignment as in [12]. In contrast, all approaches presented here are comprehensive. Our approaches not only obtain a message to slot assignment, but also assign a message offset within the slot, as required by FlexRay.[2] Offset assignment is required to efficiently use all new features of FlexRay 3.0, i.e., a variable cycle number. Furthermore, we propose approaches for a direct message to slot packing to achieve minimal bandwidth utilization compared to a two-stage approach including frame packing. In accordance with the current design approach in the automotive industry, an asynchronous scheduling with the goal of a high bandwidth utilization is considered. For asynchronous scheduling no timing synchronization between the ECUs and the bus is performed. Asynchronous scheduling is usually applied to robust control functions that are not affected negatively by additional delays which is the case for many automotive applications. However, asynchronous scheduling approaches might be extended with a task scheduling to support synchronous scheduling (see [13], [14]). Here, the focus is on message scheduling over the FlexRay bus, taking additional parameters like the message offset and the scalability of the schedule synthesis into account. A detailed discussion of related work is given in Section VII.

The scheduling problem addressed in this paper is the determination of a schedule for a set of messages with the goal of a minimal bandwidth utilization. The schedule defines a slot, base cycle, and offset for each message. Section II gives a detailed problem description and introduces our framework. This paper presents several techniques for scheduling the FlexRay static segment that fulfill different requirements in terms of runtime and quality of results.

1) A single-stage ILP formulation to obtain an optimal schedule, suitable for problems with a moderate size and complexity (Section III).
2) A multistage ILP approach which allows to first determine subsystem schedules before integrating them into a global schedule for the entire network (Section IV). This is in particular relevant for legacy systems as testing

[2]Defining a message offset within a slot is necessary for time-triggered protocols which allow to pack multiple messages in one slot and do not define a message header. A receiving ECU requires the message offset to identify in which part of the slot payload a message is packed.



(a) FlexRay 2.1 limits the cycle number k to 64 while 3.0 allows to customize k.

(b) FlexRay 2.1 requires a slot to be explicitly assigned to one ECU while 3.0 supports multiple ECUs per slot.

Fig. 4. Comparison of version 2.1 and 3.0 of FlexRay. (a) Selecting a suitable cycle number with FlexRay 3.0 might prevent oversampling and save bandwidth. Here, the messages m_{01} and m_{02} might be sent with their initial periods $r_{m_{01}} = 6$ and $r_{m_{02}} = 3$ for $k = 60$ cycles while oversampling is required for $k = 64$ as it is not divisible by 3 or 6. (b) Slot sharing allows to further reduce the bandwidth usage with FlexRay 3.0.

TABLE I
CHARACTERISTICS OF PROPOSED APPROACHES

method	bandwidth usage	scalability	extensibility/ modularity
Single-stage ILP	++[1]	- -	- -
Multi-stage ILP	○[2]	+	++
Greedy Heuristic	++	++	- -
Genetic Algorithm	+[3]	+	- -
Simulated Annealing	+[3]	+	- -

[1] optimal approach, determining minimal bandwidth usage
[2] average bandwidth utilization
[3] scales with runtime

and certification efforts are clearly reduced if an existing configuration is reused, instead of creating a new configuration from scratch. The approach allows to convert or integrate existing FlexRay 2.1 schedules to FlexRay 3.0 schedules, optimizing their bandwidth usage.

3) A greedy heuristic that determines schedules for full state-of-the-art in-vehicle networks, sending up to 1000 messages within few seconds while minimizing the required bandwidth (Section V).
4) To evaluate the approaches, a genetic algorithm (GA) and a simulated annealing (SA) approach were implemented (Sections V-C and V-D). An extensive case study shows the benefits of our approaches compared to GA- and SA-based approaches.

Table I gives an overview of the capabilities of all approaches. All approaches support both FlexRay 2.1 and 3.0, complying with the automotive open system architecture (AUTOSAR) standard used in the automotive industry for all FlexRay implementations [15].

A large case study consisting of 420 test-cases and a realistic test case modeling a full in-vehicle network are presented to give evidence of the efficiency of the proposed scheduling approaches. At the same time, the results show the possible advantages of FlexRay 3.0 over FlexRay 2.1 in terms of the utilization of the bus (Section VI).

II. FLEXRAY SCHEDULING

This section first introduces the general scheduling in the FlexRay static segment and its notation. Second, a depiction of the differences between FlexRay 2.1 and 3.0 is given. Finally, the framework is introduced.

A. FlexRay Parameters

Extending the high-level introduction in Section I, in the following, a general introduction to FlexRay scheduling is given which applies to both FlexRay 2.1 and 3.0. All major car manufacturers follow the AUTOSAR standard for their FlexRay implementations in series-production vehicles. AUTOSAR is developed by an alliance of car manufacturers and suppliers with the goal of defining a standardized automotive software architecture for ECUs. To increase the bandwidth usage of the FlexRay bus, AUTOSAR suggests cycle multiplexing. Cycle multiplexing allows to alternate the messages sent in one slot from cycle to cycle to increase the utilization [see messages m_{01} and m_{02} alternating in Fig. 4(a)].

FlexRay requires the configuration of various parameters. The number of communication cycles k, the cycle duration t_c, the payload size of a static slot l_s in bytes, and the number of available static slots n in one cycle. A message m is defined by its size l_m and its period p_m. To schedule a set of messages $m \in M$, FlexRay requires that all message periods are a multiple of the communication cycle t_c. If this is the case, the message period can be represented by the message repetition $r_m = (p_m/t_c)$. Given a message m with its size l_m and repetition r_m, scheduling is formally defined as follows. To schedule m, FlexRay requires r_m to be a divisor of the number of cycles k to allow a periodic occurrence. If this is not the case, oversampling becomes necessary, i.e., the next smaller value for r_m which is a divisor of k has to be chosen. To schedule a message m, three parameters need to be determined.

1) The slot s in which m is scheduled.
2) The base cycle b_m that indicates the first cycle in which m is scheduled.
3) The x-offset x_m, indicating at which position in the slot the message m is scheduled in the payload of a slot. x_m might be any value between 0 and $l_s - l_m$.

For instance, in Fig. 4(a) m_{02} is scheduled in slot 0 with a base cycle 1 and x-offset 0. Thus, a message m is scheduled

in all cycles cc_i with

$$i = (b_m + r_m \cdot a) \bmod k, a \in \mathbb{N}_0. \quad (1)$$

This scheduling has to be performed such that no messages intersect. Two messages m and $\tilde{m}$ scheduled in the same slot, at the same payload position, do not intersect if they never share the same cycle

$$\forall i, j \in \mathbb{N}_0 : (b_m + r_m \cdot i) \bmod k \neq (b_{\tilde{m}} + r_{\tilde{m}} \cdot j) \bmod k. \quad (2)$$

Schedules for single FlexRay slots are illustrated in Fig. 4(a) and (b). The representation is derived from Fig. 1 through introducing a row for each cycle on the y-axis. Hence, b_m is the offset on the y-axis and x_m is the offset on the x-axis within a slot.

B. FlexRay Versions

The FlexRay 3.0 standard introduces several changes (in comparison to version 2.1) to the FlexRay protocol. In particular for scheduling the static segment, the following two major changes are highly relevant and might improve the utilization of the bus: 1) the number of cycles k is variable and 2) slots might be shared by different ECUs.

1) The number of cycles k of a FlexRay schedule is configured at design time. For FlexRay 3.0, k can be any even number of cycles between 8 and 64. A message might only be scheduled with a repetition r_m that is a common divisor of k, hence $k \bmod r_m = 0$ must hold. As example for $k = 60$, a message m might be sent with repetitions of $r_m \in \{1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30, 60\}$ cycles. In contrast, for version 2.1 of FlexRay, the number of cycles of a FlexRay schedule is fixed to $k = 64$, defining $r_m \in \{1, 2, 4, 8, 16, 32, 64\}$. The variable cycle number of FlexRay 3.0 might allow a higher flexibility in terms of available repetition values. Hence, a lower bandwidth utilization might be achieved by preventing oversampling. Fig. 4(a) illustrates how two messages need to be oversampled for $k = 64$ in comparison to $k = 60$. For this example, scheduling with $k = 60$ leads to over 30% less bandwidth utilization compared to the $k = 64$ of FlexRay 2.1. The unused slot space might then be used to schedule other messages and reduce the total number of slots. However, while $k = 64$ only supports message repetitions which are a multiple of 2, a variable cycle number might allow to send messages with repetitions larger than 1 that only share 1 as greatest common divisor in one slot. Hence, while message offsets might be trivially defined for FlexRay 2.1 by filling a slot from left to right [9], version 3.0 requires a specific offset assignment.[3]
2) Additionally, while for FlexRay 2.1 a slot is exclusively assigned to one ECU to send messages, version 3.0 allows slot sharing between different ECUs. Thus, for each communication cycle, a different ECU might send its messages in a specific slot. Fig. 4(b) shows how three messages sent by two ECUs might be scheduled in a single slot with FlexRay 3.0 while version 2.1 requires two slots to schedule the same messages. While this new feature allows to use slots more efficiently, the problem complexity is clearly increased as a concurrent scheduling of all ECUs becomes necessary. Hence, while the schedule synthesis might be applied for each ECU independently for version 2.1, FlexRay 3.0 requires a global scheduling approach that requires a more complex optimization model.

[3]For two messages m and $\tilde{m}$ with $1 < r_m < r_{\tilde{m}}$ and $\gcd(r_m, r_{\tilde{m}}) = 1$, (2) is not satisfied for any base cycle combination. Therefore, the offset assignment must ensure that m and $\tilde{m}$ are placed one after the other.

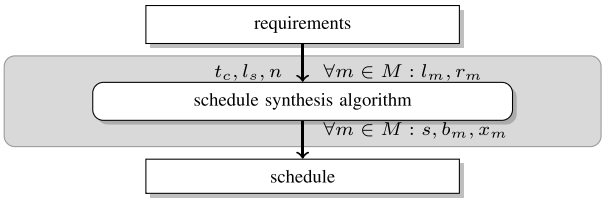


Fig. 5. Framework for FlexRay scheduling. For a set of requirements, a feasible schedule is determined. These user defined requirements include the FlexRay parameters and the message set sent over the FlexRay bus.

Limitations of FlexRay 3.0: The current FlexRay release suggests slot sharing for repetitions of $\{1, 2, 4, 5, 8, 10, 16, 20, 32, 40, 50, 64\}$ [6]. However, we argue that this restriction strongly limits the potential of the combination of a variable cycle counter and slot sharing as it only allows to exploit the full potential of slot sharing with $k = 64$ cycles. Therefore, in this paper we assume that slots can be shared for any common divisor of k. The benefits of omitting this restriction are quantitatively specified for the case study in the experimental results in Section VI.

C. Schedule Synthesis Framework

This paper proposes a framework to determine a FlexRay schedule for a set of messages. The framework allows a schedule synthesis for FlexRay 3.0, taking into account all its new features. At the same time, all approaches are backward compatible to FlexRay 2.1. The main optimization objective is the minimization of required slots which is a common approach for asynchronous scheduling since it implicitly improves the utilization of the FlexRay bus.

The framework is illustrated in Fig. 5. As input, the optimization algorithm requires the parameters of the FlexRay bus such as the communication cycle duration t_c, the available payload in a static slot l_s, and the number of static slots n. The set of messages M defines a size l_m and repetition r_m for each message m. Our framework then packs all messages into slots, defining a slot s, base cycle b_m and offset x_m for each message. FlexRay imposes the additional constraint that messages of different senders cannot share the same slot or frame for FlexRay 2.1 or version 3.0, respectively. The objective is to obtain a schedule which requires a minimal number of slots.

III. ILP-Based Optimal Schedule Synthesis

This section presents a single-stage ILP formulation to determine a message to slot assignment. It determines

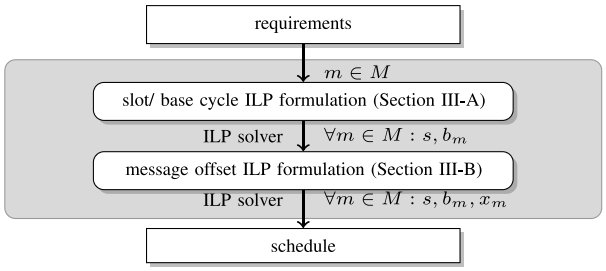


Fig. 6. Optimization flow determining a schedule with minimal bandwidth utilization. A single-stage ILP approach first assigns a slot and base cycle to a message, before the message offset is determined in a second step.

a schedule, using a minimal number of slots for a given set of messages, implicitly minimizing the bandwidth usage (see Section III-A). As FlexRay also requires a message offset within the slot, Section III-B presents an ILP approach to determine suitable offsets in a second step. Fig. 6 illustrates this approach.

A. Single-Stage ILP for Slot and Base Cycle Assignment

The ILP determines a slot s and a base cycle b_m for each message m in a given set M. For FlexRay 3.0, this schedule is determined for all ECUs in a global ILP. If slot sharing is not possible like in FlexRay 2.1, the ILP is applied to each ECU separately and the allocated slots are combined to the global schedule afterward. The ILP is based on the following constants, as introduced in detail in Section II.

1) $s \in S$: Index for slot.
2) $m \in M$: Message, where M_e indicates all messages sent by ECU $e \in E$.
3) $e \in E$: ECU, where E contains all ECUs sending at least one message.
4) $c \in \{0, \ldots, k-1\}$: Cycle, where k denotes the number of cycles.
5) l_s: Length of a slot in byte.
6) r_m: Repetition of message m.
7) l_m: Size of message m in byte.

The ILP uses the following variables.

1) $\mathbf{y}_s$: Binary variable indicating if a slot s is allocated.
2) $\mathbf{z}_{(m,s,b)}$: Binary variable indicating if message m is scheduled in slot s with base cycle b.
3) $\mathbf{v}_{(e,s,c)}$: Binary variable indicating ownership of an ECU e for slot s in cycle c.

The ILP formulation minimizes the following objective:

$$\min \sum_{s \in S} \mathbf{y}_s. \tag{3}$$

Hence, the number of allocated slots is minimized. Here, an upper bound of required slots is defined by the cardinality of S. Based on this objective, the following constraints determine a schedule defining a slot and base cycle for each message m:

$$\forall m \in M : \sum_{s \in S} \sum_{b=0}^{r_m - 1} \mathbf{z}_{(m,s,b)} = 1 \tag{4a}$$

$$\forall s \in S, c = \{0, \ldots, k-1\} : \sum_{\substack{m \in M \\ b = c \bmod r_m}} \mathbf{z}_{(m,s,b)} \cdot l_m \leq l_s \tag{4b}$$

$$\forall s \in S, m \in M : \mathbf{y}_s \geq \sum_{b=0}^{r_m - 1} \mathbf{z}_{(m,s,b)} \tag{4c}$$

$$\forall s \in S, c = \{0, \ldots, k-1\} : \sum_{e \in E} \mathbf{v}_{(e,s,c)} \leq 1 \tag{4d}$$

$$\forall s \in S, c = \{0, \ldots, k-1\}, \forall e \in E, m \in M_e, b = c \bmod r_m : \mathbf{v}_{(e,s,c)} - \mathbf{z}_{(m,s,b)} \geq 0. \tag{4e}$$

Constraint (4a) ensures that each message is scheduled in exactly one slot with a specific base cycle. To ensure messages scheduled in a slot do not exceed the slot size, the sum of all message sizes in one cycle must not be larger than the slot length as stated in constraint (4b). Furthermore, a message can only be scheduled in slots that have been allocated as defined in constraint (4c). Constraint (4d) ensures that each cycle in a slot can only be filled by messages from not more than one ECU. If a message is scheduled in a specific cycle, its sender needs to be assigned as cycle owner as stated in constraint (4e).

For FlexRay 2.1 scheduling, $k = 64$ is fixed and a slot is exclusively assigned to one sender. Hence, the ILP is executed for each ECU separately, and constraints (4d) and (4e) are omitted.

B. ILP for Message x-Offsets

The ILP presented in the previous section assigns a slot and a base cycle to all messages. As several messages can be scheduled in one slot in the same cycle, additionally an offset x_m needs to be assigned to each message m. Based on the previously calculated allocation of m to s and b_m, x_m is calculated. Note that due to the variable cycle number in FlexRay 3.0, back-tracking might be required to obtain suitable offsets. As heuristic approaches struggle with these late decisions, we have selected an ILP-based approach to determine the following variables.

1) $\mathbf{x}_m \in \mathbb{N}_0$: Integer-variable for x-offset of message m.
2) $\mathbf{x}_{(m,\tilde{m})}$: Binary variable indicating that message m is placed before message $\tilde{m}$ in the slot.

The ILP is applied to each slot separately and formulated as follows:

$$\forall m \in M_s : 0 \leq \mathbf{x}_m \leq l_s - l_m \tag{5a}$$

$$\forall m, \tilde{m} \in M_s, \exists c \in \{0, \ldots, k-1\}, c \mod r_m = b_m \wedge c \mod r_{\tilde{m}} = b_{\tilde{m}} : \mathbf{x}_m + l_m \leq \mathbf{x}_{\tilde{m}} + l_s \cdot \mathbf{x}_{(\tilde{m},m)} \tag{5b}$$

$$\mathbf{x}_{\tilde{m}} + l_{\tilde{m}} \leq \mathbf{x}_m + l_s \cdot \mathbf{x}_{(m,\tilde{m})} \tag{5c}$$

$$\mathbf{x}_{(\tilde{m},m)} + \mathbf{x}_{(m,\tilde{m})} = 1. \tag{5d}$$

The ILP first defines appropriate bounds for the x-offset for each message in constraint (5a). If two messages are sent in at least one cycle together, it holds that either the first message is transmitted before the second one starts or vice versa [see constraints (5b) and (5c)]. Here, the binary variables $\mathbf{x}_{(m,\tilde{m})}$ and $\mathbf{x}_{(\tilde{m},m)}$, respectively, are used as switch variables. Only one of the switch variables can be active as stated in constraint (5d). As the ILP is applied to each slot individually, the problem size is small compared to the slot/base cycle integer linear

programmings (ILPs). As a result, the computational complexity of the x-offset calculation is negligible compared to the slot/base cycle calculation.

Though assigning offsets in a second step generally leads to viable schedules, in theory it might happen that the ILP for determining x_m is not satisfied. This issue can be resolved through a message offset relaxation which allows to assign different offsets to a message in different cycles. To apply an offset relaxation, a message m is considered as two separate messages m_a and m_b which have the same size ($l_m = l_{m_a} = l_{m_b}$) but an increased repetition ($r_{m_a} = r_{m_b} = 2 \cdot r_m$) and different base cycles ($b_{m_a} = b_m$, $b_{m_b} = b_m + r_m$). This allows to schedule both messages with different offsets, resolving conflicts.[4] As this process does not require the allocation of additional slots, the minimal bandwidth requirement determined by the single-stage ILP is not affected. However, as an ECU generally only sends messages with a limited number of distinctive repetitions r_m, it is very unlikely that message offset relaxation is required. Moreover, when running various test cases, message offset relaxation was never required.

IV. Multistage ILP: ILP-Based Heuristic Solution

The single-stage ILP has a limited scalability for FlexRay 3.0 where messages of all ECUs are scheduled concurrently. For instance, determining a schedule for 130 messages already requires more than 24 h. However, if the scheduling problem is partitioned as it is the case for FlexRay 2.1,[5] the scalability is only limited by the number of messages sent by each single ECU instead. This makes the single-stage ILP applicable for FlexRay 2.1 scheduling of a full state-of-the-art in-vehicle network since the number of messages sent by each ECU is usually much smaller. As a remedy, this section presents a multistage ILP which partitions the scheduling problem, i.e., it determines a message to slot assignment for each ECU individually, using the single-stage ILP. A second ILP allows to combine slots to a single slot, following the FlexRay 3.0 specification [see Fig. 4(b)].

Fig. 7 illustrates the optimization flow of the multistage ILP. In the fist stage, the single-stage ILP is applied to each ECU e individually. It implicitly determines a frame $f \in F$ for each used slot. A frame $f \in F$ contains the messages $m \in M_f$. M_f is defined for a slot s as follows:

$$m \in M_f \iff \exists b \in \{0, \ldots, r_m - 1\} : \mathbf{z}_{(m,s,b)} = 1.$$

Hence, if a message m is scheduled in slot s, it is also part of the message set M_f of frame f contained in s.[6] A frame uses the cycles C_f which is defined as follows:

$$c \in C_f \iff \exists m \in M_f, a \cdot r_m + b = c : \mathbf{z}_{(m,s,b)} = 1, a \in \mathbb{N}_0.$$

Therefore, all cycles used by the messages in slot s are also used by frame f. In the second stage, an ILP integrates the individually generated subsystem schedules in a global schedule. Hence, it assigns frames f using the cycles C_f to slots, determining the final slot and base cycle for each of the frames. The results obtained by this multistage approach might not be optimal anymore, but the scalability is significantly increased. The ILP formulation is based on the following constants.

1) $s \in S$: Index for slot.
2) $m \in M$: Message, where M_f indicates all messages sent in the frame f.
3) C_f: Set of cycles used by frame f, hence, cycles in which at least one message is sent.
4) r_f: Smallest repetition of all messages scheduled in f, $r_f = \min_{m \in M_f}(r_m)$, determining the largest possible base cycle for f.

The ILP uses the following variables.

1) $\mathbf{y}_s$: Binary variable indicating if a slot s is allocated.
2) $\mathbf{u}_{(f,s,b)}$: Binary variable indicating if f is scheduled in slot s with base cycle b.

The integration ILP is formulated as

$$\min \sum_{s \in S} \mathbf{y}_s \tag{6a}$$

$$\forall f \in F : \sum_{s \in S} \sum_{b=0}^{r_f - 1} \mathbf{u}_{(f,s,b)} = 1 \tag{6b}$$

$$\forall s \in S, c = \{0, \ldots, k-1\} : \sum_{f \in F} \sum_{\substack{b \in \{0,\ldots,r_f-1\} \\ (c-b) \bmod k \in C_f}} \mathbf{u}_{(f,s,b)} \leq 1 \tag{6c}$$

$$\forall s \in S, f \in F : \mathbf{y}_s \geq \sum_{b=0}^{r_f - 1} \mathbf{u}_{(f,s,b)}. \tag{6d}$$

The objective function (6a) minimizes the number of allocated slots. The upper bound is the number of slots required if slots are not shared. Constraint (6b) ensures that each f is only placed once. It also implies that b must be smaller than r_f. Furthermore, each cycle in a slot cannot be used by more than one frame as stated in constraint (6c). As the used cycles in C_f do not necessarily have a constant repetition, all base cycles which would lead to a utilization of cycle c are determined. Constraint (6d) ensures that the frame f can only be scheduled in slots that have been allocated.

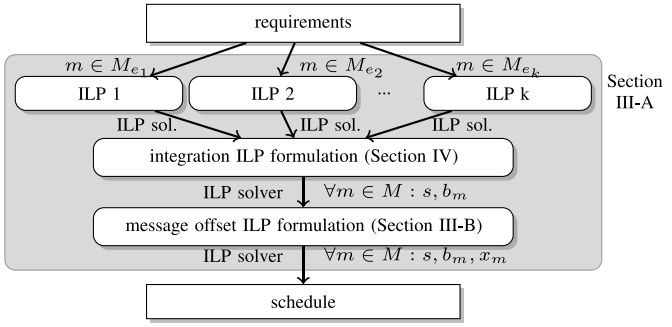


Fig. 7. Optimization flow for the multistage ILP approach. First, the single-stage ILP approach is applied for each subsystem individually, before the integration ILP combines the results to a global schedule. Finally, the message offsets are determined.

[4]Note that offset relaxation might be applied multiple times to a message, as long as the repetition of the submessages m_a and m_b does not exceed the number of FlexRay cycles k.

[5]For FlexRay 2.1 the messages sent by each ECU are scheduled individually since slot sharing is not permitted.

[6]The variable $\mathbf{z}_{(m,s,b)}$ indicates that message m is scheduled in slot s with a base cycle of b as introduced in the previous section.

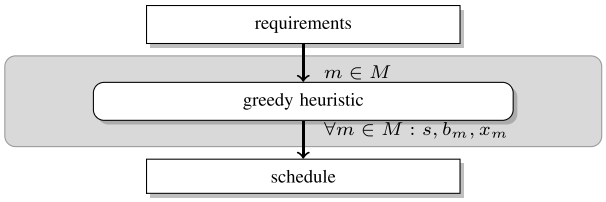


Fig. 8. Schedule synthesis with the greedy heuristic.

Algorithm 1 Greedy Heuristic for FlexRay Scheduling

```
1:  S = {}
2:  M_σ = defineOrder(M)
3:  for m ∈ M_σ do
4:      for s ∈ S do
5:          C_(s,e) = availableCycles(s, e_m)
6:          if place(m, s, C_(s,e)) then
7:              continue with next m
8:          end if
9:      end for
10:     create new s and add it to S
11:     C_(s,e) = {0, 1, ..., k − 1}
12:     place(m, s, C_(s,e))
13: end for
```

In the automotive domain, legacy subsystems commonly have to be integrated in the architecture. Configurations for these subsystems have already been intensively tested and certified for previous product generations. The multistage approach allows to integrate these subsystem schedules in a global system, clearly minimizing the efforts for testing and certification.

V. Greedy Heuristic

This section first presents a greedy approach for schedule synthesis. Second, an optimized order for the greedy approach as well as GA and SA approaches are proposed.

A. Schedule Synthesis

In the following, a schedule synthesis based on a greedy approach is presented. For FlexRay 3.0, this approach is applied to the entire set of messages. For FlexRay 2.1, slots cannot be shared by ECUs and thus the approach is applied to the message set of each ECU separately and the slots are finally combined to a schedule. In contrast to the ILP approaches, the greedy approach assigns the slot s, basecycle b_m and offset x_m to a message m in a single iteration (see Fig. 8). With $C_{(s,e)}$ denoting the set of cycles in slot s in which the ECU e might send its messages, the greedy heuristic is outlined in Algorithm 1. It iteratively fills the schedules with messages and allocates new slots if necessary.

The algorithm starts with an empty set of slots S. After the messages M have been put in order (line 2), the heuristic tries to place each message m in one of the existing slots s (lines 3–9). If a message cannot be placed in any existing slot in S, a new slot s is allocated and the message is placed in this new slot (lines 10–12). Before a message is placed in s, the function availableCycles(s, e_m) determines the set of cycles $C_{(s,e)}$ in which the sender of the current message e_m is allowed to schedule its messages (line 5). This set of cycles $C_{(s,e)}$ might be sparse in case another ECU already scheduled a message in the same slot. In a next step, the function place$(m, s, C_{(s,e)})$ tries to place the message m into the slot s by considering the available cycles and determining a valid base cycle b_m and offset x_m. Here, a message m can be placed in the slot s if: 1) the base cycle b_m is smaller than the message repetition r_m and 2) m does not intersect with other messages for all its repetitions. The function place$(m, s, C_{(s,e)})$ places messages in lower cycle numbers first and fills them from the left to achieve a dense scheduling. If a suitable b_m and offset x_m can be found, place$(m, s, C_{(s,e)})$ returns true and the algorithm continues with the next message. In case the message cannot be scheduled in any existing slot, a new slot is allocated in which the message is scheduled. The proposed heuristic is a greedy algorithm and the runtime depends on the number of messages as well as the number of assigned slots. The complexity is therefore defined as $\mathcal{O}(|M| \cdot |S|)$, being polynomial in the number of messages as $\mathcal{O}(|M| \cdot |S|) \leq \mathcal{O}(|M|^2)$.

B. Determine Message Order

The order of messages M_σ strongly affects the results. The following order is proposed: 1) messages are arranged by ascending repetition and 2) messages with the same repetition are arranged by their message size in descending order. The greedy heuristic benefits from 1) as messages with the same repetition are scheduled in one group and can therefore be easily combined in one slot. At the same time, messages with a high repetition are placed later and used to fill unused cycles in a slot as they occupy less cycles (k/r_m) and are therefore more flexible in their placement. Similarly, 2) ensures that large messages are placed first while smaller messages are used to fill remaining payload in the slot.

In the following, the order is introduced formally. Given a set of messages $\{m_0, m_1, \ldots, m_n\} \in M$, the message order is defined by the permutation $\sigma : \mathbb{N}_0 \to \mathbb{N}_0$. Arranging the message set by the repetition r_m is defined as

$$\forall m \in M, i, j \in \mathbb{N}_0, i \neq j, r_{m_{\sigma(i)}} \leq r_{m_{\sigma(j)}} : 0 \leq i \leq j \leq n. \quad (7)$$

The resulting list is then defined as $M_\sigma = \{m_{\sigma(0)}, m_{\sigma(1)}, \ldots, m_{\sigma(n)}\}$. As two messages might have equal repetitions, messages with a larger size l_m are additionally ordered to the front

$$\forall m \in M_\sigma, i, j \in \mathbb{N}_0, i \neq j, r_{m_{\sigma(i)}} = r_{m_{\sigma(j)}}, l_{m_{\sigma(i)}} \geq l_{m_{\sigma(j)}} : 0 \leq i \leq j \leq n. \quad (8)$$

The ordered message list M_σ improves the results obtained by the greedy heuristic compared to an unsorted list as shown in the following section.

C. Genetic Algorithm-Optimized Order

In the following, an efficient GA [16] approach is presented for FlexRay scheduling. The main procedure of an GA is based on reproduction and selection that are performed alternately in an iterative process to optimize a given objective.

Reproduction creates new individuals from the current population, using mutation and crossover operators. For the experimental results, a population size of 100 is used, generating 25 offspring individuals in each generation. The task of selection is to remove the worst individuals in terms of their fitness function to ensure a convergence of the algorithm toward the optimal solutions. Here, elitism selection is used that always removes the worst 25 individuals in each generation. These parameters are default values of the used optimization framework [17] and might be adapted to further improve the results which, however, is beyond of the scope of this paper. In our experiments, we use 200 generations and, thus, the optimization is stopped after 5000 evaluated solutions. Generally, a longer runtime of meta-heuristics improves the quality of results.

One important factor of a GA optimization is the problem encoding. While it would be possible to encode the mapping of each message to a slot, this would result in many infeasible solutions as particularly for larger problems obtaining overfull slots is highly probable. Instead, we use a decoding-based approach such that each individual is feasible: the genetic representation of an individual in this case is a permutation of messages M_σ that is always decoded to a feasible scheduling, using the greedy heuristic Algorithm 1. In this case, the initial population is determined by a random order of messages M_σ while the fitness function is the number of required slots of the currently determined schedule. As a result, the runtime for the evaluation of a single individual is dominated by Algorithm 1 which is polynomial. Note that a random initialization is performed to prevent premature convergence as well as to allow fair comparison with the other approaches. In practice, the GA might be initialized with a good fix order to obtain potentially better results in a short time.

For the crossover and mutation operator, we rely on the standard operators in [17]. Here, the crossover takes a sublist of the first permutation from the beginning to a random cut point and fills the remaining elements from the second permutation. The mutation is applied with a probability of $1/|M|$ and either moves a single element, swaps two elements, or reverts a random sublist.

D. Simulated Annealing-Optimized Order

We further apply SA [18] which is an optimization algorithm inspired by the annealing process in metallurgy. The algorithm varies a single solution by a neighborhood operator. The common neighborhood operator for binary problems is a bit flip or the sampling from a normal distribution for real-valued problems. Corresponding to the GA approach, a permutation of messages is used to represent a solution to ensure the feasibility of the solution by using the greedy heuristic from Algorithm 1 as decoder. The neighbor is determined by the neighbor operator for permutations in [17]: either a single element is moved, two elements are swapped, or a sublist is reverted.

SA improves a single solution iteratively. Here, we accept and continue with an order M'_σ, which is the neighbor of M_σ, if $f(M'_\sigma) < f(M_\sigma)$ where f is the fitness function that uses the heuristic in Algorithm 1 to determine the number of required slots. In case the neighbor solution does not improve the number of slots, it is accepted with a probability of

$$e^{\frac{f(M_\sigma)-f(M'_\sigma)}{T_i}} \tag{9}$$

where T_i is the temperature at iteration i. As cooling function

$$T_i = T_0 \cdot \frac{n-i}{n} \tag{10}$$

is used with an initial temperature of $T_0 = 1$ and the number of iterations $n = 5000$. A high number of iterations generally improves the solutions but also increases the runtime linearly. Corresponding to the GA approach, the SA is initialized with a random order.

VI. Experimental Results

To evaluate the schedule synthesis techniques, an extensive performance analysis and a large automotive case study is presented. All optimizations have been carried out on an Intel Xeon 3.2 GHz Quad Core with 12 GB RAM and CPLEX version 12.6 [19] as ILP solver for solving the ILP formulations and Opt4J 3.1.2 [17] for the GA and SA, respectively. As a measure for the quality of the obtained schedules, the number of slots required to schedule a set of messages is used. One additional major criterion is the runtime of the method since in particular methods based on ILP formulations might not scale well. Note that the schedule is obtained at design time such that runtimes of several minutes and even hours are still acceptable. As an upper bound for the number of slots for the single-stage ILP formulation, the result obtained by the greedy heuristic is used. The times stated include the generation of the subsystem schedules and message offsets. In the following, first an extensive performance analysis is presented. Second, an automotive case study representing a full in-vehicle network is discussed.

A. Performance Analysis

In the following, an extensive case study consisting of 420 grouped synthetic test cases is presented. The size of the test cases ranges from 40 to 300 messages sent by eight ECUs. The message size and period distribution follows typical message sets used in the automotive industry (see [9] for details). The messages were randomly assigned to the sending ECUs. For FlexRay 2.1 the schedule obtained by the multistage ILP is identical with the single-stage ILP.

Fig. 9 shows the results of the case study. The number of FlexRay cycles is $k = 64$ for both FlexRay 2.1 and version 3.0. The results show that all approaches enable a better bandwidth utilization with FlexRay 3.0. At the same time, the runtime is increased due to the increased complexity. The single-stage ILP is unable to terminate its calculation before reaching a time-out of 1 h for test cases with more than 100 messages. If the greedy heuristic is applied to an ordered message set, it is able to determine close to optimal results for both FlexRay versions. The GA and SA approaches are also able to determine close to optimal results for small message sets,

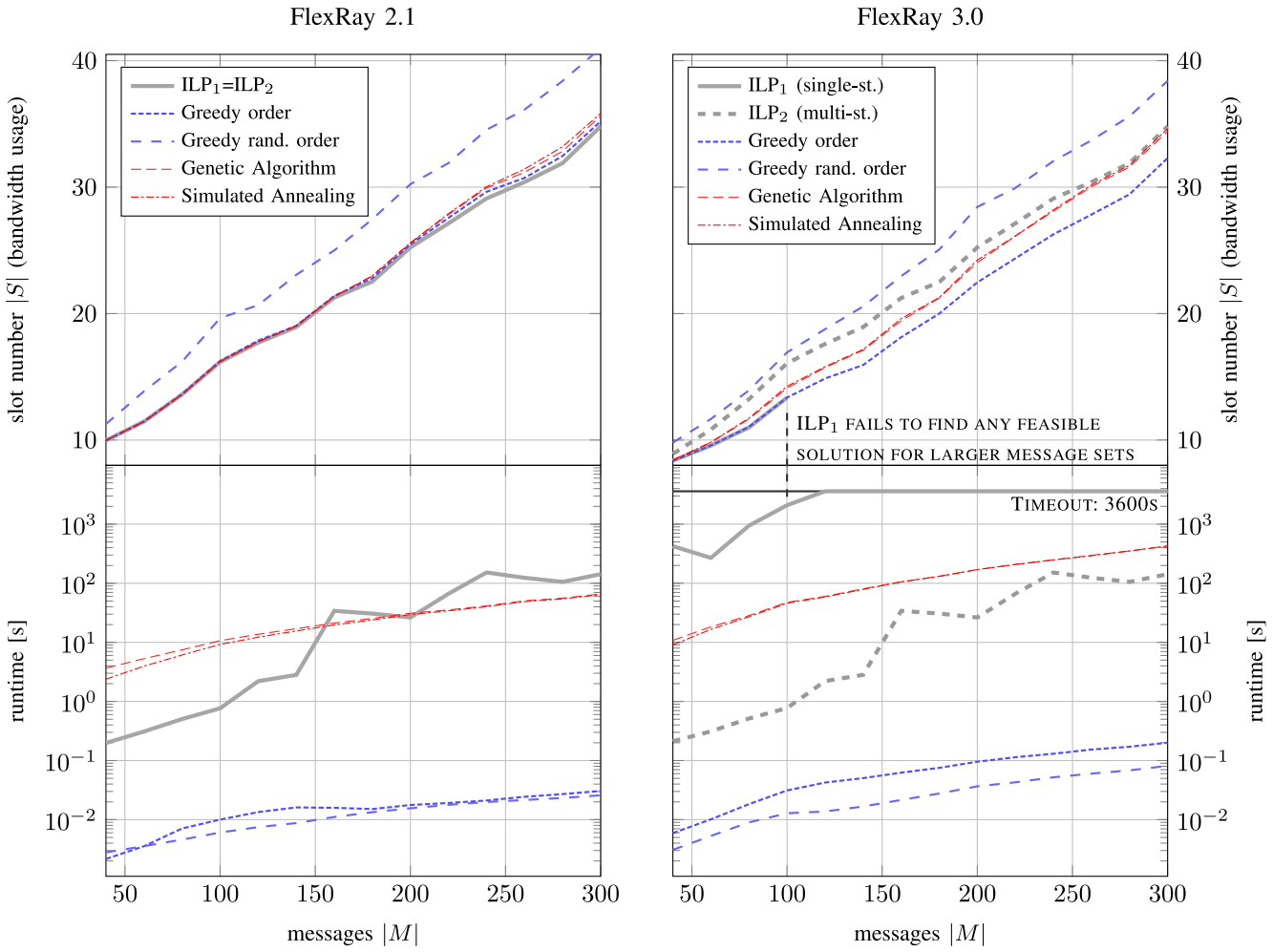


Fig. 9. Analysis of bandwidth utilization and runtime for 420 synthetic test cases. To improve legibility, the test cases are grouped by their message number and the average value is displayed.

but with an increasing problem size they are outperformed by the heuristic with an ordered message set. For FlexRay 2.1, for 6.0% and 4.0% of the test cases, respectively, GA and SA find a better solution than the heuristic which finds a better solution for 13.6% of the test cases. For FlexRay 3.0, the heuristic finds a better solution for 83.5% of the test cases while GA and SA do not outperform the heuristic for any test case. Increasing the number of iterations or fitness function evaluations, respectively, for GA and SA or improving their parameterization would reduce this gap at the cost of additional runtime or configuration. For FlexRay 3.0, the multistage ILP is able to reduce the bandwidth utilization for several test-cases compared to FlexRay 2.1 scheduling. Finally, if the greedy heuristic is applied to a randomly-ordered message set, it is clearly outperformed by all other approaches. The greedy heuristics determine all their solutions in less than a second. Note that the runtime of the metaheuristics scales linearly in the number of fitness function evaluations while each of these requires the decoding with the greedy heuristic and, thus, for 5000 evaluations their runtime is about three orders of magnitude higher. The runtime of the multistage ILP for FlexRay 3.0 strongly depends on the runtime of the first stage, determining a schedule for each ECU individually. Here, the runtime increase depends less on the overall number of messages, but rather the number of messages sent by each ECU.

Please note, the runtime in general increases with the number of messages. However, the test cases differ in their complexity. Single test-cases might therefore have an increased runtime compared to test cases with a larger message number. For instance, the deviations at 160 and 240 for the multistage ILP are resulting from single test cases with an increased complexity.

Sorting Strategy Heuristic: As selecting a suitable sorting strategy strongly influences the results obtainable by the heuristic, in the following we evaluate different sorting strategies for the same 420 synthetic test cases. We evaluate our sorting strategy which sorts messages by increasing repetition and decreasing message size ($r_m \uparrow l_m \downarrow$) to an order with increasing repetition and increasing size ($r_m \uparrow l_m \uparrow$), decreasing repetition and decreasing size ($r_m \downarrow l_m \downarrow$), decreasing repetition and increasing size ($r_m \downarrow l_m \uparrow$) and an unsorted message set.

Fig. 10 shows the results of this evaluation. The results indicate the benefits of sorting messages by an increasing repetition and a decreasing size. In particular, sorting messages by

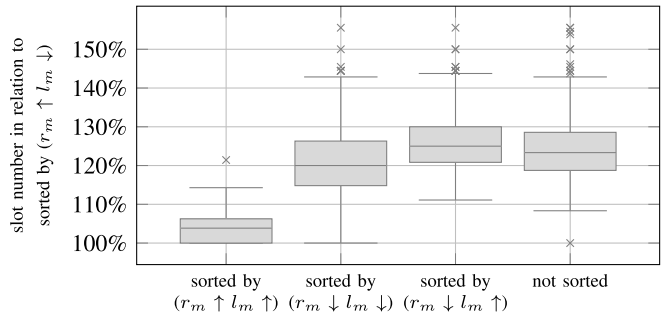


Fig. 10. Analysis of the message order regarding the number of slots required to schedule a message set of 420 test cases. The box plot illustrates the relation of the slots required by alternative message order strategies in comparison to the selected sorting strategy. It illustrates the median, the lower 0.25-quartile and the upper 0.75-quartile. The whiskers represent the lowest value within 1.5 interquartile-range (IQR) of the lower quartile, and the highest value within 1.5 IQR of the upper quartile. All values which are not within this range are illustrated independently.

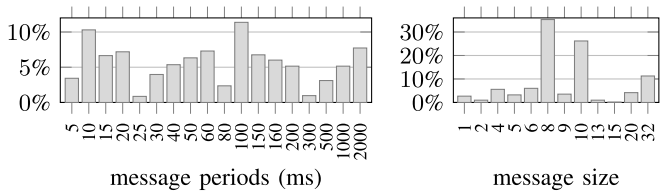


Fig. 11. Distribution of message periods and sizes for automotive case study.

an increasing repetition has a strong influence and accounts for more than 20% reduced slot usage. Sorting the messages by decreasing repetition and increasing size leads to even worse results than if the heuristic is applied to an unsorted message set. Overall, the selected sorting strategy, increasing the repetition and decreasing the size, clearly leads to the best results and is not outperformed for any of the test cases.

B. Automotive Case Study

This section presents a case study modeling an entire state-of-the-art in-vehicle network connected by a FlexRay bus as envisioned in [20]. The network consists of 32 ECUs, sending 932 messages of different sizes and periods as illustrated in Fig. 11. The parameters of the FlexRay bus were predefined such that the static segment consists of 62 slots with a slot payload of 42 bytes. As AUTOSAR requires that 1 byte of the payload is reserved for update bits, only $l_s = 41$ bytes can be used for scheduling. The duration of a communication cycle was set to $t_c = 5$ ms.

The results in Table II show that with FlexRay 2.1 it is not possible to schedule all messages within 62 slots. For FlexRay 3.0, the greedy heuristic based on an ordered message set is able to schedule the messages with the given FlexRay parameters. If, in addition, the cycle number is selected as 60, the GA and SA algorithms are able to schedule all messages. Sixty cycles lead to a clear reduction of bandwidth utilization for all approaches. Here, the number of slots might be reduced by up to 10%, depending on the applied approach. The reason for the better bandwidth usage with 60 cycles with FlexRay 3.0 is a more efficient bandwidth usage due to the avoidance of oversampling, e.g., for message repetitions

TABLE II
RESULTS OF AUTOMOTIVE CASE STUDY

Method	**60 cycles**		**64 cycles**	
	runtime	slots	runtime	slots
FlexRay2.1				
ILP_1 = ILP_2		-	3.8s	76[!]
Greedy order		-	0.039s	76[!]
Genetic Algorithm		-	172.2s	76[!]
Simulated Annealing		-	173.4s	76[!]
FlexRay 3.0				
ILP_1 (single-st.)	24h[2]	-	24h[2]	-
ILP_2 (multi-st.)	913.5s	63[!]/70[1]	907.3s	69[!]/76[1]
Greedy order	0.76s	54	1.01s	60
Genetic Algorithm	2198.1s	57	3167.9s	63[!]
Simulated Annealing	2218.4s	57	3201.6s	63[!]
Greedy order lim. rep.[3]	0.81s	62	1.01s	60

[!] exceeds number of slots in static segment
[1] result without applying integration ILP
[2] timeout
[3] message repetitions limited to $\{1, 2, 4, 5, 10, 20\}$ according to the FlexRay 3.0 standard.[6]

that are multiples of 3. The results furthermore show that the ILP approach is unable to find a solution within 24 h. The multistage ILP shows its ability to efficiently integrate independent schedules but is unsuitable for the current case study.

For this test case we assumed that the message repetition might be any divisor of 60. However, the FlexRay 3.0 standard currently limits the message repetitions to values of {1, 2, 4, 5, 8, 10, 20, 32, 40, 50, 64}. Consequently, the results obtained with our heuristic for a cycle number of 60 are clearly worse, increasing the number of required slots for this test case by 14.8% for 60 cycles. We therefore suggest to omit this limitation on message repetitions to further improve the bandwidth utilization with FlexRay.

VII. RELATED WORK

The FlexRay protocol specification was developed by the FlexRay consortium including Bosch, BMW, General Motors, and Volkswagen. The protocol was finalized in 2010 with the release of FlexRay 3.0. After its first implementation in a series-production vehicle with the BMW X5 in 2006 [5], the FlexRay bus is currently becoming an integral part of the in-vehicle networks of top-of-the range cars [4], [21]. All current series-production vehicles using the FlexRay bus are compliant with the FlexRay AUTOSAR interface specification [15].

Recent publications cover various topics regarding FlexRay networks. In [22] an integrated system architecture relying on a time-triggered system which uses FlexRay for external communication is proposed. Broy and Muller-Glaser [23] gave an introduction to the configuration of FlexRay applications. An optimization method to define optimal parameters for the static segment and the communication cycle can be found in [24]. In [25] and [26], a timing and performance analysis of the FlexRay protocol is proposed with a focus on the dynamic segment. Several schedule optimization methods have been proposed for the dynamic FlexRay segment like in [27], [28], or [29]. However, here the focus is on the static segment as the dynamic segment is commonly only used for diagnosis and configuration.

For time-triggered scheduling in the static segment, generally two different approaches are applied: 1) synchronous scheduling and 2) asynchronous scheduling. In a synchronous scheduling (see [13], [14]), the network participants execute tasks in a time-triggered manner in accordance with the bus schedule. The goal of synchronous scheduling is the minimization of the entire end-to-end delay at the cost of a significantly higher complexity. In the automotive domain, ECUs, their functions, and their schedules are often developed and determined independently, preventing a synchronous scheduling. As a result, many applications and robust control functions do not require synchronization of tasks such that asynchronous scheduling may be applied. This reduces the complexity of the scheduling and integration efforts significantly.

For the static segment, the FlexRay standard intends a fixed message to slot assignment. However, various approaches have been proposed to increase the flexibility of a FlexRay schedule. For instance, Lange *et al.* [30], [31] presented a middleware which allows to pack messages to slots at runtime. A scheduling algorithm based on response time analysis is proposed to determine a slot to ECU assignment supporting a predefined message set. Another approach was presented in [32], allowing to preempt and pack single messages in multiple slots. It proposes an ILP and a heuristic approach to determine a slot to ECU assignment. While these approaches allow a more flexible utilization of the FlexRay bus, in the automotive industry FlexRay is currently used in domains which benefit from the predefined static schedule, such as the chassis domain. The deterministic behavior of a predefined schedule also improves the safety of the system and facilitates testing and diagnosis. In the work at hand, we therefore focus on determining a static FlexRay schedule as intended by AUTOSAR.

Several approaches have been proposed for a static FlexRay scheduling. For instance, Ding *et al.* [33] presented an approach for asynchronous FlexRay scheduling using a GA. The approach also takes the timing delay from sending to receiving tasks into account. However, as Ding [34] stated, GAs do not scale for nontrivial problems, and therefore propose a hybrid approach using a mix of a bin-packing algorithm and a GA. In [7], a two-stage ILP approach is presented. It packs signals to frames in a first step before determining a schedule from the frames. Two scheduling approaches, complying with the AUTOSAR standard, can be found in [8] and [9]. In [8], FlexRay schedules are obtained by a heuristic, assuming one signal per frame and, thus, leading to inferior bandwidth utilization. Lukasiewycz *et al.* [9] presented a schedule synthesis that considers the packing of messages to slots based on the bin-packing problem. In summary, all discussed approaches are tailored to FlexRay 2.1. As a remedy, a generalized approach that obtains asynchronous schedules and is applicable to the different versions of FlexRay is presented.

Approaches supporting FlexRay 3.0 features have been proposed in [35] and [36], addressing the problem of holistic scheduling with FlexRay. Hu *et al.* [35] proposed heuristic approaches based on list scheduling while Darbandi and Kim [36] applied a transformation to the strip packing problem to schedule the messages. While both approaches support slot sharing as introduced by FlexRay 3.0, they do not consider a variable cycle number. An approach for asynchronous FlexRay 3.0 scheduling is presented in [10]. In contrast to the work at hand, this paper only considers a packing of frames to slots, and assumes the packing of messages to frames was done in a previous design step. It is therefore not applicable to many relevant automotive scenarios that rely on the AUTOSAR specification where a given frame packing is not assumed. Kang *et al.* [11] presented an approach for packing signals to frames, supporting multiple periods in a frame as supported by FlexRay 3.0. To generate the final schedule, it relies on the scheduling analysis proposed in [37]. While these approaches reduce the problem complexity, they also lead to clearly worse results than an implicit frame packing that is defined by scheduling of messages directly into slots. Finally, in [12] an approach for holistic scheduling for a system connected by a FlexRay bus is presented. This paper proposes an ILP approach which considers a direct packing of messages to slots suitable for FlexRay 3.0 scheduling. However, Darbandi *et al.* [12] do not consider a message offset assignment. Offset assignment becomes necessary for FlexRay 3.0 where a variable cycle number allows to send messages with repetitions that only share 1 as greatest common divisor in one slot (see Section II-B for details). As a remedy, this paper proposes several approaches that comply with the AUTOSAR specification and considers a direct message to slot packing for the single-stage ILP and the greedy approach, obtaining the best possible utilization of the bus. In addition, the multistage ILP allows to convert legacy FlexRay 2.1 schedules to FlexRay 3.0 schedules with reduced bandwidth requirements. All approaches include a message offset assignment which has been omitted by previous work.

VIII. Conclusion

This paper proposes a generic framework to schedule the static segment of the FlexRay bus using asynchronous communication. The presented results show the clear potential of FlexRay 3.0 to improve the bandwidth usage compared to version 2.1, making optimal approaches for FlexRay 3.0 necessary. Previous work largely addressed FlexRay 2.1 and does not extend to version 3.0 while existing approaches for 3.0 are nonoptimal in several aspects. In contrast, the scheduling approaches presented here support all features of version 3.0 while being backward compatible. Several scheduling approaches are presented.

1) A single-stage ILP approach determining an optimal solution.
2) A multistage ILP, integrating previously generated subsystem schedules into a global schedule.
3) A greedy heuristic obtaining good results with a minimal runtime.
4) A GA and an SA approach to evaluate the proposed approaches.

This paper presents an extended performance analysis and a realistic case study. The results show that the proposed algorithms are capable of finding feasible solutions for both FlexRay 2.1 and FlexRay 3.0. They also reflect the benefits of

the new FlexRay 3.0 features to reduce the bandwidth requirements. At the same time, it is shown that for FlexRay 3.0, the ILP approach does not scale well and becomes intractable for large sets of messages. However, for small size problems where runtimes of up to several days are acceptable, it provides an optimal solution. The greedy heuristic generates highly competitive results in a short time if the messages are arranged by their repetition and size before scheduling. It might therefore be applied for large message sets where the ILP approach becomes intractable. The GA and SA approaches were only able to obtain better results than the heuristic for small test cases and FlexRay 2.1. For large message sets and, in particular FlexRay 3.0, the heuristic clearly outperforms the metaheuristics in terms of obtained results and runtime. Furthermore, the results show that the multistage ILP is suitable for optimizing schedules in an incremental design approach. It might be applied to convert existing legacy FlexRay 2.1 schedules to 3.0, or to integrate existing subsystems into an architecture if reduced testing and integration efforts are more important than optimal bandwidth usage. The large case study shows the necessity for FlexRay 3.0 where the novel features help to obtain a feasible schedule. Here, in particular, setting the number of cycles to 60 allows a better bandwidth utilization, if all feasible repetitions are permitted. We therefore suggest to omit the limitation on the message repetition, as defined by the FlexRay standard [6].

In future work, the presented framework will be extended to support task scheduling on the ECUs. Various work has been done in the area of processor scheduling, and we will investigate how the presented FlexRay schedule synthesis can be integrated with these approaches for a global schedule synthesis in time-triggered systems. As synchronous scheduling generally suffers from a limited scalability, the focus will be on schedule integration techniques extending the multistage ILP presented here.

REFERENCES

[1] S. C. Ergen *et al.*, "The tire as an intelligent sensor," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 28, no. 7, pp. 941–955, Jul. 2009.

[2] K. W. Tindell, H. Hansson, and A. J. Wellings, "Analysing real-time communications: Controller area network (CAN)," in *Proc. Real Time Syst. Symp. (RTSS)*, San Juan, PR, USA, Dec. 1994, pp. 259–263.

[3] Y. Yamaguchi and T. Murakami, "Adaptive control for virtual steering characteristics on electric vehicle using steer-by-wire system," *IEEE Trans. Ind. Electron.*, vol. 56, no. 5, pp. 1585–1594, May 2009.

[4] J. Kötz and S. Poledna, "Making FlexRay a reality in a premium car," in *Proc. SAE Converg.*, Oct. 2008, p. 8.

[5] C. Böke, "FlexRay is driving," *Vector Informat.*, Mar. 2007.

[6] FlexRay Consortium, *FlexRay Communications System Protocol Specification Version 3.0.1*, 2010.

[7] K. Schmidt and E. G. Schmidt, "Message scheduling for the FlexRay protocol: The static segment," *IEEE Trans. Veh. Technol.*, vol. 58, no. 5, pp. 2170–2179, Jun. 2009.

[8] M. Grenier, L. Havet, and N. Navet, "Configuring the communication on FlexRay: The case of the static segment," in *Proc. Embedded Real Time Softw. (ERTS)*, Toulouse, France, Jan. 2008.

[9] M. Lukasiewycz, M. Glaß, J. Teich, and P. Milbredt, "FlexRay schedule optimization of the static segment," in *Proc. Hardw. Softw. Codesign Syst. Synth. (CODES+ISSS)*, Grenoble, France, Oct. 2009, pp. 363–372.

[10] T. Schenkelaars, B. Vermeulen, and K. Goossens, "Optimal scheduling of switched FlexRay networks," in *Proc. Design Autom. Test Europe (DATE)*, Grenoble, France, Mar. 2011, pp. 1–6.

[11] M. Kang, K. Park, and M.-K. Jeong, "Frame packing for minimizing the bandwidth consumption of the FlexRay static segment," *IEEE Trans. Ind. Electron.*, vol. 60, no. 9, pp. 4001–4008, Sep. 2013.

[12] A. Darbandi, S. Kwon, and M. K. Kim, "Scheduling of time triggered messages in static segment of FlexRay," *Int. J. Softw. Eng. Appl.*, vol. 8, no. 6, pp. 195–208, 2014.

[13] H. Zeng, M. Di Natale, A. Ghosal, and A. Sangiovanni-Vincentelli, "Schedule optimization of time-triggered systems communicating over the FlexRay static segment," *IEEE Trans. Ind. Informat.*, vol. 7, no. 1, pp. 1–17, Feb. 2011.

[14] D. Goswami, M. Lukasiewycz, R. Schneider, and S. Chakraborty, "Modular scheduling of distributed heterogeneous time-triggered automotive systems," in *Proc. Asia South Pac. Design Autom. Conf. (ASP-DAC)*, Sydney, NSW, Australia, Jan. 2012, pp. 665–670.

[15] AUTOSAR. (2014). *AUTOSAR 4.2*. [Online]. Available: http://www.autosar.org

[16] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*. Berlin Heidelberg, Germany: Springer-Verlag, 1996.

[17] M. Lukasiewycz, M. Glaß, F. Reimann, and J. Teich, "Opt4J—A modular framework for meta-heuristic optimization," in *Proc. Genet. Evol. Comput. Conf. (GECCO)*, Dublin, Ireland, 2011, pp. 1723–1730.

[18] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[19] ILOG. *CPLEX*. Accessed on May 26, 2016. [Online]. Available: http://www.ilog.com/products/cplex/

[20] S. Schliecker *et al.*, "System level performance analysis for real-time automotive multicore and network architectures," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 28, no. 7, pp. 979–992, Jul. 2009.

[21] J. Berwanger, M. Peteratzinger, and A. Schedl, "FlexRay startet durch—FlexRay-bordnetz fuer fahrdynamik und fahrerassistenzsysteme (in German)," *Elektronik Automotive: Sonderausgabe 7er BMW*, Oct. 2008.

[22] R. Obermaisser, C. El Salloum, B. Huber, and H. Kopetz, "From a federated to an integrated automotive architecture," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 28, no. 7, pp. 956–965, Jul. 2009.

[23] J. Broy and K. D. Muller-Glaser, "The impact of time-triggered communication in automotive embedded systems," in *Proc. Int. Symp. Ind. Embedded Syst. (SIES)*, Lisbon, Portugal, Jul. 2007, pp. 353–356.

[24] I. Park and M. Sunwoo, "FlexRay network parameter optimization method for automotive applications," *IEEE Trans. Ind. Electron.*, vol. 58, no. 4, pp. 1449–1459, Apr. 2011.

[25] G. Cena and A. Valenzano, "On the properties of the flexible time division multiple access technique," *IEEE Trans. Ind. Informat.*, vol. 2, no. 2, pp. 86–94, May 2006.

[26] A. Hagiescu *et al.*, "Performance analysis of FlexRay-based ECU networks," in *Proc. Design Autom. Conf. (DAC)*, San Diego, CA, USA, Jun. 2007, pp. 284–289.

[27] T. Pop, P. Pop, P. Eles, and Z. Peng, "Bus access optimisation for FlexRay-based distributed embedded systems," in *Proc. Design Autom. Test Europe (DATE)*, Nice, France, Apr. 2007, pp. 1–6.

[28] S. Samii, Y. Yin, Z. Peng, P. Eles, and Y. Zhang, "Immune genetic algorithms for optimization of task priorities and FlexRay frame identifiers," in *Proc. Embedded Real Time Comput. Syst. Appl. (RTCSA)*, Beijing, China, Aug. 2009, pp. 486–493.

[29] E. G. Schmidt and K. Schmidt, "Message scheduling for the FlexRay protocol: The dynamic segment," *IEEE Trans. Veh. Technol.*, vol. 58, no. 5, pp. 2160–2169, Jun. 2009.

[30] R. Lange, F. Vasques, P. Portugal, and R. S. de Oliveira, "Guaranteeing real-time message deadlines in the FlexRay static segment using a on-line scheduling approach," in *Proc. Workshop Fact. Commun. Syst. (WFCS)*, Lemgo, Germany, May 2012, pp. 301–310.

[31] R. Lange, F. Vasques, R. S. de Oliveira, and P. Portugal, "A scheme for slot allocation of the FlexRay static segment based on response time analysis," *Comput. Commun.*, vol. 63, pp. 65–76, Jun. 2015.

[32] P. Mundhenk, F. Sagstetter, S. Steinhorst, M. Lukasiewycz, and S. Chakraborty, "Policy-based message scheduling using FlexRay," in *Proc. Hardw. Softw. Codesign Syst. Synth. (CODES+ISSS)*, New Delhi, India, Oct. 2014, pp. 1–10.

[33] S. Ding, N. Murakami, H. Tomiyama, and H. Takada, "A GA-based scheduling method for FlexRay systems," in *Proc. Embedded Softw. (EMSOFT)*, Jersey City, NJ, USA, May 2005, pp. 110–113.

[34] S. Ding, "Scheduling approach for static segment using hybrid genetic algorithm in FlexRay systems," in *Proc. Comput. Inf. Technol. (CIT)*, Bradford, U.K., Jun. 2010, pp. 2355–2360.

[35] M. Hu, J. Luo, Y. Wang, M. Lukasiewycz, and Z. Zeng, "Holistic scheduling of real-time applications in time-triggered in-vehicle networks," *IEEE Trans. Ind. Informat.*, vol. 10, no. 3, pp. 1817–1828, Aug. 2014.
[36] A. Darbandi and M. K. Kim, "Schedule optimization of static messages with precedence relations in FlexRay," in *Proc. Int. Conf. Ubiquitous Future Netw. (ICUFN)*, Shanghai, China, Jul. 2014, pp. 495–500.
[37] T. Pop, P. Pop, P. Eles, Z. Peng, and A. Andrei, "Timing analysis of the FlexRay communication protocol," *Real Time Syst.*, vol. 39, nos. 1–3, pp. 205–235, Aug. 2008.

Florian Sagstetter (M'13) received the Dipl.-Ing. degree in electrical engineering from the Technical University of Munich (TUM), Germany, in 2010, where he is currently pursuing the Ph.D. degree.

He is currently a Research Associate with the TUM Campus for Research Excellence and Technological Enterprise (CREATE), Centre for Electromobility, Singapore. His current research interests include embedded systems in the automotive domain with a focus on time-triggered systems and schedule synthesis.

Martin Lukasiewycz (M'11) received the Ph.D. degree in computer science from the University of Erlangen-Nuremberg, Germany, in 2010.

He was a Principal Investigator with the TUM CREATE, Singapore, from 2011 to 2015. He was with the E/E Architecture and FlexRay Division at AUDI AG, Germany, the University of Erlangen-Nuremberg, and TUM. From 2014 to 2015, he was an Adjunct Assistant Professor with Nanyang Technological University, Singapore. Since 2015, he has been with Robert Bosch GmbH, Germany.

Samarjit Chakraborty (SM'15) received the Ph.D. degree in electrical and computer engineering from ETH Zurich, Switzerland, in 2003.

He is a Professor of Electrical Engineering at the Technical University of Munich, where he holds the Chair for Real-Time Computer Systems. Prior to joining TUM in 2008, he was an Assistant Professor of Computer Science at the National University of Singapore from 2003–2008. He obtained his Ph.D. in Electrical and Computer Engineering from ETH Zurich in 2003. His research interests include system-level design of embedded and cyber-physical systems, with applications in automotive, healthcare, and sensor network-based information processing.