Optimal scheduling of switched FlexRay networks

Thijs Schenkelaars
Electronic Systems group
Eindhoven University of Technology
The Netherlands

Bart Vermeulen
Central Research and Development
NXP Semiconductors
The Netherlands
bart.vermeulen@nxp.com

Kees Goossens
Electronic Systems group
Eindhoven University of Technology
The Netherlands

***Abstract*—This paper introduces the concept of switched FlexRay networks and proposes two algorithms to schedule data communication for this new type of network. Switched FlexRay networks use an intelligent star coupler, called a switch, to temporarily decouple network branches, thereby increasing the effective network bandwidth. Although scheduling for basic FlexRay networks is not new, prior work in this domain does not utilize the branch parallelism that is available when a FlexRay switch is used. In addition to the novel exploitation of branch parallelism, the scheduling algorithms proposed in this paper also support all slot multiplexing options as defined in the FlexRay v3.0 protocol specification. This includes support for the newly added repetition rates and support for multiplexing frames from different sending nodes in the same slot. Our first algorithm quickly produces a schedule given the communication requirements, network topology and FlexRay parameters, but cannot guarantee an optimal schedule in terms of the bandwidth efficiency and extensibility. Therefore, a second, branch-and-price algorithm is introduced that does find optimal schedules.**

I. Introduction

The proliferation of distributed control applications in cars drives the search for faster and more reliable in-vehicle networks. The increasing demand of bandwidth has led to the development and standardization of the FlexRay protocol [1]. FlexRay supports bandwidths up to 10 Mbps and is introduced as the new de facto standard for in-vehicle applications requiring reliable communication, such as adaptive cruise control, drive-by-wire and electronic stability control.

The FlexRay protocol organizes time into communication cycles, as indicated in Fig. 1. Every cycle consists of four parts.

1) The *static segment* is used to send critical, real-time data, and is divided into *static slots*, in which the electronic control units (ECUs) can send a *frame* on the bus. These frames consist of a header, payload and trailer and are assigned to the slots according to a static, TDMA-based schedule. Channel idle time is enforced between frames to prevent overlapping consecutive frames.
2) The *dynamic segment* enables event-triggered communication. The lengths of the mini slots in the dynamic segment depend on whether or not an ECU sends data.
3) The *symbol window* is used to transmit special symbols, for example to start up the FlexRay cluster.
4) The *network idle time* interval is used by the nodes to allow them to correct their local time bases in order to stay synchronized to each other.

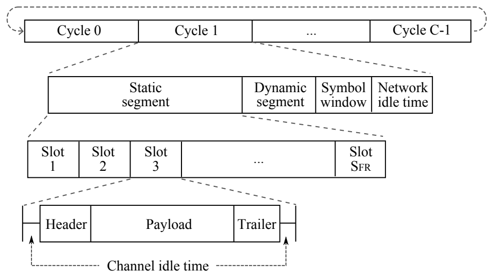


Fig. 1. The FlexRay cycle structure

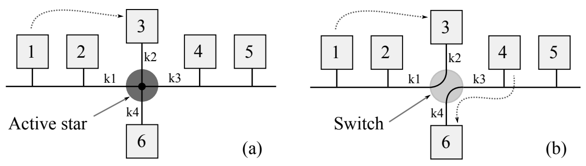


Fig. 2. (a) Active star topology, (b) Switched FlexRay network

FlexRay can be used with multiple network topologies. The most commonly used topology is the hybrid network, in which multiple *branches* are connected together, optionally using a central active star device (see Fig. 2(a)). Each branch can contain one or multiple ECUs, referred to as *nodes*.

When a node has to send a data frame every cycle, it is assigned a static slot in the FlexRay communication schedule. However, it is also possible to assign multiple nodes to a slot, when their frames do not have to be sent every cycle, but can be interleaved in time. In one cycle, the frame of the first node can be sent, and in the next cycle, the frame of the other node. This is called *slot multiplexing*.

The active star forwards the data received from one branch to all other branches. In Fig. 2(a) for example, if Node 1 sends data to Node 3, all other nodes also receive this data, even though they have no need for this data.

By replacing the active star with a FlexRay switch [2], which forwards frames only to those branches that connect to nodes that actually need the information, the network can potentially be simultaneously used by multiple senders, resulting in more net bandwidth. This is called *branch parallelism*. A FlexRay switch uses the channel idle time between frames to internally reconfigure the connections between the branches. Therefore every slot can, in principle, have a different switch configuration. In the switched network in Fig. 2(b), Nodes 1 and 3 can communicate with each other in the same slot in which Nodes 4 and 6 communicate with each other, if the switch keeps the two sets of branches separated for the duration of that slot.

The duration and total number of static slots in a cycle is defined by the car manufacturer. However, by minimizing the number of slots that are actually used to send frames, the schedule can be more easily extended in the future, when the amount of in-vehicle communication increases. An unoptimized schedule has the remaining bandwidth scattered over all slots. When additional frames are added, this scattered bandwidth is more difficult to reuse than empty slots. If not enough continuous bandwidth can be found, the schedule even needs to be completely redesigned.

The problem addressed in this paper is how exactly to schedule the communication in a switched FlexRay network, utilizing both slot multiplexing and branch parallelism, to minimize the number of slots used. For this, we focus on the FlexRay static segment, because currently most networks only make use of this segment, and because the variable lengths of the slots in the dynamic segment make the utilization of branch parallelism in this segment very difficult [2].

The remainder of this paper is organized as follows. In Section II, the scheduling problem is defined, and the notation is introduced. Section III gives an overview of other FlexRay scheduling algorithms and states the differences with the work presented here. Then, in Section IV a first algorithm is introduced that creates reasonably good schedules very quickly for hybrid, switched FlexRay networks. These schedules however do not always use the minimum number of slots. In order to create optimal schedules, Section V describes a second, branch-and-price algorithm that creates schedules with the guaranteed minimum number of slots at the expense of a larger run time. The different algorithms are evaluated and compared to each other in Section VI. Section VII concludes this paper.

II. Problem definition

The problem addressed in this paper, is to create a FlexRay schedule by packing a set of frames in as few slots as possible. A requirement however is that there are no collisions between the frames. The input to our scheduling algorithms is a *use case*, which consists of the communication requirements, the network topology and the parameters of the FlexRay network.

The *communication requirements* define a set of frames $\mathcal{F}$. For every frame $f \in \mathcal{F}$, a repetition rate r_f is given, that states every how-many cycles the frame needs to be sent. The sender of a frame is denoted by $n_f \in \mathcal{N}$, where $\mathcal{N}$ is the set of all nodes. The set $\mathbf{d}_f$ denotes the set of receivers.

The second input for the scheduling algorithms is the *network topology*. A topology is defined by a set of nodes $\mathcal{N}$, a set of branches $\mathcal{K}$ and a mapping function $branch\colon \mathcal{N} \rightarrow \mathcal{K}$, which for every node n returns the branch k on which it is located.

The third input of the scheduling algorithms is the set of *FlexRay parameters*. The important scheduling parameters are the number of cycles C and the total number of slots in the static segment S_{FR} (see Fig. 1).

Given a use case, the scheduler assigns each frame $f \in \mathcal{F}$ to a slot s_f. As stated, multiple frames may be placed in the same slot by using slot multiplexing and branch parallelism, in order to minimize the number of used slots. Two frames can be multiplexed in the same slot, if the least common multiple of their repetition rates r_f is larger than 1. By assigning every frame a different base cycle b_f, which is the first cycle in which a frame is sent, the frames will be interleaved over the cycles.

A frame f with base cycle b_f is sent in the cycles

$$c = (b_f + n \cdot r_f). \qquad \forall n \in \mathbb{N}_0 | b_f + n \cdot r_f < C \tag{1}$$

In this paper, we consider only integer repetition rates that exactly divide the number of cycles C. To be periodic, the base cycle of a frame must be strictly smaller than its repetition rate:

$$0 \leq b_f < r_f. \tag{2}$$

If the sender and receivers of two frames are located on disjoint sets of branches, branch parallelism is used by isolating these two sets of branches, as shown in the example of Fig. 2(b).

III. Related work

In this section, the work presented in this paper is compared with several previously published papers on FlexRay scheduling. All previous scheduling algorithms, as well as the algorithms presented in this paper, have the common objective to minimize the number of used slots, and thereby to maximize the extensibility of the schedule.

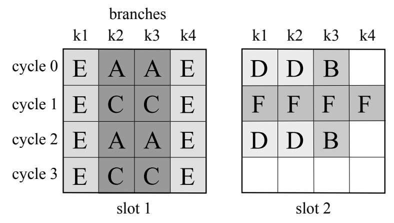
Packing frames into slots is a two dimensional bin-packing problem. Because bin-packing problems are known to be NP-hard, no optimal polynomial-time algorithm for the scheduling problem is known.

The most fundamental difference between this work and previous work on FlexRay scheduling, is that, to the best knowledge of the authors, this paper is the first to address the scheduling of switched FlexRay networks. A second advancement is that the algorithms presented here, fully support the FlexRay v3.0 specification of slot multiplexing, while all prior work is based on older specifications. In these earlier FlexRay specifications, all repetition rates had to be a power of two, and only frames coming from the same sender were allowed to be multiplexed in a single slot. With the introduction of FlexRay v3.0, additional repetition rates are added and frames from different senders are allowed to share a single slot, creating many more possibilities to schedule the communication.

An important difference however, is that instead of frames, the previously published papers regard the *signals* or protocol data units (PDUs) as the basic communication units. A signal is a small quantity of data, such as a sensor reading or a control signal that is sent over the network. These signals are grouped together in PDUs. Depending on their sizes, multiple PDUs fit in one frame. The process of putting multiple PDUs into frames is called *frame packing*. After slot multiplexing

TABLE I
EXAMPLE COMMUNICATION REQUIREMENTS

frame f	rep. rate r_f	sender n_f	receivers $\mathbf{d}_f$	branches $\mathbf{k}_f$	weight w_f
A	2	3	{4}	$\{k2, k3\}$	$\frac{1}{4}$
B	2	4	{5}	$\{k3\}$	$\frac{1}{8}$
C	2	5	{3}	$\{k2, k3\}$	$\frac{1}{4}$
D	2	2	{3}	$\{k1, k2\}$	$\frac{1}{4}$
E	1	6	{1,2}	$\{k1, k4\}$	$\frac{1}{2}$
F	4	1	{3,5,6}	$\{k1, k2, k3, k4\}$	$\frac{1}{4}$



f	s_f	b_f
A	1	0
B	2	0
C	1	1
D	2	0
E	1	0
F	2	1

Fig. 3. Schedule created by decreasing first-fit algorithm

and branch parallelism, frame packing is the third dimension along which multiple frames (or PDUs in this case) can be placed in a single slot. It should be possible to extend the algorithms described in this paper to also take advantage of this third dimension. However, this is still future work and therefore not covered in this paper.

In [3], a best-slot-first heuristic is introduced that packs signals into frames and frames into slots. In addition to scheduling, the genetic algorithm in [4] maps control tasks to the available ECUs, given a task graph with repetition rates.

The scheduler in [5] distinguishes two phases. First, the PDUs are packed into frames, thereby minimizing the number of frames per node. Then, the resulting frames are assigned to slots. The algorithm in [6] combines these two phases into a single optimization step. An integer linear program is solved to create an initial schedule. Then simulated annealing is used to improve the extensibility of the individual slots.

In [7], the concept of a FlexRay switch is also introduced. Although a generic system design approach is outlined, no actual scheduling algorithm is described. A proof-of-concept implementation of a FlexRay switch is described in [2].

IV. DECREASING FIRST-FIT ALGORITHM

Before presenting the first, polynomial-time decreasing first-fit (DFF) scheduler, an example use case is given. Table I shows a set of communication requirements. The table contains two additional columns that can be derived from the other data. The column $\mathbf{k}_f$ states for every frame on which branches it must be sent. Taking into account the topology of Fig. 2(b), $\mathbf{k}_f$ is the set of branches on which the sender or one of the receivers of a frame is located. The second extra column gives every frame a weight w_f, which is defined by:

$$w_f = \frac{1}{r_f} \cdot \frac{|\mathbf{k}_f|}{|\mathcal{K}|}, \tag{3}$$

and states how much of the available bandwidth per slot is required by the frame. The number of cycles C must be a common multiple of all repetition rates. In the example, the smallest common period is 4.

Listing 1. Decreasing first-fit algorithm

```
Sort frames in $\mathcal{F}$ in decreasing order of weight $w_f$.
**for each** frame $f \in \mathcal{F}$
    **for each** slot $s \in \mathcal{S}$
        **for each** base cycle $b \in \{0, \ldots, r_f - 1\}$
            **if** no overlap with earlier placed frames:
                place frame $f$ in slot $s$ with base cycle $b$
                continue with next frame $f$
            **end if**
        **end for**
    **end for**
    frame $f$ could not be scheduled, so DFF failed
**end for**
```

The DFF scheduling algorithm is stated in Listing 1. Before scheduling, the frames are sorted according to their weight w_f in decreasing order. $\mathcal{S}$ is the set of (initially empty) slots. The scheduler tries to place every frame f in the first slot $s \in \mathcal{S}$ in which the frame fits. To check whether a frame can be placed in a slot, every base cycle $b \in \{0, \ldots, r_f - 1\}$ is checked whether there would be a collision between any of the previously scheduled frames and the current frame. Whenever a frame cannot be placed in any of the slots, the DFF scheduler fails to find a feasible schedule.

The schedule in Fig. 3 is created by the DFF algorithm. Besides the graphical representation, it contains a table with the slots and base cycles of the frames. The two large squares in Fig. 3 represent two slots. Each slot is built-up from smaller squares called *cells*, each representing one cycle on a single branch of the given slot. Every frame needs one or more cells in a specific configuration, in order to meet its communication requirement. Every cell can be used by at most one frame.

On the horizontal axis of the slots in Fig. 3, branch parallelism is shown. Different frames can be placed next to each other, as long as their sets of branches $\mathbf{k}_f$ are disjoint. In the example, frames A and E can be placed in the same slot, because the switch can separate their branches. Slot multiplexing is shown on the vertical axis of the slots. Having a repetition rate of 2, frames A and C are placed in the same slot on base cycles 0 and 1, respectively. Although they share the same branches, there is no collision because the frames are sent in alternating cycles.

V. BRANCH-AND-PRICE ALGORITHM

The schedules obtained by the DFF algorithm however do not always have the guaranteed minimum number of used slots. Optimal schedules can be obtained by formulating the problem as an integer linear program (ILP), referred to as the *full ILP* formulation. However, for realistic use cases as described in Section VI, this full ILP has more than 20,000 binary variables and 5,000 constraints. As will be shown, solving this problem takes a long time with a generic ILP solver. However, by applying the Dantzig–Wolfe decomposition [8], a decomposed formulation is obtained that takes advantage of the fact that every slot has similar constraints, allowing the problem can be solved more efficiently. This section describes a branch-and-price (BP) algorithm [9]–[11], which finds optimal schedules by solving this decomposed formulation. The BP algorithm focuses on individual slots. A set of frames that can be scheduled in a single slot is called a *packing*. In the example schedule in Fig. 3, the slots 1 and 2 contain the packings {A,C,E} and {B,D,F}. For the example use case in Table I, there are in total 22 unique packings. A schedule is created by assigning some of these packings to the slots of the static segment. To create a schedule that uses the least possible number of slots, the smallest possible subset of packings needs to be found that contains all frames at least once. The problem of finding this smallest subset is called the *master problem*.

A. Column generation

However, for larger problems, it is infeasible to enumerate all possible packings. Therefore, the BP algorithm starts with only a small set of packings and iteratively adds packings to the set that can potentially reduce the number of slots. In this way, not all packings have to be generated upfront. In the master problem, all packings are represented by columns. For example, the packing {A,B,D} is represented by column $\mathbf{p} = [1\,1\,0\,1\,0\,0]^{\mathrm{T}}$. The principle of iteratively adding packings to the master problem is therefore called column generation.

When a master problem is solved with only a limited number of all possible packings, it is called an restricted master problem (RMP). Whether packing $\mathbf{p}_i$ is selected in the smallest subset is denoted by variable λ_i, where $i \in \{1, \ldots, N\}$ and N is the number of packings in the RMP. The RMP is formulated as the linear optimization problem in Eq. (4).

$$\text{minimize:} \quad S = \sum_{i=1}^{N} \lambda_i \tag{4a}$$

$$\sum_{i=1}^{N} \lambda_i \; \mathbf{p}_i \geq \mathbf{1} \tag{4b}$$

$$0 \leq \lambda_i \leq 1 \qquad \forall i \in \{1, \ldots, N\} \tag{4c}$$

The objective of minimizing the number of selected packings is expressed in Eq. (4a). The left-hand side of Eq. (4b) results in a vector stating how often every frame is selected, which must be at least once for every frame. The bounds of the λ_i variables are stated in Eq. (4c).

This linear formulation is solved with the simplex algorithm [12]. The simplex algorithm hereby iteratively moves from one feasible subset of packings to the other, until an optimal one has been found. At every step, it inspects which packings should be included in the smallest subset. From [9], it follows that only packings for which the constraint

$$\mathbf{y}^{\mathrm{T}}\mathbf{p} > 1 \tag{5}$$

holds are considered. Therefore, only those packings have to be added to the restricted master problem. In Eq. (5), $\mathbf{y}$ is the dual value vector obtained by the simplex algorithm, in which every frame maps to a value. The product $\mathbf{y}^{\mathrm{T}}\mathbf{p}$ represents the sum of these dual values of the frames present in packing $\mathbf{p}$.

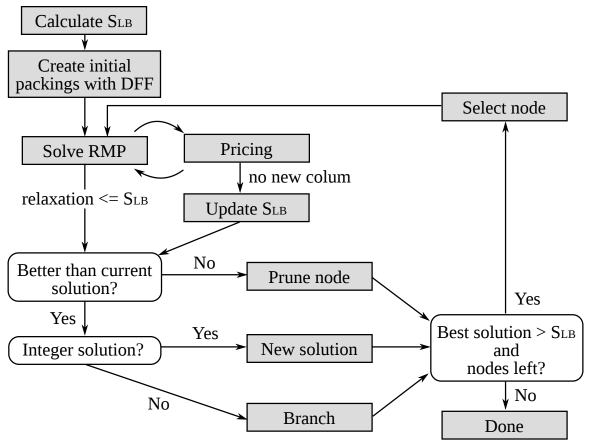


Fig. 4. Branch-and-price implementation

B. Pricing problem

To find a packing for which Eq. (5) holds, a *pricer* tries to find a new, feasible packing $\mathbf{p}$ that maximizes this product. If it is larger than 1, the packing is added to the RMP. However if no packing with a value larger than 1 is found, the current solution of the restricted master problem forms an optimal schedule.

The first-fit pricer tries to quickly find a packing with a value $\mathbf{y}^{\mathrm{T}}\mathbf{p}$ larger than 1. Similar to the DFF scheduler, it tries to iteratively place the frames on all possible base cycles. The frames are sorted in decreasing order of y_f/w_f, so that the frames with the highest 'value per bandwidth' are tried first. If the first run does not give a packing with a value larger than 1, the process is repeated several times, but every time with a small variation in the order of the frames.

It is however not guaranteed that the first-fit pricer always finds a packing with a value larger than 1 when there is one. In order for the BP algorithm to be optimal, it must be ensured that any potential packing that improves the existing solution will always be found. By formulating the pricing problem as an ILP, a packing with a value $\mathbf{y}^{\mathrm{T}}\mathbf{p}$ larger than 1 will always be found, if one exists. However, because this ILP pricer is much slower than the first-fit pricer, the ILP pricer is only used when the first-fit pricer cannot find a packing first.

C. Lower bound

Before the BP algorithm starts, a lower bound on the obtainable number of slots is calculated. As soon as a schedule with this number of slots is found, the column generation process is terminated. The lower bound on the required number of slots is obtained by calculating per branch how many slots are needed to schedule the communication on that specific branch. When taking the maximum number of slots over all branches, the following bound is obtained:

$$S_{\mathrm{LB}} = \max_{k \in \mathcal{K}} \left\lceil \sum_{f \in \mathcal{F} : k \in \mathbf{k}_f} \frac{1}{r_f} \right\rceil \text{ slots.} \tag{6}$$

Here, the set $\{f \in \mathcal{F} : k \in \mathbf{k}_f\}$ denotes the set of frames that have a sender or receiver on branch k.

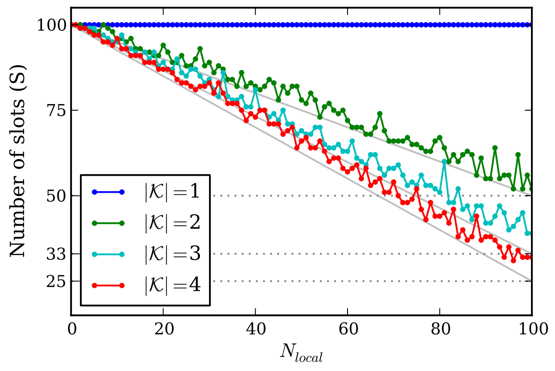


Fig. 5. Impact of branch parallelism

D. *Branch-and-bound*

Although the solution of an RMP for which no new columns can be found is optimal in the sense that it minimizes the number of slots, it is not guaranteed that all λ_i are 0 or 1 (refer to Eq. 4c). The branch-and-bound algorithm described in [9] is used to make all variables integer.

The implementation of the branch-and-price algorithm, which is a combination of column generation and branch-and-bound, is shown in Fig. 4. Whenever an optimal solution of a master problem has been found, or if its objective value reaches S_{LB}, there are three options:

1) The node is *pruned* if the number of slots of this solution is higher than the best solution so far.
2) If the linear program (LP) solution is better than the best solution found, and all variables λ_i are 0 or 1, a *new solution* has been found.
3) On the other hand, if the solution is better than the current best solution, but contains fractional values, the node is *branched*.

The branching rule focuses on pairs of frames. Whenever the optimal solution of an RMP contains fractional values, two frames are selected, and on the left child-node, these two frames must appear together in all packings and on the right child-node, the frames are not allowed to be scheduled in the same slot. These branching constraints can easily be incorporated into the pricing problem [11].

After a node has been processed, the algorithm checks whether the lower bound has been obtained. When it has, or when there are no nodes left to be processed, the algorithm is done. Otherwise, the next node is processed.

VI. Results

First, the BP algorithm is validated by inspecting the influence of branch parallelism and slot multiplexing in isolation. After this validation, more realistic use cases are explored, and a comparison is made between the different algorithms described in this paper.

To show the impact of branch parallelism, a set of use cases is created with two types of frames: broadcast frames and local frames. Broadcast frames are sent to all branches in the network and local frames are sent between two nodes on the same branch. The total number of frames per use

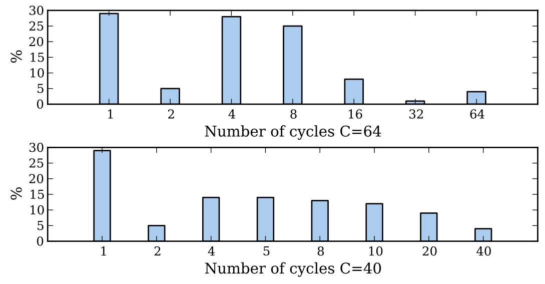


Fig. 6. Distribution of frame repetition rates

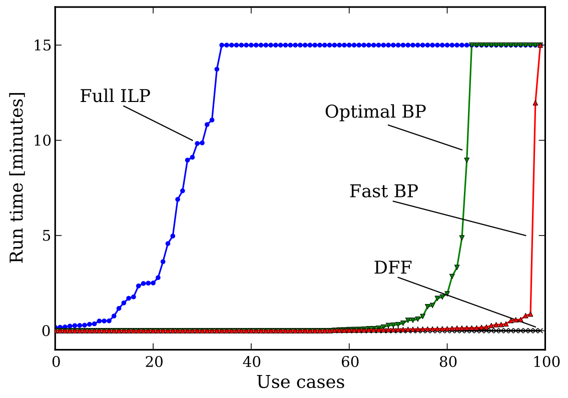


Fig. 7. Run time comparison of different algorithms

case is 100. Fig. 5 shows the required number of slots for topologies with 1 to 4 branches. Every branch contains two nodes. If the number of local frames (N_{local}) is increased, more potential branch parallelism is introduced, resulting in a lower number of required slots. If the local frames are divided evenly over all branches, the minimum number of slots is $N_{\text{broadcast}}+\lceil\frac{N_{\text{local}}}{|\mathcal{K}|}\rceil$. These lower bounds are depicted by the thin gray lines. However, because the local frames in realistic use cases are typically not uniformly distributed, this lower bound is not reached in most cases. For all use cases, the scheduler is however able to find the least possible number of used slots. Similar tests have been performed to show the effect of slot multiplexing in isolation. These tests also resulted in optimal solutions.

To show the behavior of the algorithms for more realistic use cases, another set of 100 use cases with 100 frames each has been randomly generated. The communication requirements are scheduled on a network with 8 nodes on 4 branches. The number of branches on which the frames are sent, is for every frame uniformly distributed from 1 to 4. Half of the use cases have a cycle count C of 64 and the other use cases have 40 cycles. The distributions from which the repetition rates are drawn, are shown in Fig. 6. These distributions are based on the “realistic case study” in [6].

Fig. 7 shows the run times of the different algorithms, when executed on an 3.16 GHz Intel Xeon X5460 processor. The *full ILP* run times are obtained by solving the ILP formulation of the scheduling problem before applying the Dantzig–Wolfe decomposition with the Coin-or branch-and-cut solver [13]. Because the run times can get very large, all runs in this figure are terminated after 15 minutes. Fig. 7 also shows the run

TABLE II
ALGORITHM RESULTS

	DFF	Optimal BP	Fast BP
Avg. nr. of slots	30.22	30.05	29.99
Optimal use cases	71	91	94
Maximum runtime	50ms	10 hours	30 min

times of the DFF algorithm, which always finishes within 50 milliseconds. Even though the DFF algorithm is very fast, it provides reasonable good schedules. For 71 of the 100 use cases, the schedules obtained were optimal, see Table II.

Also the runtimes of the optimal BP scheduler are plotted in Fig. 7. In 80 of the 100 use cases, it finds an optimal schedule within 2 minutes and is therefore much faster than the full ILP algorithm. After 10 hours, the optimal BP scheduler finds optimal schedules for 91 use cases. It reduces 20 schedules obtained with the DFF scheduler by 1 slot, which is a reduction of 3 to 5.5 percent. However, in 9 use cases, the optimal scheduler was still running after 10 hours. This long run-time is caused by the ILP pricer, which sometimes takes many hours to find out whether a packing with a value $\mathbf{y}^{\mathrm{T}}\mathbf{p}$ larger than one exists. The fast BP algorithm is introduced to overcome this problem by disabling the ILP pricer. Whenever the first-fit pricer cannot find a packing with a value larger then one, it is (incorrectly) assumed that there is none. This makes the algorithm much faster, at the cost of not being able to prove the optimality of the schedules.

A more detailed comparison between the fast BP and the optimal BP algorithms is shown in Fig. 8. It shows that the average number of slots of the fast BP algorithm indeed drops much faster than the optimal BP algorithm. To show that most improvements are made in the first minutes, the figure has two different time scales.

All runs of the fast BP algorithm finish within 30 minutes. The fast algorithm then optimized 94 use cases, with an average of 29.99 slots per use case. The optimal BP algorithm on the other hand has an average of 30.05 slots per use case, after 30 minutes. Within limited time, the fast BP algorithm therefore even performs better than the optimal algorithm.

In theory however, with more runtime, the optimal BP algorithm does better then the fast BP algorithm. This can be seen from the dotted lines in Fig. 8, which indicates the lower bounds on the number of slots that the algorithms will eventually achieve. After 30 minutes, the optimal BP algorithm still has room for improvement, whereas the fast BP algorithm finished.

VII. Conclusion

This paper presented an approach to schedule data communication in switched FlexRay networks, with the objective to minimize the number of used slots, and thereby to maximize the extensibility of the resulting schedules. The algorithms use the branch parallelism introduced by the FlexRay switch and slot multiplexing as defined in FlexRay v3.0.

The decreasing first-fit scheduler produces good schedules very quickly. However, in general the resulting schedules are not optimal. The branch-and-price algorithm is introduced to find these optimal schedules. However, for some use cases, the optimal BP algorithm takes very long. This might lead to problems if the computer runs out of memory when the branch-and-bound tree becomes too large. The fast BP algorithm is introduced to overcome this problem. Within limited time, the fast BP pricer produces even better schedules than the optimal algorithm. However, the fast BP algorithm cannot guarantee the optimality of the schedules.

Although the improvements that the BP algorithms bring are not very large, the extra time investment of finding smaller schedules, may be worth the extra computation time, given that for every car model these schedules only have to be calculated once.

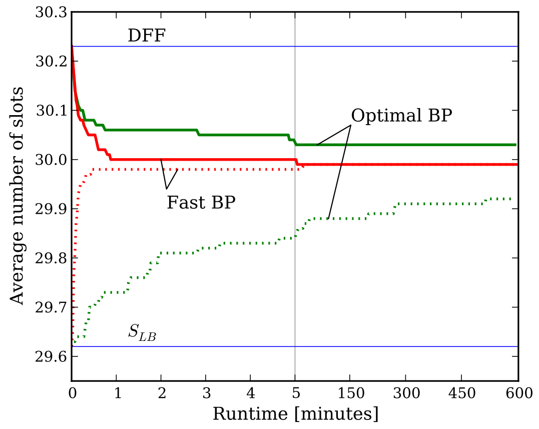


Fig. 8. Convergence of fast and optimal BP algorithms

References

[1] *Protocol specification Version 3.0*, FlexRay Consortium, Dec. 2009. [Online]. Available: http://www.flexray.com

[2] P. Milbredt, B. Vermeulen, G. Tabanoglu, and M. Lukasiewycz, "Switched FlexRay: Increasing the Effective Bandwidth and Safety of FlexRay Networks," *IEEE ETFA*, 2010.

[3] M. Grenier, L. Havet, and N. Navet, "Configuring the communication on flexray - the case of the static segment," in *ERTS*, 2008.

[4] S. Ding, N. Murakami, H. Tomiyama, and H. Takada, "A GA-based scheduling method for FlexRay systems," in *EMSOFT: conference on embedded software*. ACM, 2005, pp. 110–113.

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

[6] M. Lukasiewycz, M. Glaß, J. Teich, and P. Milbredt, "Flexray schedule optimization of the static segment," in *CODES+ISSS*, 2009.

[7] G. Tabanoglu, "Synchronous Switched Scheduling with Heterogeneous Cycle Configurations for Efficient Bandwidth Usage in a FlexRay Cluster," 2010.

[8] G. B. Dantzig and P. Wolfe, "Decomposition principle for linear programs," *Operations Research*, vol. 8, no. 1, pp. 101–111, 1960.

[9] P. H. Vance, C. Barnhart, E. L. Johnson, and G. L. Nemhauser, "Solving binary cutting stock problems by column generation and branch-and-bound," *Comput. Optim. Appl.*, vol. 3, no. 2, pp. 111–130, 1994.

[10] C. Barnhart, E. Johnson, G. Nemhauser, M. Savelsbergh, and P. Vance, "Branch-and-price: Column generation for solving huge integer programs," *Operations Research*, vol. 46, no. 3, pp. 316–329, 1998.

[11] E. Johnson, G. Nemhauser, and M. Savelsbergh, "Progress in linear programming-based algorithms for integer programming: An exposition," *INFORMS Journal on Computing*, vol. 12, no. 1, pp. 2–23, 2000.

[12] F. S. Hillier, G. J. Lieberman, F. Hillier, and G. Lieberman, *Introduction to Operations Research*. McGraw-Hill, July 2004.

[13] "Coin-or branch and cut solver, version 2.4.0," http://www.coin-or.org/, COIN-OR: Common Optimization Interface for Operations Research.