FlexRay Switch Scheduling - A Networking Concept for Electric Vehicles

Martin Lukasiewycz, Samarjit Chakraborty
TU Munich, Germany
{martin.lukasiewycz, samarjit.chakraborty}@rcs.ei.tum.de

Paul Milbredt
AUDI AG, Germany
paul.milbredt@audi.de

***Abstract*—It is projected that the communication data volume in electric vehicles will significantly increase compared to state-of-the-art vehicles due to additional functionalities like x-by-wire and safety functions. This paper presents a networking concept for electric vehicles to cope with the high data volume in cases where a single FlexRay bus is not sufficient. We present a FlexRay switch concept that is capable of increasing the effective bandwidth and improving the safety of existing FlexRay buses. A prototype FPGA implementation shows the feasibility of our approach. Further, a scheduling approach for the FlexRay switch that obtains the optimal results based on Integer Linear Programming (ILP) is presented. Since the ILP approach becomes intractable for real-world problems, we present a heuristic three-step approach that determines the branches of the network, performs a local scheduling for each node, and finally assembles the local schedules into a global schedule. Test cases and an entire realistic in-vehicle network are used to emphasize the benefits of the proposed approach.**

I. Introduction

Next generation electric vehicles will radically change the design paradigms in the automotive network domain. For the sake of simplicity, the legacy Control Area Network (CAN) [1] buses shall be replaced by a homogeneous core network. A strong candidate for this network is the standardized FlexRay protocol [2] that is tailored to the automotive domain. However, even current top-of-the-range cars already contain one FlexRay bus and several CAN buses such that the entire communication data would exhaust the capacity of a single FlexRay bus. One natural answer for this problem would be a gateway that interconnects multiple FlexRay buses and, thus, increases the bandwidth of the network. However, this solution comes at a high price: The gateway leads to an additional delay since multiple FlexRay buses cannot be synchronized and messages have to be buffered. This approach goes against the benefits of the time-triggered FlexRay bus and implementing distributed functions with strict real-time requirements might be hampered. As a remedy, we propose a FlexRay *switch* and a scheduling approach that is capable of increasing the effective bandwidth without introducing additional delays to the communication.

FlexRay Protocol: The amount of communication in vehicle networks is constantly increasing due to the growing number of safety, comfort, and entertainment functions. Most recently, several car manufacturers introduced the first FlexRay buses in production vehicles to cope with the high data volume [3], [4]. The *FlexRay* communication system [2] has been developed by an industrial consortium to cope with the ever increasing data volume and real-time communication requirements of state-of-the-art vehicles. It offers a time-triggered and an event-triggered segment, a bandwidth of 10 Mbit/s, and supports different topologies, i.e., linear bus, star and hybrid topologies. Thus, FlexRay is the prospective automotive standard communication system.

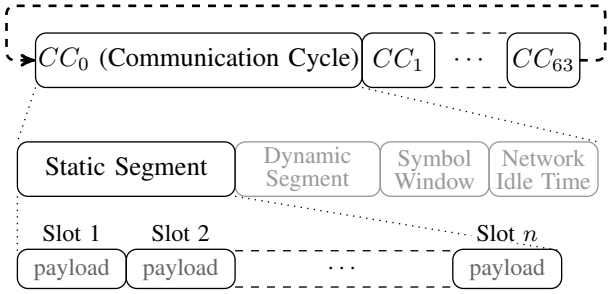

Fig. 1. The FlexRay communication protocol. Each communication cycle contains a static segment that comprises a set of equally sized slots.

The FlexRay communication is organized in 64 *cycles*, as illustrated in Figure 1. Each cycle is divided into four segments of configurable duration: (1) The *static segment* enabling a guaranteed real-time transmission of critical data, (2) the *dynamic segment* for low-priority and event-triggered data, (3) the *symbol window* used to transmit special symbols, and (4) the *network idle time* used to perform a clock synchronization. The focus of this paper is put on scheduling the static segment. The static segment is made up of n equally sized *slots* where each slot is uniquely assigned to one *node* (or none). One node, however, may occupy more than one slot. Each slot consists of a header and trailer segment and a payload segment that is statically configured to carry between 0 and 254 bytes. By a predefined schedule, each slot is filled with *messages* (also referred to as *frames* or *protocol data units*) of the applications. The size of each message is fixed and given in bytes as the basic unit. The AUTomotive Open System ARchitecture (AUTOSAR) FlexRay Interface Specification [5] suggests a multiplexing technique using the communication cycles to increase the utilization of the bus.

Contributions of the Paper: The proposed FlexRay switch is a central hardware device that is capable of connecting multiple FlexRay branches transparently, i.e., a modification of the FlexRay bus participants is not necessary. In comparison with the interconnection of buses using a gateway, the switch does not require data buffering and also does not introduce any additional delay to the communication. The proposed switch separates communication branches such that each FlexRay slot may be used by different nodes concurrently on distinct branches. Hence, this approach increases the effective bandwidth of the bus. Moreover, the switch might serve as a central bus guardian, thereby increasing the safety of the network.

As a proof-of-concept, the switch is implemented on an Field Programmable Gate Array (FPGA) platform. Experiments show that the prototype meets the FlexRay Electric Physical Layer (EPL) Specification [6].

To capitalize on the advantages of the switched FlexRay, an

978-3-9810801-7-9/DATE11/©2011 EDAA

appropriate topology and scheduling determination becomes necessary. In this paper, we present an ILP approach that is capable of determining the optimal network topology and scheduling. Due to the exponential nature of ILP formulations, a heuristic approach becomes necessary to handle problems of realistic size. This heuristic approach is performed in three steps. The first step determines the network topology, i.e., the branches. In the second step, the scheduling for each node is determined. In the third and last step, the schedules of each node are assembled into a global schedule.

Test cases of various sizes show the benefits of the proposed approach. Increasing the number of branches, significantly decreases the number of used slots and, thus, boosts the effective bandwidth. A scenario of a migration from a realistic in-vehicle network consisting of four CAN buses and a FlexRay bus to a homogeneous FlexRay network is investigated. While a single FlexRay bus does not provide enough bandwidth, the switch concept with the proposed scheduling approach allows to determine a feasible schedule.

Organization of the paper: The remainder of the paper is organized as follows: Section II discusses related work. The FlexRay switch concept and the scheduling problem definition are presented in Section III. Section IV outlines the FPGA implementation of the switch. In Section V, an exact scheduling approach using ILP is proposed as well as an heuristic three-step approach. Experimental results including test cases and a realistic case study are presented in Section VI, before concluding remarks are made in Section VII.

II. Related Work

Today, the most prevailing bus system for in-vehicle communication is the event-triggered CAN [1] which is restricted to a bit rate of 1Mbit/s only. A significantly higher bandwidth with 10Mbit/s is provided by the FlexRay bus [2]. Moreover, the FlexRay protocol allows a time-triggered communication which is necessary for several control functions with real-time requirements. In a dual channel mode, the FlexRay protocol even allows to double the bandwidth. However, in this case the second channel cannot be used for error correction. On the other hand, Ethernet [7] might be an alternative to cope with the high data volume of upcoming electric vehicles. However, to ensure real-time properties, special implementations of the Ethernet protocol become necessary [8]. Moreover, the electromagnetic compatibility for Ethernet in the automotive domain is not as well researched as the FlexRay bus [6] and hence Ethernet is only considered for non-critical multimedia applications as a compensation for the cost-intensive Media Oriented Systems Transport (MOST) [9]. Due to the standardization of the FlexRay bus for the automotive domain it is a strong candidate for upcoming (homogeneous) in-vehicle networks that require a significantly higher bandwidth.

A major challenge for the proposed FlexRay switch concept is the determination of an efficient schedule. In this paper, we focus on scheduling the static segment. An approach that optimizes the static segment with a Genetic Algorithm (GA) is proposed in [10]. The work in [11] introduces an ILP approach for a proposed custom software architecture. In [12], the authors present a Mixed Integer Linear Programming (MILP) approach for scheduling messages and tasks in a synchronous architecture. Recent work on scheduling the static segment is proposed in [13] and [14] which consider the optimization of the schedule with respect to cycle multiplexing. However, these papers do not consider scheduling multiple FlexRay buses and, therefore, are not applicable to the scheduling problem that originates from using the FlexRay switch as proposed in the work at hand.

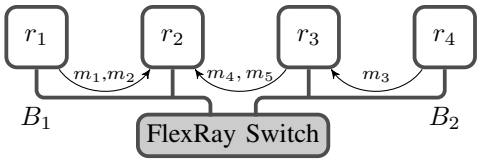

Fig. 2. FlexRay network with four nodes $R = \{r_1, r_2, r_3, r_4\}$ and a switch separating the bus into two branches $B_1 = \{r_1, r_2\}$ and $B_2 = \{r_3, r_4\}$. The messages $M = \{m_1, m_2, m_3, m_4, m_5\}$ are scheduled on the network.

(a) Scheduling *without* FlexRay Switch

(b) Scheduling *with* FlexRay Switch

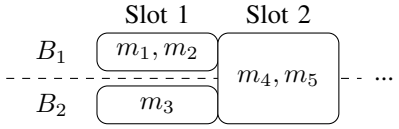

Fig. 3. Schedules for the network in Figure 2 resulting in (a) three used slots without a switch and (b) two used slots by using the proposed FlexRay switch.

III. Switch Concept

In the following, the FlexRay switch concept is presented, using a simple example. Additionally, the resulting scheduling problem is introduced in a formal description.

A. Example

To illustrate the benefits of the FlexRay switch in terms of an increased bandwidth, Figure 2 is introduced as an example architecture. Here, four nodes are connected to a FlexRay bus that is separated into two branches using the proposed FlexRay switch. Assuming that the messages m_1,m_2 and m_4,m_5 can be sent in a single slot (see Section I for an introduction of the FlexRay protocol), the schedule without the switch requires three slots as illustrated in Figure 3(a). Using the switch, the number of used slots is reduced to two by scheduling the messages m_1, m_2 and m_3 in parallel as illustrated in Figure 3(b).

This approach does not require any modifications of the communication controllers of the FlexRay participants since each branch has a local schedule that entirely meets the FlexRay specification. Compared to a solution using a gateway, the switch does not induce any additional delay. Moreover, a gateway requires buffers to store and forward messages. Hence, the FlexRay switch is a simple and cheap solution that increases the effective bandwidth while ensuring the real-time properties of the FlexRay bus. Note that a branch might be cascaded using an active star. This becomes necessary if a branch consists of more than eight nodes as required in the FlexRay specification [2].

B. Problem Formulation

To benefit from the FlexRay switch, an appropriate topology and a schedule have to be determined. The system requirements are defined as a set of communicating nodes R and a set of messages M:

- The network consists of a set of communicating FlexRay nodes R. A FlexRay node $r \in R$ might be an Electronic Control Unit (ECU) or a gateway to other buses.
- Each message $m \in M$ is defined by its size $w_m \in \mathbb{N}$ and repetition $r_m \in \{2^n | n \in \{0, .., 6\}\}$ (the message m is sent each r_m-th cycle). Given the period of the a message p_m, the repetition is determined by $r_m = 2^{\max(\lceil \log_2 \frac{p_m}{p} \rceil, 6)}$ with p being the duration of a communication cycle. Each message has exactly one sender defined by $src : M \to R$ and a set of a receivers $dest : M \to 2^R$.

The topology and schedule are determined by three tasks: (1) The determination of the branches $\mathcal{B}$, (2) the packing of messages in the slots $\mathcal{S}$, and (3) the assignment of a slot-id for each slot σ. Here, $\mathcal{B}$ defines the topology and $(\mathcal{S}, \sigma)$ the schedule:

- Each branch $B \in \mathcal{B}$ is a subset of the nodes. Hence, the following holds:

$$\forall B \in \mathcal{B} : B \subseteq R \tag{1}$$

  On the other hand, all branches are disjoint sets and each resource is on exactly one branch:

$$\forall B, \tilde{B} \in \mathcal{B}, B \neq \tilde{B} : B \cap \tilde{B} = \{\} \tag{2}$$

$$\bigcup_{B \in \mathcal{B}} B = R \tag{3}$$

- Each slot $S \in \mathcal{S}$ contains a set of messages. It holds:

$$\forall S \in \mathcal{S} : S \subseteq M \tag{4}$$

  All messages in a slot have the same sender:

$$\forall S \in \mathcal{S}, m, \tilde{m} \in S : src(m) = src(\tilde{m}) \tag{5}$$

  In this paper, we assume an AUTOSAR [5] compliant packing, i.e., each message is assigned a base cycle b_m and offset x_m: A message is sent in its slot in each cycle $(n \cdot r_m + b_m)\%64$ with $b_m < r_m$ and $n \in \mathbb{N}_0$ at the position x_m which is the offset in bytes in a slot. Depending on the messages in a slot, each slot occupies a set of branches determined by the function $b : \mathcal{S} \to 2^{\mathcal{B}}$ as follows:

$$b(S) = \{B | B \in \mathcal{B} \wedge B \cap \bigcup_{m \in S} src(m) \cup dest(m) \neq \{\}\} \tag{6}$$

- For each slot $S \in \mathcal{S}$ the function $\sigma : \mathcal{S} \to \mathbb{N}$ determines a slot-id (*FRIF_SLOT_ID* in [5]). If two slots share the same slot-id, the occupied branches have to be disjoint:

$$\forall S, \tilde{S} \in \mathcal{S}, S \neq \tilde{S} : \sigma(S) = \sigma(\tilde{S}) \to b(S) \cap b(\tilde{S}) = \{\} \tag{7}$$

The determination of $(\mathcal{B}, \mathcal{S}, \sigma)$ is performed with the goal of using as few slot-ids as possible. This optimization objective arises from the fact that there exists a limited number of slot-ids for each schedule and the number of necessary slot-ids determines the schedulability. This holds, in particular, in a scenario with a high data volume, i.e., in upcoming electric vehicles. Note that additional constraints on $\mathcal{S}$ and σ might become necessary in order to consider synchronized ECUs, see [12].

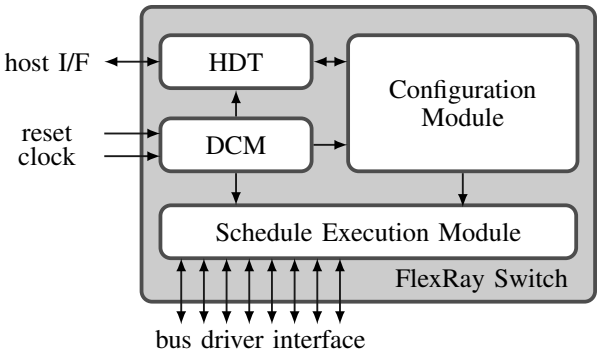

Fig. 4. Block diagram of the FlexRay switch prototype.

IV. Switch Implementation

For the prototype of the FlexRay switch, an automotive testing platform [15] based on an FPGA (Xilinx Spartan 3-1500) with eight FlexRay transceivers was used. The prototype switch was programmed in VHDL at register transfer level (RTL). A detailed introduction is given in [16], in the following the architecture of the switch prototype is outlined.

The behavior of the switch is primarily defined by the configured schedule which controls routing during synchronous operation. In the dynamic segment or asynchronous operation mode, e.g., startup, the switch behaves as a central bus guardian, protecting the branches from faulty nodes. The switch is divided into several parts as illustrated in Figure 4. The switch configuration might be performed at run time by the host via the host interface. For this purpose, the Host Data Translator (HDT) transforms incoming commands and data from the platform-specific host interface to the internally used format. The Digital Clock Management (DCM) generates the required 80 MHz clock for an eight times oversampling of the FlexRay transmission rate of 10 Mbit/s. The configuration of the switch schedule is performed in the Configuration Module. The Schedule Execution Module is responsible for switching the branches. This module has to implement the FlexRay clock synchronization to be aware of the global time and the current slot-id. For each slot-id i, the module has a matrix A_i that determines if a slot is forwarded from branch B to branch $\tilde{B}$. The matrix is defined as follows:

$$A_i(B, \tilde{B}) = \begin{cases} 1 & \text{if } \exists m \in S, S \in \mathcal{S}_i, src(m) \in B, dest(m) \cap \tilde{B} \neq \{\} \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

with

$$\mathcal{S}_i = \{S | S \in \mathcal{S} \wedge \sigma(S) = i\}. \tag{9}$$

A major challenge in the implementation of the switch is the required compliance with the FlexRay EPL [6]: The maximal delay between two communicating nodes is 250ns which equals 2.5bits in case of a bandwidth of 10Mbit/s. Extensive tests revealed that the prototype FPGA FlexRay switch satisfies this constraint, see [16]. Moreover, a commercial off-the-shelf FlexRay switch should have the same behavior as an active star.

V. Switch Scheduling

In this paper, we use the industrially accepted AUTOSAR [5] compliant packing, i.e., each message is assigned a base cycle b_m and offset x_m in a slot as described in Section III. The base cycle and offset have to be chosen such that two messages never overlap. In this work, we use

the transformation to bin packing. Thus, each message is assigned a width w_m (the payload) and height

$$h_m = H/r_m \tag{10}$$

with W being the slot size in bytes and $H = 64$, i.e., the number of communication cycles. Instead of determining the base cycle, the goal is now to determine the y-offset y_m which is determined by

$$y_m = l_m \cdot h_m = t(b_m, r_m) \cdot h_m \tag{11}$$

with

$$t(x,y) = \begin{cases} 0, & x = 0 \\ t(\frac{x}{2}, \frac{y}{2}), & x \text{ is even} \\ t(\frac{x-1}{2}, \frac{y}{2}) + \frac{y}{2}, & x \text{ is odd} \end{cases} \tag{12}$$

Here, l_m denotes the level of a message m. Correspondingly, the base cycle b_m is determined as follows:

$$b_m = t(l_m, r_m) = t(\frac{y_m}{h_m}, r_m) \tag{13}$$

The offset x_m in the slot equals the x-offset in the bin. A detailed introduction on this transformation is given in [14].

For the sake of completeness, an ILP formulation that solves the presented scheduling problem optimally is presented in the following. However, experimental results reveal that this exact approach is only applicable to small problems due to the exponential runtime complexity. Hence, a heuristic approach becomes necessary for realistic problems where the ILP becomes intractable. A three-step heuristic is presented that is capable of delivering competitive schedules as shown in the experimental results.

A. Exact Solution with Integer Linear Programming

The ILP formulation relies on the following binary variables:

- $\mathbf{m_{\sigma,l}}$: message m is placed at the level l in slot with slot-id i
- $\mathbf{i}$: slot with slot-id i is used (i_{max} is defined as the maximal number of slot-ids)
- $\mathbf{r_B}$: node $r \in R$ is on branch $B \in \mathcal{B}$
- $\mathbf{r_{B,i}}$: node $r \in R$ is using the slot with id i on branch $B \in \mathcal{B}$

The ILP is formulated as follows:

$$min \sum_{i \in \{1,..,i_{max}\}} \mathbf{i} \tag{14}$$

$$\forall\ m \in M : \sum_{i \in \{1,..,i_{max}\}} \sum_{l=0}^{\frac{H}{h_m}-1} \mathbf{m_{i,l}} = 1 \tag{15}$$

$$\forall\ r \in R, i \in \{1,..,i_{max}\}, y \in \{0,..,H-1\} :$$
$$\sum_{m \in M, src(m)=r} w_m \cdot \mathbf{m}_{\mathbf{i},\lfloor \frac{\mathbf{y}}{\mathbf{h_m}} \rfloor} \leq W \tag{16}$$

$$\forall\ r \in R : \sum_{B \in \mathcal{B}} \mathbf{r_B} = 1 \tag{17}$$

$$\forall\ i \in \{1,..,i_{max}\}, B \in \mathcal{B} : \sum_{r \in R} \mathbf{r_{B,i}} \leq 1 \tag{18}$$

$$\forall_{\substack{m \in M, i \in \{1,..,i_{max}\}, l \in \{0,..,\frac{H}{h_m}-1\}, \\ B \in \mathcal{B}, r \in src(m) \cup dest(m), \tilde{r} = src(m)}} :$$

$$-\mathbf{m_{i,l}} - \mathbf{r_B} + \tilde{\mathbf{r}}_{\mathbf{B,i}} \geq -1 \tag{19}$$

$$\forall\ r \in R, i \in \{1,..,i_{max}\}, B \in \mathcal{B} : -\mathbf{r_{B,i}} + \mathbf{i} \geq 0 \tag{20}$$

The objective function (14) of the ILP minimizes the number of used slot-ids. Here, i_{max} is the minimal number of slot-ids necessary to solve the problem. This number is either the minimal value of the available slots or an upper bound determined by a heuristic for scheduling without the switch. The constraints (15) state that each message m is placed in exactly one slot with slot-id i at the specific level l. By adding the sizes of the messages and restricting this sum by the size of a slot, the constraints (16) ensure that the size of each slot is not exceeded. The constraints (17) state that each node is assigned to exactly one branch as required in Equation (1)-(3). The requirement that a slot-id on a specific branch can at most be assigned to one node as given in Equation (5), is stated in the constraints (18). Correspondingly to the requirement in Equation (7), the constraints (19) ensure that for each message that is assigned to a slot-id, the sender is using the slot-id on the specific affected branch. The constraints (20) state that if a slot-id is used on a specific branch by any resource, the slot-id is allocated for the schedule.

Solving this ILP, provides a slot-id i and level l for each message m. Placing the elements starting from the highest element to the most left void space in the bin at the level l results in a feasible solution of the bin packing problem. Each bin packing solution is transformed to a slot packing $\mathcal{S}$ by determining the base cycle using the level of each message, see Equations (10) and (13). The slot-id assignment σ is deduced from the $\mathbf{m_{i,l}}$ variables. The variables $\mathbf{r_B}$ determine the topology $\mathcal{B}$.

B. Heuristic Solution

Since the ILP approach becomes intractable for problems of even moderate size, we propose a heuristic approach that provides a set of topologies and schedules for a network using the FlexRay switch. For the sake of simplicity and extensibility, the heuristic is separated in three steps: (1) The topology is determined by deducing a set of branches, (2) the messages are packed into slots, (3) a slot-id is assigned to each slot. This multi-step approach has the advantage that some step might be performed manually, e.g., the topology might be already given by replacing an active star device by the proposed FlexRay switch.

1) Topology: The goal of the first step is to determine an appropriate set of branches. Here, nodes which have a high communication flow between each other shall be on same branch to reduce the inter-branch communication. For this purpose, we provide an iterative algorithm that returns a set of candidate topologies in Algorithm 1.

Algorithm 1 Iterative algorithm to determine a set of candidate topologies.

```
1: procedure T(G_R(R,E), ω : E → ℕ)
2:     while |E| > 0 do
3:         select e with ω(e) = min_{ẽ∈E}(ω(ẽ))
4:         E = E\{e}
5:         B = wcc(G_R(R,E))
6:         B_candidates = B_candidates ∪ {B}
7:     end while
8: end procedure
```

The algorithm receives the fully-meshed graph $G_R(R, E)$ for all nodes R and the function $\omega : E \rightarrow \mathbb{N}$ which is determined as the data volume between two nodes as follows:

$$\omega(e = (r, \tilde{r})) = \sum_{\substack{m \in M \\ (r = src(m), \tilde{r} \in dest(m) \vee \tilde{r} = src(m), r \in dest(m))}} w_m \cdot \frac{H}{r_m} \tag{21}$$

The algorithm iteratively removes the edge $e \in E$ with the lowest data volume (line 3,4). The weakly connected components (wcc), i.e., the graphs that are not connected, are determined and the corresponding nodes of each unconnected graph form a branch (line 5). The obtained topology $\mathcal{B}$ is added to the candidate topologies (line 6). Note that this algorithm generates exactly $|R|$ candidate topologies. The worst-case complexity of the algorithm is $O(|R|^4 + |M|)$ since the number of edges is in $O(|R|^2)$, the determination of the weakly connected components requires $O(|R|^2)$, and the function ω is determined in $O(|M|)$.

2) Slot Packing: We use an iterative heuristic algorithm as given in Algorithm 2.

Algorithm 2 Fast greedy heuristic for slot packing.

```
1: S = {}
2: for m ∈ M do
3:     for S ∈ S with ∀m̃ ∈ S : src(m) = src(m̃) do
4:         if place(m, S) then
5:             continue with next m
6:         end if
7:     end for
8:     create new S and add it to S
9:     place(m, S)
10: end for
```

The algorithm starts with an empty set of slots $\mathcal{S}$ (line 1). Each message $m \in M$ is tried to be placed subsequently in a slot $S \in \mathcal{S}$ if the sender of the message equals the sender of the slot (line 2,3). Here, the function $place(m, S)$ tries to place each message m in slot S with the minimal offset and base cycle and returns *true* if the placing is successful, and *false* otherwise (line 1). If a message is not placed in any of the slots in $\mathcal{S}$, a new slot S is allocated, added to $\mathcal{S}$, and the message m is placed into this new empty slot (line 8,9).

The order of the messages in M is determined lexicographically by three attributes that are determined as follows:

- messages that affect a higher number of nodes determined by

$$\sum_{B \in \mathcal{B}, (src(m) \cup dest(m)) \cap B \neq \{\}} |B| \tag{22}$$

  are ordered to the front
- messages with a lower repetition are ordered to the front
- messages with a higher payload are ordered to the front

This ensures that slots are filled first with messages that affect a high number of nodes. Hence, the subsequent packed slots affect a less number of nodes such that a parallel scheduling of these slots is more likely and, thus, bandwidth might be saved. The complexity of this algorithm is in $O(|M| \cdot |\mathcal{S}|)$

3) Slot Assignment: Finally, each slot requires the assignment of a slot-id. In case the number of slot-ids is minimized, this problem equals the vertex coloring problem for the graph $G_{\mathcal{S}}(\mathcal{S}, E_{\mathcal{S}})$ with $(S, \tilde{S}) \in E_S \leftrightarrow b(S) \cap b(\tilde{S}) = \{\}$. Each pair of slots that share at least one branch cannot have the same color or slot-id, respectively. For the vertex coloring, we use a heuristic approach in [17]. The used contraction-based Recursive-Largest-First (RLF) algorithm has a worst-case complexity of $O(|\mathcal{S}|^3)$.

test case	slots	runtime[s]	$\lvert R\rvert$	$\lvert M\rvert$
tc1	28	8.2	8	217
tc2	58	130	16	464
tc3	114	362	32	923

TABLE I
THE THREE TEST CASES PROVIDED FOR A SCALABILITY ANALYSIS. GIVEN IS THE OPTIMAL SOLUTION IN CASE A SINGLE FLEXRAY BUS IS USED AS WELL AS THE REQUIRED RUNTIME. ADDITIONALLY, THE NUMBER NODES AND MESSAGES ARE GIVEN.

VI. EXPERIMENTAL RESULTS

This section discusses our experimental results based on three test cases and one realistic case study. All experiments were carried out on an Intel Core i5 2.53 GHz with 4 GB RAM. The FlexRay bus was configured such that the cycle duration is 5ms and the duration of one static segment is 4.03ms. The static segment consists of 62 slots with each slot having a payload of 42 bytes (one byte is reserved for update bits).

A. Test cases

In order to illustrate the benefits of the FlexRay switch scheduling, a set of test cases is first presented. The proposed three test cases are sampled from the distribution of message lengths and periods as given in the realistic case study of an existing FlexRay bus. The size of messages ranges from 1byte to 32bytes and the periods range from 5ms to 320ms, see [14]. The ratio of multicast messages is approximately 50%. The number of ECUs and messages are given in Table I. The data volume of the first test case equals a state-of-the-art FlexRay bus in the automotive domain. The second test case embodies the data volume that is common in a modern in-vehicle network. The third test case shall serve as an example of an upcoming electric vehicle network with an increased bandwidth due to x-by-wire and additional safety functions.

For a very small example with six nodes ($|R| = 6$) and 42 messages ($|M| = 42$), the ILP approach is able to find the optimal FlexRay switch schedule within 1142s. However, the optimal solution using 8 slots and 2 branches is also found with the proposed heuristic in 29ms. Moreover, for the proposed test cases, the exact ILP approach failed to deliver results within a reasonable amount of time: The optimization was aborted after 24 hours. Thus, the ILP approach is only applicable for small problems such that a heuristic becomes necessary for problems of realistic size.

The optimal results obtained for a network consisting of a single FlexRay bus without the switch are given in Table I. Here, the presented ILP approach is applied separately for each sender node and, finally, the results are assembled in a global schedule. Note that this approach is significantly more efficient, since the resulting ILP formulations are more compact and may be solved separately. While the first test case allows a FlexRay schedule without a switch, the second test case already results in a utilization of 93% (58/62 slots), prohibiting almost any scope for substantial further extensibility. The third test case cannot be scheduled on a FlexRay bus since the required number of slots clearly exceeds the number of available slots.

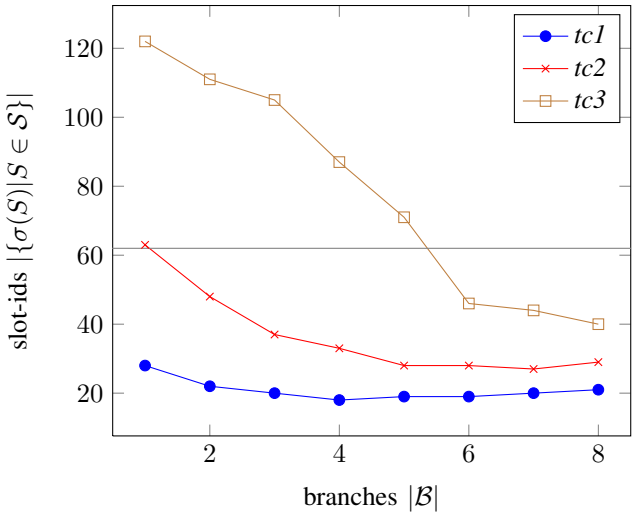

Fig. 5. Results of the test cases for a switched FlexRay scheduling with the proposed three-step heuristic. For the given bus configuration, 62 slots exist in the static segment.

branches	slot-ids	runtime[s]
1 (Powertrain,Chassis,Body,Comfort,FlexRay)	63	1.4
5 (Powertrain)(Chassis)(Body)(Comfort)(FlexRay)	35	0.28
2 (Powertrain,Chassis,FlexRay)(Body,Comfort)	45	0.25

TABLE II
RESULTS OF THE REALISTIC CASE STUDY.

The results obtained with the proposed heuristic are illustrated in Figure 5. For the first test case, the number of slots is reduced to 18 slot-ids saving 10 slots in the case where four branches are used. The second test case shows that the number of used slots may be reduced to 28 using five branches. The most significant gain is achieved for the third test case using eight branches. Here, the number of used slots is reduced to 40 allowing a feasible schedule which cannot be obtained without the presented FlexRay switch. The runtime of the scheduling for a given approach ranges from 24ms to 113ms for *tc1*, 35ms to 175ms for *tc2*, and 77ms to 265ms for *tc3*. Hence, the heuristic is capable of delivering competitive results always within less than 1s. This short runtime enables an iterative optimization of the FlexRay parameters like the communication cycle duration, slot number, or slot size.

B. Case Study

Finally, a realistic case study of a state-of-the-art entire in-vehicle network is used to emphasize the benefits of the proposed FlexRay switch. The used network consists of four CAN buses (*Powertrain,Chassis,Body,Comfort*) and one FlexRay bus interconnected by a central gateway. The number of ECUs is 56 with a total number of 370 messages that are transmitted on the static segment (diagnosis, calibration, maintenance data were not considered since they are not critical and are usually scheduled on the dynamic segment). If all ECUs are interconnected by a single FlexRay bus using a cascaded active star topology, an optimal schedule with 63 slots was obtained, see Table II. This schedule is not feasible since the static segment in the investigated scenario has only 62 slots.

In order to achieve a feasible schedule, the gateway was replaced by the proposed FlexRay switch, i.e., the branches are defined by the given topology such that each CAN bus forms a branch and the FlexRay bus forms a branch. In this case, the proposed heuristic reduced the number of required slots to 35 and hence the resulting schedule is feasible. Our last example used two branches, i.e., the (*Body,Comfort*)-branch and the (*Powertrain,Chassis,FlexRay*)-branch. Here, the obtained number of used slots was 45 which is still sufficient, respecting the upper bound of 62 slots.

VII. CONCLUSION

This paper presents a novel concept for upcoming in-vehicle networks, using a FlexRay switch to increase the safety and effective bandwidth of the network. The prototype FlexRay switch was implemented on an FPGA and tested extensively in terms of compliance with the FlexRay EPL specification. In order to benefit from the FlexRay switch architecture, an approach for determining the topology and scheduling of the network becomes necessary. In this paper, we proposed an exact approach based on ILP and a three-step heuristic approach. Since the exact approach is intractable for problem of realistic size, the heuristic becomes necessary. The heuristic is capable of delivering competitive results as shown for the presented test cases and a realistic case study where it significantly decreases the number of required slots such that a feasible schedule may be obtained.

REFERENCES

[1] CAN, "Controller Area Network," http://www.can.bosch.com/.
[2] *FlexRay Communications System, Protocol Specification Version 2.1 Revision A*, FlexRay Consortium, Dec. 2005. [Online]. Available: http://www.flexray.com
[3] J. Berwanger, M. Peteratzinger, and A. Schedl, "FlexRay startet durch - FlexRay-Bordnetz für Fahrdynamik und Fahrerassistenzsysteme (in German)," in *Elektronik automotive: Sonderausgabe 7er BMW*, 2008, available at http://www.elektroniknet.de/home/automotive/bmw-7/flexray-startet-durch/.
[4] G. Linn, W. Sichert, P. Milbredt, and G. Kistler, "Serieneinfhrung eines weckfhigen FlexRay-Bussystems," *Elektronik Automotive*, vol. Sonerausgabe Audi A8, pp. 102–104, Februar 2010, in German.
[5] AUTOSAR, *Specification of the FlexRay Interface Version 3.0.2*, 2008, http://www.autosar.org.
[6] *FlexRay Communications System, Electrical Physical Layer Specification, Version 2.1*, FlexRay Consortium, May 2005. [Online]. Available: http://www.flexray.com
[7] R. Daoud, H. Amer, H. Elsayed, and Y. Sallez, "Ethernet-based car control network," in *Proc. of CCECE '06*, 2006, pp. 1031–1034.
[8] D. Jansen and H. Buttner, "Real-time Ethernet: the EtherCAT solution," *Computing and Control Engineering*, vol. 15, p. 16, 2004.
[9] MOST, "Media Oriented Systems Transport," http://www.mostcooperation.com/.
[10] S. Ding, N. Murakami, H. Tomiyama, and H. Takada, "A ga-based scheduling method for flexray systems," in *Proc. of EMSOFT '05*. New York, NY, USA: ACM, 2005, pp. 110–113.
[11] K. Schmidt and E. Guran Schmidt, "Message Scheduling for the FlexRay Protocol: The Static Segment," *IEEE Transactions on Vehicular Technology*, vol. 58, no. 5, pp. 2170–2179, 2009.
[12] 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 '09*, 2009, pp. 874–877.
[13] M. Grenier, L. Havet, and N. Navet, "Configuring the communication on flexray: the case of the static segment," in *Proc. of ERTS 2008*, 2008.
[14] M. Lukasiewycz, M. Glaß, P. Milbredt, and J. Teich, "FlexRay Schedule Optimization of the Static Segment," in *Proc. of CODES+ISSS '09*, 2009, pp. 363–372.
[15] P. Milbredt, A. Steininger, and M. Horauer, "Automated Testing of FlexRay Clusters for System Inconsistencies in Automotive Networks," in *Proc. of DELTA '08*, Hong Kong, China, 2008.
[16] P. Milbredt, B. Vermeulen, G. Tabanoglu, and M. Lukasiewycz, "Switched FlexRay Increasing the Effective Bandwidth and Safety of FlexRay Networks," in *Proc. of EFTA '10*, Bilbao, Spain, 2010.
[17] A. Hertz, "A Fast Algorithm for Coloring Meyniel Graphs," *Journal of Combinatorial Theory, Series B*, vol. 50, no. 2, pp. 231–240, 1990.