Schedule Integration for Time-Triggered Systems

Florian Sagstetter, Martin Lukasiewycz
TUM CREATE
Singapore
Email: {florian.sagstetter,martin.lukasiewycz}@tum-create.edu.sg

Samarjit Chakraborty
TU Munich
Munich, Germany
Email: samarjit.chakraborty@rcs.ei.tum.de

***Abstract*—This paper presents a framework for schedule integration of time-triggered systems tailored to the automotive domain. In-vehicle networks might be very large and complex and hence obtaining a schedule for a fully synchronous system becomes a challenging task since all bus and processor constraints as well as end-to-end-timing constraints have to be taken concurrently into account. Existing optimization approaches apply the schedule optimization to the entire network, limiting their application due to scalability issues. In contrast, the presented framework obtains the schedule for the entire network, using a two-step approach where for each cluster a local schedule is obtained first and the local schedules are then merged to the global schedule. This approach is also in accordance with the design process in the automotive industry where different subsystems are developed independently to reduce the design complexity and are finally combined in the integration stage. In this paper, a generic framework for schedule integration of time-triggered systems is presented. Further, we show how this framework is implemented for a FlexRay network using an Integer Linear Programming (ILP) approach which might also be easily adapted to other protocols. A realistic case study and a scalability analysis give evidence of the applicability and efficiency of our approach.**

I. Introduction

Recent years have seen a strong increase in the functionality in vehicles based on software and electronics. A growing number of these functions like drive-by-wire or driver assistance systems have hard real-time requirements. As a result, the design paradigm in the automotive industry is currently shifting from event-triggered systems based on the Control Area Network (CAN) [12] to time-triggered systems based on FlexRay [10] or Ethernet, using Precision Time Protocol (PTP) [1] for synchronization. Time-triggered systems that use global synchronous schedules increase the safety and control quality due to their deterministic behavior and minimal jitter. Moreover, simulation, integration, and testing efforts are significantly reduced due to the predictability of the system.

Determining a schedule for synchronous time-triggered scheduling is a challenging task as a multitude of constraints like end-to-end delays or resource sharing have to be taken into account. Furthermore, applications in an automobile are generally developed in different clusters by different teams and are integrated to the complete system in a later stage. Therefore, it is generally desirable to first determine schedules for clusters, e.g., for body, chassis, or information functions, and merge these schedules in a later stage while keeping the previously defined structure.

Contributions of the paper. This paper proposes a framework for the integration of time-triggered synchronous systems. Based on a set of schedules of independent clusters which share common resources, the framework determines a global schedule, ensuring that all timing constraints are satisfied. We first present the generic problem formulation and framework for schedule integration for general time-triggered systems. Based on this generic framework, we apply the methodology to a FlexRay system that is used as the main communication bus for different subsystems. We present an ILP formulation for schedule integration which supports both FlexRay 2.1 and FlexRay 3.0. Finally, we present a realistic case study and a scalability analysis with 5400 synthetic test cases to give evidence of the applicability of our framework.

Organization of the paper. The remainder of the paper is organized as follows: Section II discusses related work. In Section III, we introduce the generic problem formulation and framework in detail. Section IV presents the ILP formulation for schedule integration for a FlexRay network. Experimental results are given in Section V before the paper is concluded in Section VI.

II. Related Work

Time-triggered systems are increasingly used in the automotive domain after the introduction of the FlexRay bus [10] by a consortium of car manufacturers and suppliers. FlexRay was designed for safety-critical applications like drive-by-wire [14] and has been introduced in various top-of-the-range cars, see [3]. At the same time, Ethernet PTP [2] is emerging as time-triggered automotive communication bus for safety-critical systems. First results using Ethernet in the automotive domain limited to simulations and timing analysis were presented in [6] and [7]. However, using a network efficiently for time-triggered communication requires a methodology that determines synchronous schedules. An approach for synchronous time-triggered scheduling for *TTEthernet* [13], a real-time Ethernet standard primarily used in industrial automation, can be found in [11]. It presents an approach based on a Satisfiability Modulo Theories (SMT) solver. To increase the scalability, the authors present an incremental approach and introduce discrete time steps. However, the scalability is still limited to small and mid-size problems and cannot be applied to large state-of-the-art in-vehicle networks. For the FlexRay bus, approaches for synchronous time-triggered scheduling have been presented recently as well. In [15], an approach for synchronous scheduling at *job-level* is presented with the goal of minimizing the used bandwidth. In [8], the authors present a schedule optimization at *task-level*. All these approaches provide good results for subsystems, but the problem complexity does not allow their application to an entire state-of-the-art in-vehicle network due to their restricted scalability. At the same time, while these approaches might be used incrementally, they do not allow an integration of already existing schedules to an entire system. As a remedy, we present an approach which is capable of determining a global schedule for an entire in-vehicle network using local schedules of subsystems (the local schedules might be determined with

This publication is made possible by the Singapore National Research Foundation under its Campus for Research Excellence And Technological Enterprise (CREATE) programme.

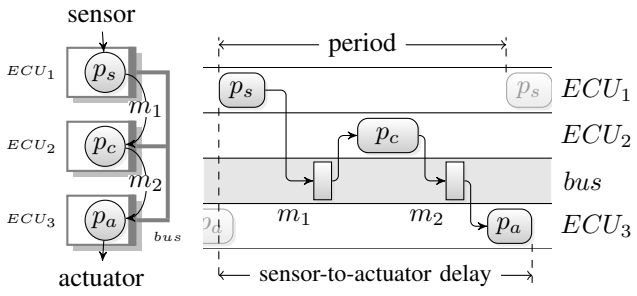


Fig. 1. Example topology of an actuator controlled over a communication bus based on sensor data, and a corresponding process scheduling. p_s, p_c and p_a are the sensor, the controller and the actuator process, respectively, which communicate over a bus with the messages m_1 and m_2.

the approaches in [8] and [15]). Such schedule integration, has not been sufficiently studied in scientific literature, despite being of great importance to the industry.

III. Framework

In the following, a general framework for the integration of synchronous time-triggered systems is presented. A specific implementation for the FlexRay network is presented in the next section.

A. Time-Triggered Synchronous Scheduling

The framework presented in this paper is based on time-triggered synchronous scheduling. For synchronous scheduling, all processes are executed based on a schedule that is triggered using the global time. In order to ensure that all network participants or Electronic Control Units (ECUs), respectively, are aware of the global time, bus protocols like FlexRay or Ethernet PTP become necessary to ensure the clock synchronization. The basis for synchronous scheduling is a predefined schedule which specifies at which times each process is executed periodically, guaranteeing all necessary resources are available during runtime.

In state-of-the-art vehicles, various applications are implemented over several spatially distributed ECUs. An example of such a distributed system is given in Figure 1, including the schedule that ensures the correct execution of the application. The schedule defines the exact timing of each of the processes on the ECUs and the message transmission times for the bus. Very often several applications are developed in different clusters based on their functionality with distinct requirements regarding performance, reliability, and safety. In the automotive area, these clusters might comprise functions in the domains of *body*, *chassis*, *information*, etc. In this paper, we assume that the clusters are predefined but there might exist also graph-based approaches to determine clusters of an entire system by using techniques like minimal cuts. While most ECUs are not shared between clusters, there are some hardware components like buses or gateways that are shared to reduce the costs of the entire system. An example of an architecture that shares components between clusters is illustrated in Figure 2(a). For the sake of simplicity, each cluster is defined by a single application in this example. To define a schedule for the complete system becomes a computationally intensive task and common methods like [8] and [15] are not capable of delivering a feasible solution within a reasonable amount of time due to scalability issues. Moreover, it is common practice to develop each cluster separately and, thus, also to determine the schedules separately. In a final step, an integration of the subsystems becomes necessary such that for a large time-triggered system, a schedule integration has to be performed. To avoid conflicts in the resulting schedule, we propose two techniques as illustrated in Figure 2(b) and 2(c) which are described formally in the following.

To determine whether a process p utilizes a resource c at a specific time t, the following function is defined:

$$c(p,t) = \begin{cases} 1 & p \text{ utilizes } c \text{ at } t \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

A process is generally executed periodically, starting from the start time s_p with a period h_p. Without loss of generality, we assume in the following that the processes are non-preemptive. Hence, the task p utilizes c for all t:

$$s_p \leq t \leq s_p + e_p \tag{2}$$

$$s_p = \Delta t + n \cdot h_p,\ n \in \mathbb{N}_0 \tag{3}$$

Where h_p is the period in which p is executed and e_p the execution time. Δt indicates the temporal offset.

For the schedule integration, it has to be ensured that no resource is utilized by two processes at the same time. If two clusters D and $\tilde{D}$ share the same resource c with their processes $p \in D$ and $\tilde{p} \in \tilde{D}$, their schedules might be merged if the following requirement is always fulfilled:

$$\forall t:\ c(p,t) + c(\tilde{p},t) \leq 1 \tag{4}$$

Hence p and $\tilde{p}$ must never be executed at the same time.

B. Schedule Integration

In many cases a direct schedule integration will lead to conflicts by violating Equation (4). In this paper, we present a schedule integration approach to merge existing schedules of subsystems which share resources by shifting the cluster schedules and process start times, respectively. These two techniques increase the chance to find a feasible schedule for the entire system. Thus, also the complexity of the scheduling is reduced by a divide-and-conquer approach. The two techniques are described in the following for the general case.

One way to merge the schedules of the clusters D and $\tilde{D}$ is to shift the cluster schedules by a constant time which is denoted as o_D and $o_{\tilde{D}}$, respectively. Shifting the schedule by a constant time does not influence the behavior of the subsystems since the shift is performed for the entire cluster, i.e., all processes within the cluster. This approach is illustrated in Figure 2(b). Thus, the requirement in Equation (4) is extended as follows:

$$\forall t:\ c(p,t+o_D) + c(\tilde{p},t+o_{\tilde{D}}) \leq 1 \tag{5}$$

In this case, suitable offsets o_D and $o_{\tilde{D}}$ have to be found. If a cluster D is shifted by o_D, the previously defined execution times for all messages and tasks are shifted by o_D.

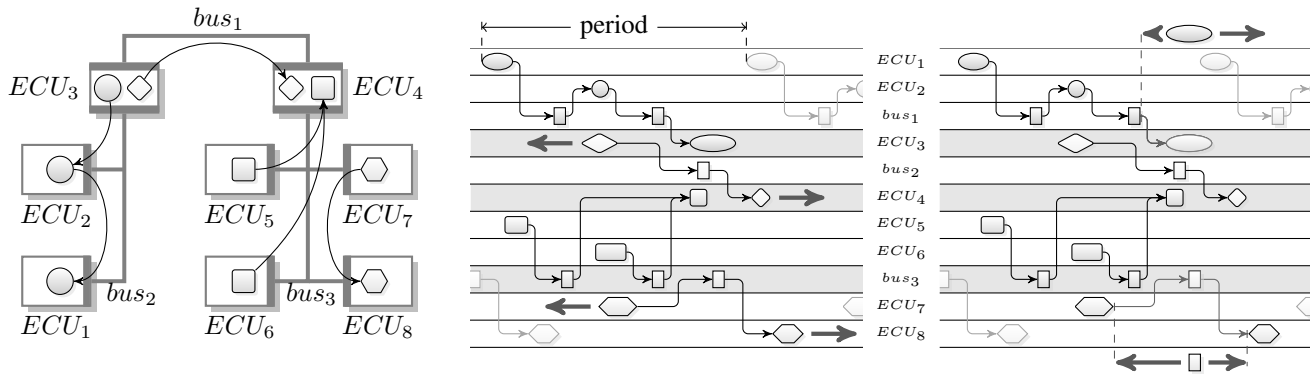
The second technique to obtain a feasible schedule for the entire system is by shifting the start time of a single process within certain bounds. This approach is illustrated in Figure 2(c). These bounds for a process p are defined by the variable o_p which is constrained as follows:

$$f_{\check{p}} \leq t + o_p \leq s_{\hat{p}} \tag{6}$$

Here, $f_{\check{p}}$ is the maximal finish time of any preceding process $\check{p} \in D$ and $s_{\hat{p}}$ is the minimal starting time of any succeeding process $\hat{p} \in D$. Thus, the requirement from Equation (5) is extended as follows:

$$\forall t:\ c(p,t+o_D+o_p) + c(\tilde{p},t+o_{\tilde{D}}+o_{\tilde{p}}) \leq 1 \tag{7}$$

In this case, additionally suitable offsets o_p and $o_{\tilde{p}}$ have to be found.



(a) architecture

(b) Shifting of an entire cluster.

(c) Shifting of a single process.

Fig. 2. The illustration shows an architecture (a) consisting of two subnetworks connected with the backbone bus_1. Four clusters (◯, ◇, □, ⬡) share the resources ECU_3, ECU_4, and bus_3. To create a feasible schedule for the architecture, entire cluster schedules can be shifted such that a feasible global schedule is obtained (b). Additionally, cluster schedules might be modified by shifting a single process within the time frame defined by sending and receiving processes (c).

On the one hand, both approaches help to find a feasible schedule for the entire system. On the other hand, the search space might become very large, in particular, with the process shifting approach. In the following, we show an efficient implementation of this framework for a FlexRay system.

IV. FlexRay Schedule Integration

This section presents a schedule integration approach to merge existing schedules of independent clusters communicating over the FlexRay bus. Hence, the shared resource in this case is the FlexRay bus. We first give a brief introduction to the FlexRay protocol, before we present a schedule integration using an ILP approach. Finally, we present methods that effectively reduce the search space and complexity.

A. FlexRay Protocol

In the following, a basic introduction to the FlexRay protocol is given, using the illustration in Figure 3(a). A FlexRay schedule consists of C communication cycles which are repeated after all cycles n_{cycles} have been scheduled. A FlexRay cycle consists of a static segment for time-triggered communication and a dynamic segment for event-triggered communication. In this paper, we focus on time-triggered sytems and, therefore, only use the static segment for scheduling. As the dynamic segment remains unused, it is considered as a time window where no messages can be scheduled. The FlexRay protocol splits the static segment into n_{static} slots that might be used by the ECUs to transfer messages or frames, respectively. Here, a frame is defined as a set of messages transfered by the applications. The local schedules and, thus, frames might be determined with the approaches in [8] and [15].

Our framework supports both common versions of FlexRay 2.1 and 3.0. An important difference between both versions is the requirement regarding the assignment of slots to ECUs. In FlexRay 2.1 each slot is exclusively assigned to one ECU for all cycles as illustrated in 3(b). In contrast, in FlexRay 3.0 each slot might be assigned to different ECUs, depending on the current cycle. As a result, a slot might be shared by different ECUs which might send different frames for each cycle as illustrated in 3(b). Due to the higher flexibility of FlexRay 3.0, a much higher utilization might be reached, but also the complexity of the schedule integration problem becomes significantly higher.

B. Schedule Integration Model

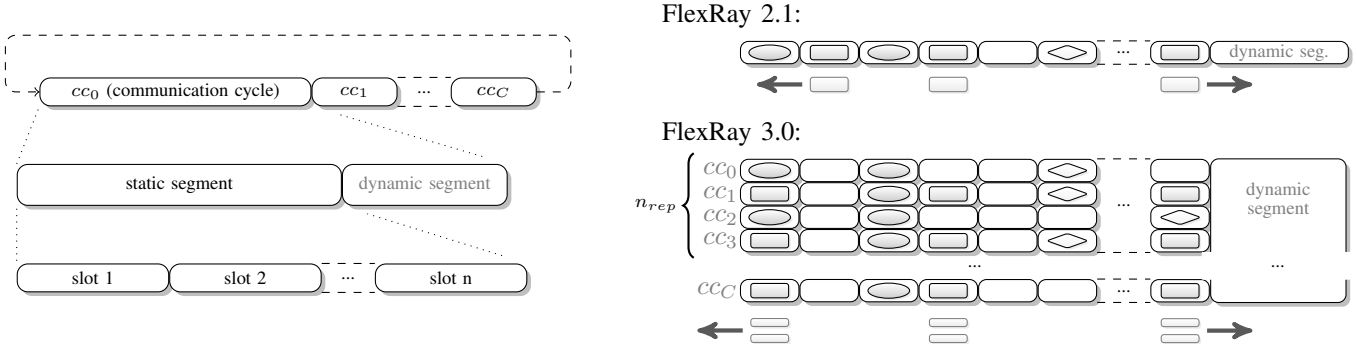
In the following, we introduce the schedule integration approach for FlexRay which might be applied to both versions 2.1 and 3.0. As a result of the slot structure, a cluster or process can only be shifted in discrete steps defined by the static slot duration. One cycle can then be split into $n_{all} = \frac{t_{cyc}}{t_{sta}} \cdot n_{static}$ steps for shifting, where t_{cyc} is the duration of one cycle and t_{sta} is the duration of the static segment. In FlexRay 3.0, additionally the shifting might be done over different cycles as explained later. As a result of shifting in discrete steps, the schedule integration problem becomes discrete, leading to a significant search space reduction in contrast to a continuous problem. Therefore, introducing discrete time steps might also be a reasonable method to reduce the complexity of schedule integration for a system that is not based on time steps, as discussed in [9].

In the general case of FlexRay, a frame is sent in a specific slot at specific cycles. As a result, the frame f does not only have a slot $s(f) \in \{0, .., n_{static} - 1\}$ defined in the cluster schedule, but also its occurrences in the cycles $c(f) \subseteq \{0, .., n_{cycles} - 1\}$. In general, each cluster might now be shifted by $n_{all} \cdot n_{cycles}$ steps. To reduce this number and, thus, the search space, the repetitions of f in the cycles have to be investigated. The goal is to find a minimal rep_f such that the following requirement is fulfilled:

$$\forall c \in c(f)\ \exists \tilde{c} \in c(f) :\ \tilde{c} = c \% rep_f \tag{8}$$

Thus, shifting frame f more than rep_f cycles does not have any effect and becomes redundant. Correspondingly to the frame shifting, $rep(D)$ is the maximal shifting that is necessary for a cluster D and is defined as $rep(D) = \underset{f \in D}{lcm}(rep_f)$ where lcm determines the least common multiple. As a result, each cluster D might be shifted by the number of steps $o(D) = \{0, .., rep(D) \cdot n_{all} - 1\}$. Additionally, the schedule repetition is defined as $n_{rep} = \underset{D \in \mathcal{D}}{lcm}(rep(D))$. Note that for FlexRay 2.1, a slot can be used by only a single ECU and, thus, the repetition of frames should be 1 to avoid sharing slots between different ECUs. This also leads to $n_{rep} = 1$ as illustrated in Figure 3(b). Each frame might additionally be shifted by its variance $v(f) \in \mathbb{Z}$ defined in slots. The variance is defined by the bounds of sending task and receiving tasks, respectively (see Equation 6).

In the following, we define an ILP formulation to find a



(a) FlexRay protocol.

(b) Schedule integration for FlexRay 2.1 and FlexRay 3.0, respectively.

Fig. 3. The illustration gives an introduction to the FlexRay protocol and the basic problem structure including a valid schedule integration for both FlexRay 2.1 and FlexRay 3.0, if three cluster schedules (⬭, ◇, ▭) should be integrated into a global schedule.

feasible schedule for the entire system. The formulation uses the following variables:

- $\mathbf{o_{D,i}}$ is a binary variable that defines whether cluster $D \in \mathcal{D}$ is shifted by offset $i \in o(D)$ (1) or not (0)
- $\mathbf{o_{f,i}}$ is a binary variable that defines whether frame f is shifted by the variance $i \in v(f)$ (1) or not (0)
- $\mathbf{x_{D,f,c}}$ is a binary variable for each cluster D which defines whether a specific slot in a specific cycle denoted as c ($c \in \{0, ..., n_{rep} \cdot n_{all} - 1\}$) is used by frame $f \in D$ (1) or not (0)

The constraints for schedule integration are then defined as follows:
$\forall D \in \mathcal{D}$:

$$\sum_{i \in o(D)} \mathbf{o_{D,i}} = 1 \tag{9}$$

$\forall D \in \mathcal{D}, f \in D$:

$$\sum_{i \in v(f)} \mathbf{o_{f,i}} = 1 \tag{10}$$

$\forall D \in \mathcal{D}, f \in D, i \in v(f), j \in o(D)$:
$\exists (s(f) + j + i)\%n_{all} \geq n_{static}$:

$$\mathbf{o_{f,i}} + \mathbf{o_{D,j}} \leq 1 \tag{11}$$

$\forall D \in \mathcal{D}, f \in D, i \in v(f), j \in o(D)$,
$(s(f) + j + i)\%n_{all} < n_{static} : c \in c(f)$,
$\tilde{c} = (((s(f) + j + i) + c \cdot n_{all})\%(n_{all} \cdot n_{cycles}))$:
$\exists \tilde{c} < n_{all} \cdot n_{rep}$:

$$\mathbf{o_{f,i}} + \mathbf{o_{D,j}} - \mathbf{x_{D,f,\tilde{c}}} \leq 1 \tag{12}$$

$c \in \{0, .., n_{rep} \cdot n_{all} - 1\}, c\%n_{all} < n_{static}$:

$$\sum_{D \in \mathcal{D}} \sum_{f \in D} \mathbf{x_{D,f,c}} \leq 1 \tag{13}$$

The first Constraint (9) defines that each cluster has exactly one offset while correspondingly Constraint (10) states that each frame has exactly one offset. Constraint (11) ensures that no frames are placed inside the dynamic segment. To ensure that each slot is only occupied once at a specific cycle, the $\mathbf{x_{D,f,c}}$ variables are introduced and Constraint (12) sets the values depending on the cluster offset and frame offset. Finally, Constraint (13) defines that each slot at a specific cycle can only be reserved once. To reduce the problem size, instead of calculating the problem for all n_{cycles} of the schedule, we only calculate the problem for the non repetitive part $n_{rep} \leq n_{cycles}$. Hence, if a solution exists for n_{cycles} cycles, it can be found for n_{rep} cycles.

C. Search Space Reduction

The complexity of the proposed ILP formulation might become an obstacle with a growing number of clusters and frames. As a remedy, we propose additional techniques which allow a further reduction of the problem complexity by using domain specific knowledge.

Filter offsets. The ILP might further be improved by eliminating unsatisfiable offsets $o(D)$. As the offsets $o(D)$ for a cluster can be defined over several cycles, some offsets might shift frames to an invalid position, i.e., outside the static segment. The following equation determines $\tilde{o}(D) \subseteq o(D)$ which only contains cluster offsets that do have at least a single mapping of all frames to the static segment:
$\forall i \in o(D), f \in D$:
$\exists j \in v(f), i + j + s(f)\%n_{all} < n_{static}$:

$$i \in \tilde{o}_f(D) \tag{14}$$

$$\tilde{o}(D) = \bigcap_{\forall f \in D} \tilde{o}_f(D) \tag{15}$$

Equation (14) determines the satisfiable offsets for each frame $\tilde{o}_f(D)$. $\tilde{o}(D)$ is then defined as the common satisfiable offsets in Equation (15). Using $\tilde{o}(D)$ instead of $o(D)$ in the model, significantly reduces the number of constraints and the problem size.

Cycle breaking. The search space can further be reduced by a cycle breaking technique. If two frames $f, \tilde{f} \in D$ that satisfy $c(f) = c(\tilde{f})$ are close to each other, the frame variances $v(f)$ and $v(\tilde{f})$ might allow to swap their position. However, swapping the position of two frames within a cluster does not lead to additional feasible solutions. For instance, if a cluster has two frames f_1 and f_2 with $s(f_1) = 3$, $s(f_2) = 4$, $v(f_1) = v(f_2) = \{-1, 0, 1\}$, $c(f_1) = c(f_2)$, then $\mathbf{o_{f_1,1}} = \mathbf{o_{f_2,-1}} = 1$ equals the same allocation of slots like $\mathbf{o_{f_1,0}} = \mathbf{o_{f_2,0}} = 1$ where both slots 3 and 4 are allocated in the cycles $c(f_1)$. Breaking these cycles reduces the search space and decreases the runtime. The following additional constraint results in cycle breaking:
$\forall D \in \mathcal{D}, f, \tilde{f} \in D, f \neq \tilde{f}, c(f) = c(\tilde{f}), s(f) < s(\tilde{f})$,
$I = \{s(f) + i | i \in v(f)\} \cap \{s(\tilde{f}) + j | j \in v(\tilde{f})\}$,
$i, j \in I, i > j$:

$$\mathbf{o_{f,i-s(f)}} + \mathbf{o_{\tilde{f},j-s(\tilde{f})}} \leq 1 \tag{16}$$

The constraint is based on the set I which defines all slots both f as well as $\tilde{f}$ can be shifted to, i.e., the slots which might lead to swapping. It defines that the frames $f, \tilde{f}$, with

$s(f) < s(\tilde{f})$, cannot be shifted to any slot positions in I such that $s(f) + i > s(\tilde{f}) + j, i \in v(f), j \in v(\tilde{f})$. Note that cycle breaking effectively accelerates the ILP solving without affecting the feasibility of the schedule integration.

V. Experimental Results

This section presents various test cases and case studies to give evidence of the capabilities and the flexibility of our framework. We first present two realistic case studies, followed by the results of a scalability analysis consisting of 5400 synthetic test cases. All simulations have been carried out on an Intel Xeon 3.2 GHz Quad Core with 12GB RAM. We used the 0-1 ILP solver Sat4J in version 2.3.1 [5] for solving the ILP formulations. This solver is based on a backtracking search and is faster for the considered problems than a common ILP solver like CPLEX [4]. Note that the schedule is obtained at design time such that runtimes of several minutes and even hours are still acceptable.

A. Realistic Case Study

To show the applicability of our framework, we present the simulation results for two realistic in-vehicle networks with FlexRay as the central backbone bus. These case studies are outlined in Table I and II. The FlexRay bus is configured with a cycle duration of 5ms and 60 slots in the static segment. The payload of one slot is 40 bytes and the duration of a slot 0.0666ms.

First case study. For the first test case, we scheduled 4 independent clusters with 64 processes and 42 frames. All clusters together require a total of 42 slots with a utilization of 29% which is defined by the allocated slots in each cycle. The frames are distributed over the static segment for all clusters and for FlexRay 2.1 slot shifting is required to combine the clusters. Therefore, a manual approach is impractical and an automated approach required. Using FlexRay 2.1 where a slot cannot be shared between ECUs, the problem formulation results in 5207 variables and 146926 constraints and is solved in $163ms$. For FlexRay 3.0, the problem formulation results in 14895 variables and 1070730 constraints and can be solved in $208ms$. Here, the advantage of FlexRay 3.0 is that slots might be shared and the number of required slots is reduced to 32 compared to 42 for FlexRay 2.1 where the sum of required slots remains the same as for all clusters together.

Second case study. To show the advantages of FlexRay 3.0, we use a larger second case study. Our system consists of 6 different clusters with 116 processes and 69 frames (see Table II for details), occupying a total of 66 slots in sum with a total utilization of 48%. As the static segment only consists of 60 slots, this example can only be scheduled with FlexRay 3.0 since FlexRay 2.1 does not allow slot sharing. The ILP consists of 24713 variables and 3121097 constraints and is solved in 5.16 seconds. The integrated schedule occupies 48 slots. These results show the capabilities of our framework on a large realistic case study. Note that the variance is very high and leads to a relatively large search space. Constraining the variance may be a valid approach to reduce the runtime further as long as the resulting solution remains feasible.

B. Scalability Analysis

To show the scalability of our framework for the FlexRay bus, we present the results of a set of synthetic test cases. The test cases have been created by a schedule generator based on 5 parameters:

- the number of clusters $|\mathcal{D}| \in \{2, ..., 10\}$ to be merged
- the total utilization $U \in \{0.3, 0.4, ..., 0.8\}$ of the slots over all cycles
- the number of slots $n_{static} \in \{30, 40, ..., 70\}$ in the static segment
- the maximal repetition $rep_{max} \in \{1, 2, 4, 8\}$ that defines the upper bound for the rep_f for each frame f
- the maximal variance per frame v_{max} that restricts the cardinality of $v(f)$ for each frame f

TABLE I
Basic information of the first case study.

cluster D	ECUs	frames $\|D\|$	repetitions rep_f	variance $v(f)$
body	2	9	2,4	3-217
chassis	3	12	2,4	0-85
information	2	11	4,8	9-278
electric	3	10	1,2	0-55
total	10	42	1,2,4,8	0-278

TABLE II
Basic information of the second case study.

cluster D	ECUs	frames $\|D\|$	repetitions rep_f	variance $v(f)$
body	2	9	2,4	15-240
chassis	3	12	2,4	0-91
information	2	15	4,8	29-368
electric	3	10	1,2	0-55
safety	3	10	1,2	3-41
telematics	3	13	4	0-161
total	16	69	1,2,4,8	0-368

For each configuration of parameters, a random test case was generated which leads to 5400 highly heterogeneous test cases. Here, we assumed the FlexRay bus to be configured with a cycle length of 5ms and a length of 4ms for the static segment.

In order to give evidence of the scalability of our approach, the runtime of the solver to obtain a feasible schedule or to prove that there exists no feasible schedule was determined. The timeout for the solver was set to 3600 seconds. Here, we considered only the FlexRay 3.0 approach since FlexRay 2.1 shows a much better scalability due to significantly smaller search space. Figure 4 illustrates the runtime of all test cases depending on the parameters. For 78.2% of the test cases the problem is solved in less than 1 second and for $94,2\%$ in less than 1 minute. This is also reflected in the low average runtime in Figure 4. It can be seen that the runtime increases with an increased complexity. In case any of the parameters $|\mathcal{D}|$, n_{static}, v_{max}, or rep_{max} is increased, the search space becomes larger and, thus, the runtime grows in average. On the other hand, for a larger utilization U, the number of feasible solutions is reduced such that the runtime also grows in average. Only 0.33% of all test cases did not obtain a solution within 60 minutes which are then assumed to be infeasible.

Figure 5 illustrates the ratio of feasibility based on the parameters. An increasing utilization U significantly reduces the probability to obtain a feasible schedule. At the same time, an increased v_{max} that enables frame shifting proves to be a suitable method to obtain feasible schedules. The higher feasibility with a growing number of clusters $|\mathcal{D}|$ is explained by the growing search space under the same utilization, etc. A growing n_{static} and rep_{max} lead to less feasible solutions

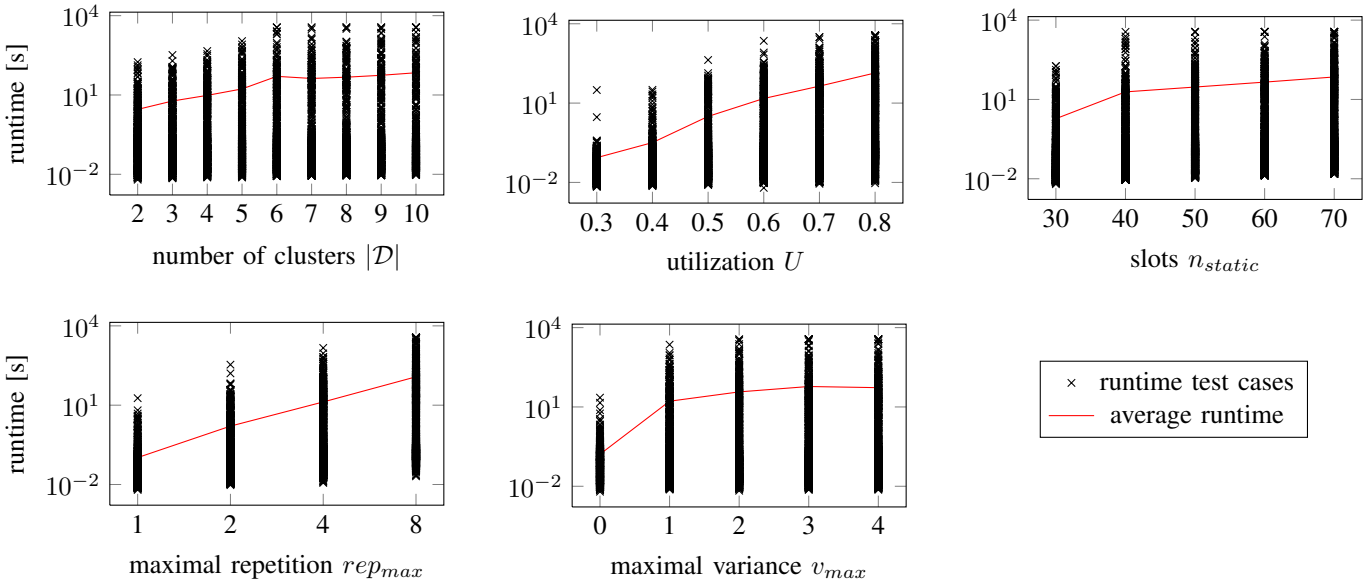


Fig. 4. Illustration of the runtime for 5400 synthetic test cases. The plots indicate how different parameters influence the runtime of our test cases.

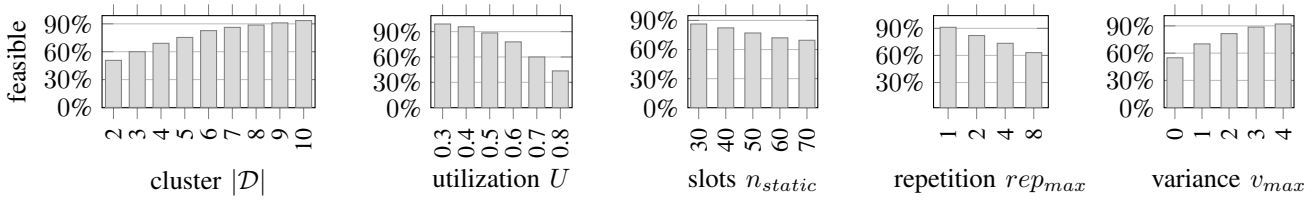


Fig. 5. Illustration of the feasibility for 5400 synthetic test cases. The charts indicate how different parameters have an influence on the feasibility of our test cases.

due to the higher probability of conflicting frames. In total 77.4% of all test cases could be successfully integrated for FlexRay 3.0. This is a very good value, considering that we only considered a small or no variance for each cluster and a high utilization. Note that if the schedule integration is done for the same test cases with FlexRay 2.1 instead, only 56.6% of all test cases could be integrated. This gives evidence of the advantage of FlexRay 3.0 over 2.1 to obtain feasible schedules.

VI. Concluding Remarks

This paper presents a schedule integration framework for time-triggered synchronous systems in the automotive domain. Our approach is capable of merging existing schedules of independent clusters to a global schedule while preserving the basic structure of the cluster schedules. This approach is in accordance with the common design process of automotive systems where different functions and clusters are developed independently and are combined in a later stage. Our framework therefore aims at assisting in this cluster integration process and at reducing the complexity of scheduling a large time-triggered system. As a proof-of-concept, we propose ILP formulations with FlexRay as the central communication bus. The framework might be also extended to other time-triggered systems and protocols. Two case studies give evidence of the applicability of the proposed framework while a scalability analysis with 5400 test cases shows the flexibility and efficiency of our schedule integration approach.

References

[1] IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems, 2008.

[2] OPEN Alliance Special Interest Group, 2011. http://www.opensig.org/.

[3] J. Berwanger, M. Peteratzinger, and A. Schedl. FlexRay startet durch - FlexRay-Bordnetz fuer Fahrdynamik und Fahrerassistenzsysteme (in German). *Elektronik automotive: Sonderausgabe 7er BMW*, 2008.

[4] ILOG. CPLEX. http://www.ilog.com/products/cplex/.

[5] D. Le Berre and A. Parrain. The Sat4j library, release 2.2. *Journal on Satisfiability, Boolean Modeling and Computation*, 7:59–64, 2010.

[6] H. Lim, D. Herrscher, and M. Waltl. Performance analysis of the IEEE 802.1 ethernet audio/video bridging standard. In *Proc. of SIMUTOOLS 2012*, pages 27–36, 2012.

[7] H. Lim, L. Völker, and D. Herrscher. Challenges in a future IP/ Ethernet-based in-car network for real-time applications. In *Proc. of DAC 2011*, pages 7–12, 2011.

[8] M. Lukasiewycz, R. Schneider, D. Goswami, and S. Chakraborty. Modular Scheduling of Distributed Heterogeneous Time-Triggered Automotive Systems. In *Proc. of ASPDAC 2012*, pages 665–670, 2012.

[9] A. Mok and W. Wang. Window-constrained real-time periodic task scheduling. In *Proc. of RTSS 2001*, pages 15–24, 2001.

[10] T. Pop, P. Pop, P. Eles, Z. Peng, and A. Andrei. Timing analysis of the FlexRay communication protocol. *Real-Time Systems*, 39:205–235, 2008.

[11] W. Steiner. An Evaluation of SMT-Based Schedule Synthesis for Time-Triggered Multi-hop Networks. In *Proc. of RTSS 2010*, pages 375–384, 2010.

[12] K. W. Tindell, H. Hansson, and A. Wellings. Analysing real-time communications: Controller Area Network (CAN). In *Proc. of RTSS 1994*, pages 259–263, 1994.

[13] TTA Group. TTEthernet Specification, 2008. http://www.ttagroup.org.

[14] TTTech Computertechnik AG. Advanced Electro-Mechanical Braking System: Delphis Brake-by-Wire Concept (Brochure). 2006.

[15] H. Zeng, W. Zheng, M. Di Natale, A. Ghosal, P. Giusto, and A. Sangiovanni-Vincentelli. Scheduling the FlexRay bus using optimization techniques. In *Proc. of DAC 2009*, pages 874–877, 2009.