# Schedulability Analysis for the Dynamic Segment of FlexRay: A Generalization to Slot Multiplexing

Bogdan Tanasa Unmesh D. Bordoloi Stefanie Kosuch Petru Eles Zebo Peng
Department of Computer Science, Linköpings Universitet, Sweden
E-mail: {bogdan.tanasa, unmesh.bordoloi, stefanie.kosuch, petru.eles, zebo.peng}@liu.se

***Abstract*—FlexRay, developed by a consortium of over hundred automotive companies, is a real-time communication protocol for automotive networks. In this paper, we propose a new approach for timing analysis of the event-triggered component of FlexRay, known as the dynamic segment. Our technique accounts for the fact that the FlexRay standard allows slot multiplexing, i.e., the same priority can be assigned to more than one message. Existing techniques have either ignored slot multiplexing in their analysis or made simplifying assumptions that severely limit achieving high bandwidth utilization. Moreover, we show that our technique returns less pessimistic results compared to previously known techniques even in the case where slot multiplexing is ignored.**

## I. Introduction

FlexRay is a hybrid communication protocol for automotive networks, i.e., it allows the sharing of the bus between both time-triggered and event-triggered messages. The time-triggered component is the static (ST) segment and the event-triggered component is known as the dynamic (DYN) segment. The FlexRay bus protocol has garnered widespread support as a vehicular communication network. Its popularity has been driven by the fact that it was developed by a wide consortium [3] of automotive companies. In fact, cars equipped with FlexRay are already in the streets or in production [4]. As the cost associated with FlexRay deployment is expected to go down over the next few years, more and more x-by-wire applications are expected to communicate over FlexRay.

### A. Our contributions and related work

Timing analysis of the DYN segment is NP-hard in the strong sense [11] and hence, efficient heuristics [11], [16] have been constructed towards providing upper bounds on the worst-case delays suffered by the messages on DYN segment. However, these methods were limited to the case where slot multiplexed is not permitted on the DYN segment. Zeng et al. have proposed a heuristic [16] that outperforms the heuristic proposed by Pop et al. [11] with respect to the degree of pessimism. Our experiments show that, in practice, the results obtained by our scheme are significantly better than the method by Zeng et al. [16]. Other techniques have limitations like relying on ILP-solvers. They also lack rigorous experimental results because they study only one cases study.

Even more importantly, our analysis is general than existing methods in the sense that it can analyze worst-case delays of messages for those FlexRay configurations where slot multiplexing is allowed, i.e., the same priority can be assigned to multiple messages. While timing analysis of the FlexRay dynamic segment has generated significant research interest in recent years, almost all the known approaches [6], [16], [11], [12] have ignored slot multiplexing. This is inspite of the fact that the FlexRay specification [3] allows slot multiplexing. Schneider et. al. [13], [14] did propose the use of slot multiplexing in the context of FlexRay DYN segment. However, their approach severely restricts the scope of multiplexing by assigning infinite delay to any message that will be displaced over more than one FlexRay cycle. As we illustrate in Section III, such a method [13], [14] is very pessimistic and returns negative results even for simple setups where the bandwidth demand from messages is quite small. In contrast, our technique is quite general and can estimate message delays that span over multiple cycles.

Finally, our result is significant because it paves the way to design approaches that optimizes the delays by configuring parameters that were not in play before. A small example to illustrate this will be discussed in Section X.

### B. Overview of the proposed scheme

In the case where slot multiplexing is ignored, Pop et al. [11] have shown that the core problem of computing the worst-case delays of messages transmitted on the DYN segment can be transformed into the bin covering problem [10]. The objective of the bin covering problem is to maximize the number of bins that can be filled to a minimum capacity with a set of items whose weights have been specified. Our algorithm, described in Section VI is directly inspired by recent theoretical advances in approximating the upper bounds on the optimal solution for the bin covering problem that were reported by Jansen and Solis-Oba [7].

However, for the case of slot multiplexing, the problem can not be transformed in to the traditional bin covering problem. Rather, the problem becomes what we call as the *bin covering problem with conflicts* in this paper. The approach proposed by Jansen and Solis-Oba was meant for the traditional bin packing problem. Hence, we need to suitably adapt their technique to the problem of bin covering with conflicts when slot multiplexing is allowed in the DYN segment of FlexRay. This will be discussed in detail in Section VII.

## II. The FlexRay Dynamic segment

The FlexRay Communication protocol [3] is organized as a periodic sequence of communication cycles with fixed length, $l_{FC}$. Each communication cycle is further subdivided into

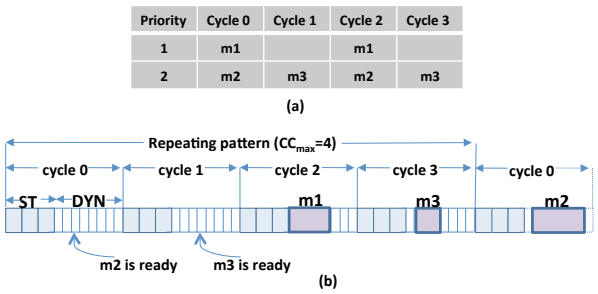


Fig. 1: Example 1: Messages $m_1$ and $m_2$ are multiplexed in FlexRay DYN segment.

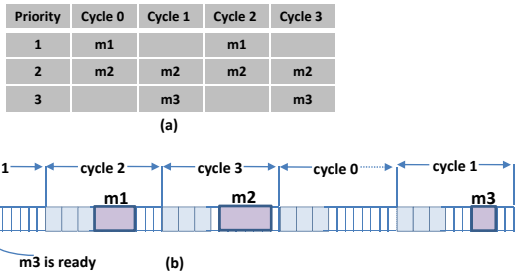


Fig. 2: Example 2: Messages $m_1$ and $m_3$ are never transmitted in same cycles, yet $m_1$ influences the transmission of $m_3$ indirectly.

two a ST and a DYN segment. In this paper, we propose a timing analysis scheme for the DYN segment and hence, in the following we discuss the DYN segment.

**Dynamic Segment:** In FlexRay a set of $CC_{max}$ communication cycles constitute a pattern which is repeated. Each cycle is indexed by a *cycle counter*. The cycle counter is incremented from 0 to $CC_{max} - 1$ after which the cycle counter is reset to 0. Figure 1(a) illustrates a FlexRay communication pattern with $CC_{max} = 4$. Each message to be transmitted over the FlexRay DYN segment is assigned two attributes that define the set of cycles between 0 and $CC_{max} - 1$ where the message is allowed to be transmitted. These attributes for a message $m_i$ are (i) the base cycle or the starting cycle $B_i$ within $CC_{max}$ communication cycles, and (ii) the cycle repetition rate $R_i$ which indicates the minimum length (in terms of the number of FlexRay cycles) between two consecutive allowable transmissions. As an example, let us consider three messages $m_1$, $m_2$ and $m_3$ to be transmitted over the FlexRay cycles in Figure 1(a). Let the base cycles be $B_1 = B_2 = 0$ and $B_3 = 1$ and let the repetition rates be set to $R_1 = R_2 = R_3 = 2$. Figure 1(a) shows the cycles where $m_1$, $m_2$ and $m_3$ can be transmitted with these properties. In this example, $m_2$ and $m_3$ can be transmitted in cycle 0 and cycle 1 respectively. Thereafter, they can be transmitted every alternate cycle. The same priority is assigned to $m_2$ and $m_3$ and they are said to be slot multiplexed. According to the FlexRay standard, the base cycle $B_i \in [0...CC_{max} - 1]$, and $B_i < R_i$. The relation $B_i \in [0...CC_{max} - 1]$ holds true by definition. The relation $B_i < R_i$ is also enforced by the specification to ensure the definition of $R_i$ when it straddles two adjacent FlexRay cycles.

Conflicts between messages to be sent in the same cycle are resolved using priorities as each message is assigned a fixed priority. Each DYN segment in FlexRay is partitioned into equal-length slots which are referred to as "minislots". At the beginning of each DYN segment, the highest priority message gets access to the bus and it occupies the required number of minislots on the bus according to its size. However, if the message is not ready for transmission or the size of the message does not fit into the remaining portion of the DYN segment, then only one minislot goes empty. In either case, the bus is then given to the next highest-priority message and the same process is repeated until the end of the DYN segment. Further, at most one instance of each message is only allowed to be transmitted in each FlexRay cycle. In Figure 1(b), the DYN segment in each FlexRay cycle consists of 8 minislots. $m_1$ is the highest priority message (priority 1) in cycle 2 and hence, occupies 5 minislots corresponding to its size. In cycle 3, however, there is no message with priority 1 that is ready and hence, one minislot is wasted before $m_3$ with priority 2 is transmitted.

## III. Motivational Examples

With the help of two examples, we shall illustrate the need for new techniques, as proposed in this paper, for timing analysis of the DYN segment when slot multiplexing is allowed. First let us consider the example shown in Figure 1(a). The size of the messages in terms of the minislots is 5MS (minislots) for $m_1$, 6 MS for $m_2$ and 3 MS for $m_3$. Let us consider the worst-case delay scenario for message $m_2$. Consider that $m_2$ is ready just after minislot 2 in cycle 0 and hence, it cannot be transmitted in cycle 0. With $B_2 = 0$ and $R_2 = 2$, $m_2$ cannot be transmitted in cycle 1. Let us assume that $m_1$ is ready now and is transmitted in cycle 2. Thereafter, $m_1$ occupies 5 MS the DYN segment has 3 MS left. Thus, $m_2$ cannot fit into cycle 2. Again, $m_2$ is not allowed to be transmitted in cycle 3. Finally, $m_2$ is transmitted in cycle 0 in the next round. Thus, $m_2$ is delayed more than 4 FlexRay cycles but less than 5 FlexRay cycles. On the other hand, in the worst-case scenario, $m_3$ just misses its minislot in cycle 1 and is transmitted in cycle 3. Thus, $m_3$ is delayed less than 2 FlexRay cycles.

From the above example, first, we note that both $m_2$ and $m_3$ have the same priority and yet they have completely different worst-case delays. Secondly, we emphasize that the approach proposed by Schneider et. al. [13], [14] would report that $m_2$ and $m_3$ would suffer from infinite delays because they are delayed more than one FlexRay cycle. Thus, their approach would report that the given message set is unschedulable. However, as we have seen in this example, $m_2$ and $m_3$ have finite delays and are actually schedulable for any values of deadlines that span more than the length of 5 and 2 FlexRay cycles.

Let us now consider a second example as shown in Figure 2(a). Compared to Figure 1(a), $m_3$'s priority is now 3 and $m_2$ now has $R_2 = 1$. Thus, $m_2$ and $m_3$ now have conflicts in cycles 1 and 3. Rest of the values remain the

same as in Figure 1(a). Note that $m_1$ and $m_3$ have no cycles in common. However, $m_1$ might delay $m_2$ and that might lead to delaying of $m_3$. Hence, techniques that have not considered slot multiplexing [6], [16], [11], [12], will not be able to report accurate results. In order to provide safe results, such techniques would have to assume that the message $m_1$ is allowed to be transmitted in every cycle and this would lead to very pessimistic results.

## IV. System Model

We assume that the set of messages $\mathbf{\Gamma}$ that will be transmitted on the FlexRay DYN segment is known. Any message $m_i \in \mathbf{\Gamma}$, is associated with the following properties.

1) The period $T_i$ that denotes the rate at which $m_i$ is being produced.
2) The deadline $D_i$, of a message $m_i$ is the relative time since the production of $M_i$ until the time by which the transmission of $m_i$ must end.
3) The repetition rate $R_i$, and the base cycle $B_i$ for each message $m_i$, as defined in Section II, is assumed to be known.
4) The size of the message $W_i$ in terms of the number of minislots that the message $m_i$ would occupy when transmitted on the DYN segment.
5) The priority $ID_i$ of each message $m_i$ that is used to resolve bus access contentions as discussed in Section II, is assumed to be known. A higher value of the priority implies a lower priority.

We assume that the FlexRay cycle length is $l_{FC}$. The length of one minislot is denoted $l_{MS}$, and the total number of minislots $N_{MS}$ is considered to be given. The length of the DYN segment is thus $l_{DYN} = l_{MS} \times N_{MS}$. Assuming that the length of the ST is $l_{ST}$, FlexRay cycle length is $l_{FC} = l_{ST} + l_{DYN}$.

## V. Preliminaries

Our schedulability analysis technique relies on computing the worst-case response time of each message. If the worst-case response time of any message is greater than its deadline, we declare the given set of messages to be unschedulable; otherwise we report the worst-case response time.

Computing the worst-case response time of a message transmitted on the FlexRay bus consists of several components [11], [16]. For simplicity of exposition, we assume that $D_i \leq T_i$. However, this is not a restriction on our proposed method and in Section VIII, we will discuss how our proposed algorithm can be extended to case where $D_i > T_i$. Similarly, it is easy to accomodate jitter suffered by messages into our framework and this will be accounted for in Section VIII.

The worst-case response time $\mathcal{WCRT}_i$ of a message $m_i$ consists of the following components.

$$\mathcal{WCRT}_i = \sigma_i + w_i + C_i \tag{1}$$

The first component $\sigma_i$ is the worst-case delay that a message can suffer during the first FlexRay cycle where the message $m_i$ is generated. To compute $\mathcal{WCRT}_i$, we are interested in the scenario where $\sigma_i$ is maximum. Let the set of high priority messages be denoted as $hp(m_i) = \{m_1, m_2, \cdots, m_N\}$. Now the worst-case, scenario occurs if $m_i$ arrives just after the corresponding minislot starts and no higher priority message $hp(m_i)$, was transmitted in this FlexRay cycle. The value of $\sigma_i$ can be computed as follows:

$$\sigma_i = l_{FC} - (l_{ST} + (ID_i - 1)l_{MS}) \tag{2}$$

Note here that $\sigma_i$ can be computed with the straightforward algebraic equation from the system model.

The second component, $w_i$ is essentially the delay caused to $m_i$ by the higher priority messages. $w_i$ is the summation of two terms:

$$w_i = busCycles_i + lastCycle_i \tag{3}$$

In the above equation, $busCycles_i$ is the total number of cycles message $m_i$ has to wait due to interference by higher priority messages and $lastCycle_i$ is the time interval from the start of the last cycle to the beginning of the transmission in that cycle. The value of $lastCycle_i$ can be bounded by considering the last possible moment when $m_i$ can be sent in the FlexRay cycle which is defined by the value of $pLatestTx$. $pLatestTx$ is specified as a part of the FlexRay configuration in the system model. The computation of $busCycles_i$ will be detailed in the following section. Let us denote the minimum amount of communication $\phi_{m_i}$ (in minislots) that needs to exists in a cycle $l$ such that the message $m_i$ is delayed into the next cycle $l+1$. $\phi_{m_i}$ can be computed based on the value of $pLatestTx$. For instance, if $pLatestTx$ is equal to $N_{MS}$, then $\phi_{m_i}$ can be computed as follows.

$$\phi_{m_i} = \phi_{m_i} + 2 - (W_{m_i} + F_{m_i}) \tag{4}$$

The last component $C_i$ of $\mathcal{WCRT}_i$, as shown in Equation 1, is the time needed by the message to be transmitted completed when, finally, it gains access to the bus and this can be computed as $C_i = l_{MS} \times W_i$.

## VI. Timing Analysis: Without Slot Multiplexing

In the above discussion, $busCycles_i$ is the only component for which we have not presented the computation technique. This will be detailed in this section. For clarity of exposition, we will first assume that slot multiplexing is not allowed by FlexRay. However, subsequently, we describe how $busCycles_i$ can be computed by our proposed approach assuming slot multiplexing is allowed on the DYN segment. Note that the calculation of the rest of the components of $\mathcal{WCRT}_i$ remain exactly same as described in Equations 1 to 3 in both cases — with and without slot multiplexing.

As mentioned before, $busCycles_i$ is the maximum number of cycles that a message $m_i$ can be delayed by the higher priority messages. An outline of our algorithm to compute $busCycles_i$ for each message $m_i$ is listed in Algorithm 1. Starting with the first cycle, i.e., $l = 1$, the algorithm iteratively tries to fill cycle $l$ with instances of higher priority messages and if it succeeds the algorithm will try to fill cycle $l+1$ and so on (lines 4 to 9). If the algorithm cannot fit all the instances

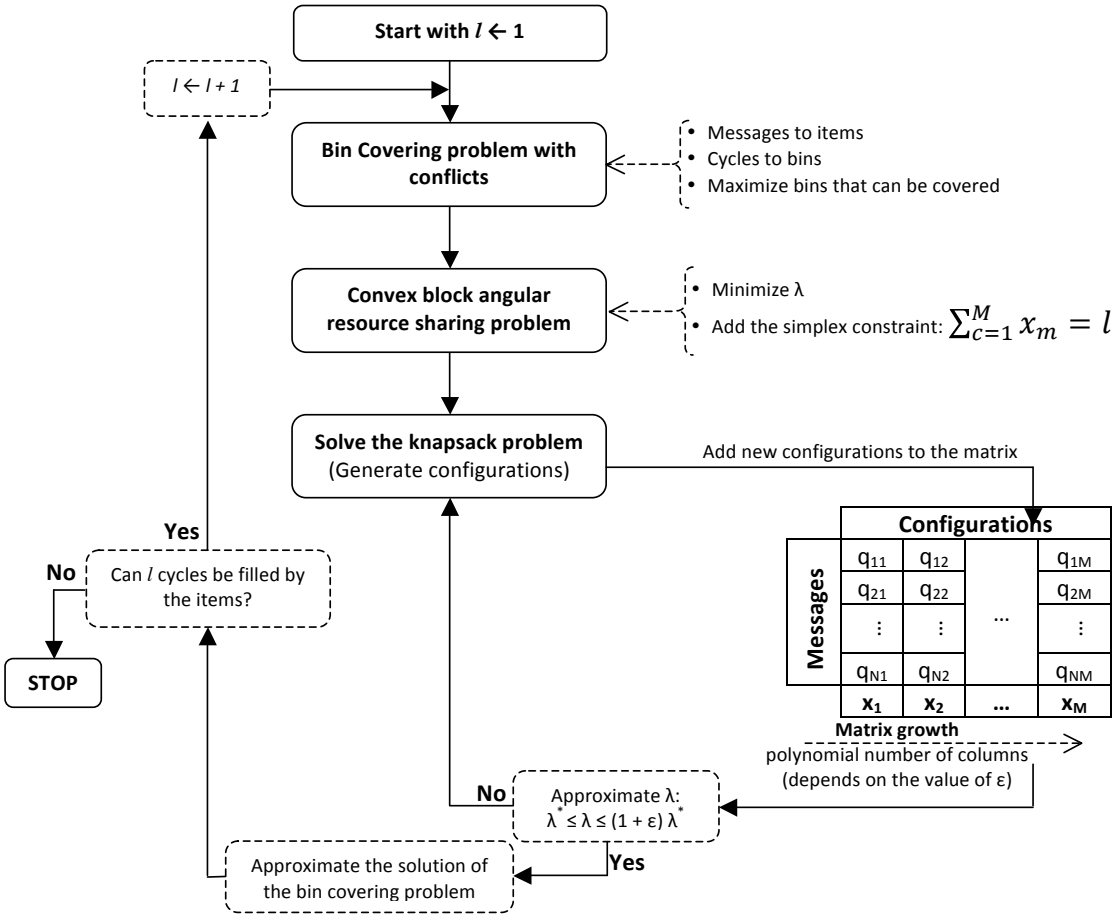


Fig. 3: Logical transformation of the problem of computing $busCycles_i$.

within $dCycle_i$ cycles for any message $m_i$, then it terminates and declares that the given message set $\Gamma$ is not schedulable (lines 9 to 17). $dCycle_i$ is computed directly from the deadline as an upper bound the relative number of cycles based on the lenght of the deadline (line 3). Otherwise, if $l \leq dCycle_i$ and the algorithm can fill completely $l-1$ cycles but not the $l$th cycle, the Algorithm 1 will report that the value of $busCycles_i$ is $l-1$.

The largest number of cycles that can be filled to the minimum level $\phi_{m_i}$ by higher priority messages from the set $hp(m_i)$ is essentially the value of $busCycle_i$. Let $k_h^l$ be the number of instances of message $m_h$ ($m_h \in hp(m_i)$) that are generated during $l$ consecutive cycles. If the algorithm manages to fill $l$ cycles, then the number of higher priority messages that need to be packed first needs to be recomputed as $k_h^{l+1}$ (line 6) in the next iteration.

At any iteration, the problem of filling $l$ cycles is essentially a bin covering problem. Bin covering tries to maximize the number of bins that can be filled to a fixed minimum capacity using a given set of items. In our case, the instances of the higher priority messages that are produced during the $l$ consecutive bus cycles are the items and the DYN segment is the bin where $\phi_{m_i}$ is the bin size that must be filled. Each message is considered as a separate item and the number of instances that are ready is considered as the number of copies of the same item. The minimum capacity of the bin that must be filled is $\phi_{m_i}$ as discussed in the previous section. Note, however, that in contrast to the traditional bin covering problem, in our case only one copy of each type of item is allowed to be packed in the same bin. This follows from the FlexRay protocol specification (see Section II) that allows only one instance of each message to be transmitted in each DYN segment cycle. Finally, the objective of this bin covering problem is to maximize the total number of bins that can be covered. Note that we are solving a decision version of this problem because we want to know whether the $l$ bins can be filled or not.

We solve the resulting bin covering problem based on the technique presented by Jansen and Solis-Oba [7]. Following their approach, a high-level scheme of our technique is illustrated in Figure 3. As shown in this figure, we first transform the bin covering problem into a *convex block angular resource*

*sharing problem* (see Section VI-A). This problem is solved by the price directive decomposition method which, in turn, must solve a knapsack like problem at its heart. This is discussed in Section VI-B.

**Algorithm 1** Computing the $busCycles_i$ for message $m_i$ for the case of no Slot Multiplexing

**Require:** The message $m_i$ ($m_i \in \mathbf{\Gamma}$), the set $hp(m_i)$ ($hp(m_i) \subseteq \mathbf{\Gamma}$), and system parameters of messages in the set $\mathbf{\Gamma}$

1: **for all** $m_i \in \mathbf{\Gamma}$ **do**
2: schedulable = false
3: $dCycle_i = \left\lceil \frac{D}{l_{FC}} \right\rceil$
4: **for** $l = 1 \rightarrow dCycle_i$ **do**
5: **for all** $m_h \in hp(m_i)$ **do**
6: $k_h^l = \left\lceil l \frac{f}{T_h} \right\rceil$
7: **end for**
8: Solve the bin covering problem using the logical transformation presented in Figure 3
9: Let $P(\epsilon)$ be the approximate solution of the bin covering problem
10: **if** $P(\epsilon) < l$ **then**
11: schedulable = true
12: **return** $l-1$ as the value of $busCycles_i$
13: **end if**
14: **end for**
15: **if** schedulable == false **then**
16: **return** the set $\mathbf{\Gamma}$is not schedulable
17: **end if**
18: **end for**

### A. Step I

As a first step, we formulate the bin covering problem as a Integer Linear Program (ILP). Without any loss of generality, we drop the subscript $i$ from $m_i$ but the interpretation remains same if the subsript is used. The set of high priority messages is $hp(m)$ and let $N = |hp(m)|$. Considering the messages instances in $hp(m)$ as items, we define a bin configuration $\mathcal{C}$ to be any subset of items from the set $hp(m)$ such that the items in $\mathcal{C}$ can cover the bin. Let the set of all possible bin configurations be $\zeta = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_{|\zeta|}\}$. For any configuration $\mathcal{C}_c$, let there be a set of boolean variables, $\left\{q_{1,c}, q_{2,c}, \cdots, q_{|\mathcal{C}|,c}\right\}$ that are appropriately set to 1 in order to represent the messages that are in the set $\mathcal{C}_c$. Note that $N = |\mathcal{C}|$. Given the above, a bin is covered if:

$$\sum_{n=1}^{|\mathcal{C}|} q_{n,c} \times (W_n - 1) \geq \phi_m \tag{5}$$

Thus, by definition, any $\mathcal{C}_c$ consists of items that satisfy the above equation.

Let there be an integer value $x_c$ associated with each configuration $\mathcal{C}_c$ that denotes how many times the configuration $\mathcal{C}_c$ occurs in the final solution. Let $M = |\zeta|$. The optimization goal of the ILP formulation is to maximize the sum of the integer variables $x_c$. We note that formulating the problem in this manner relies on the brute-force construction of the $M$ configurations as discussed above. In the worst case $M = 2^N$ and hence, generating all such feasible bin configurations grows exponentially with the value of $N$. However, as we will discuss later only a few bin configurations need to be generated in order to achieve an approximate upper bound on the solution to the bin covering problem. This configurations must be generated while adhering to FlexRay standard as will be discussed in Section VI-B.

The ILP problem is formulated below:

$$\begin{aligned} \text{maximize:} \quad & \sum_{c=1}^{M} x_c \\ \text{subject to:} \quad & \sum_{c=1}^{M} q_{n,c} x_c \leq k_n^l, \quad \forall n \in \{1, 2, \cdots, N\} \\ & x_c \in \mathbb{N}_+, \qquad\qquad \forall c \in \{1, 2, \cdots, M\} \end{aligned} \tag{6}$$

Recall that in our problem, the bin covering problem manifests itself in the form of a decision problem, i.e., in each iteration of the Algorithm 1, we check whether $l$ bins can be covered before moving to the next iteration where we check whether $l+1$ bins can be filled. We also relax the integrality constraint on the variables $x_c$ to obtain a linear program. Hence, we can re-write the problem in Equation 6 in a new form as presented below:

$$\lambda^* = \min \left\{ \lambda \middle| \begin{array}{l} \sum_{c=1}^{M} \frac{q_{n,c}}{k_n^l} x_c \leq \lambda, \forall n \in \{1, 2, \cdots, N\} \\ \sum_{c=1}^{M} x_c = l \end{array} \right\} \tag{7}$$

Note that an optimal solution of this problem with $\lambda^* = 1$ is an optimal solution of the LP relaxation obtained from Equation 6. As noticed in [7] this new problem (Equation 7) is a *convex block-angular resource sharing problem* and the price directive decomposition method [8] can be used to solve it with any given precision $\epsilon > 0$. In fact, the algorithm presented in [8] has been proved to return a solution with a bound $(1+\epsilon)\lambda^*$. In this paper, we will deploy the same algorithm to solve our bin covering problem. In the following, we provide a brief description on how this algorithm [8] works.

**Background**: Let us define with $\mathcal{X}$ the set of all possible vectors such that $\mathcal{X} = \left\{ (x_1, x_2, \cdots, x_M) \in \mathcal{R}_+^M \middle| \sum_{c=1}^{M} x_m = l \right\}$. Note that $\mathcal{X}$ is a simplex by construction. We introduce the following notations for a vector $X = (x_1, x_2, \cdots, x_M) \in \mathcal{X}$:

$$f_n(X) = \sum_{c=1}^{M} \frac{q_{n,c}}{k_n^l} x_c \text{ and } \lambda(X) = \max_{n=1}^{N} f_n(X) \tag{8}$$

Let $\mathcal{F} = (f_1, f_2, \cdots, f_N)$. The algorithm in [8] computes a solution $X \in \mathcal{X}$ such that:

$$\textbf{Primal}_\epsilon\text{: } \mathcal{F}(X) \leq (1+\epsilon)\lambda^* \times I \tag{9}$$

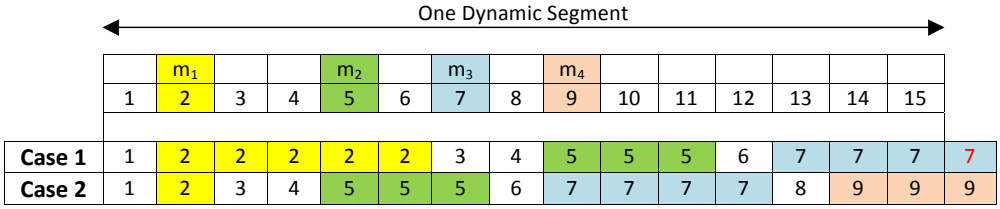


Fig. 4: The knapsack problem has a constraint corresponding to each higher priority message.

where $I = (1, 1, \cdots, 1)$ (the unit vector with $N$ elements). The approach is based on the Lagrangian duality relation:

$$\lambda^* = \min_{X \in \mathcal{X}} \max_{P \in \mathcal{P}} P^T \mathcal{F}(X) = \max_{P \in \mathcal{P}} \min_{X \in \mathcal{X}} P^T \mathcal{F}(X) \tag{10}$$

where $\mathcal{P} = \left\{ (p_1, p_2, \cdots, p_N) \in \mathcal{R}_+^N \middle| \sum_{n=1}^{N} p_n = 1 \right\}$ is the unit simplex. Denoting $\Lambda(P) = \min_{X \in \mathcal{X}} P^T \mathcal{F}(X)$ a pair $X \in \mathcal{X}$ and $P \in \mathcal{P}$ is optimal if and only if $\lambda(X) = \Lambda(P)$. The corresponding $\epsilon$-approximation dual problem has the form:

$$\textbf{Dual}_\epsilon \colon \Lambda(P) \geq (1 - \epsilon)\lambda^* \tag{11}$$

The price-directive decomposition method is an iterative strategy that solves the primal problem $\textbf{Primal}_\epsilon$ and its dual problem $\textbf{Dual}_\epsilon$ by computing a sequence of pairs $X$ and $P$ to approximate the exact solution from above and below, respectively. In [5] it has been shown that the primal and the dual problem can be solved with a $t$-approximate block solver that solves the block problem a given tolerance t. The block problem is:

$$\min_{Y \in \mathcal{X}} P^T \mathcal{F}(Y) \tag{12}$$

**Our problem:** In our case we choose $t = \epsilon$ and the block problem has the following form:

$$\min_{Y \in \mathcal{X}} \sum_{c=1}^{M} y_c \times \left( \sum_{n=1}^{N} \frac{q_{n,c}}{k_n^l} p_n \right) \tag{13}$$

We solve the above block problem in our context as the described in the following.

### B. Step II

The minimization problem obtained above in Equation 13 still contains a large number of variables because it assumes that the $M$ feasible bin configurations have been generated in a brute-force fashion. However, we overcome this problem by transforming this problem into a knapsack problem which, as mentioned before, is the final step in our algorithm. The set $\mathcal{X}$ is a simplex. Using the linear programming theory, we know that for the case when the integrality constraints are removed a minimization problem always has its optimum in one of the corners of the simplex. The corners of the simplex $\mathcal{X}$ have only one variable $x_c = l$ and all the others are equal with 0. Therefore the minimization problem in Equation 13 transforms into:

$$l \times \min_{c=1}^{M} \left( \sum_{n=1}^{N} \frac{q_{n,c}}{k_n^l} p_n \right) \tag{14}$$

This problem has M variables where M can be $2^N$ in the worst-case. Hence, we will now approximate the problem, using the algorithm by jansen and Zhang [8]. By doing this we decrease the number of variables to a polynomial number $N$. The algorithm by Zhang takes as an input an $\epsilon$ value and generates a number of columns which leads to a solution with the desired quality. A higher value of $\epsilon$ would imply that less columns will have to be generated leading to fast running times. On the other hand, the pessimism in the solution increases. The opposite argument holds true when $\epsilon$ is assigned small values. Since our algorithm is based on this approach, a designer using our scheme can also choose an appropriate value of $\epsilon$ according to her/his desired quality.

This problem is a variation of the knapsack problem with additional constraints and the variables $q_{n,c}$ are now the optimization variables. Note that now we have only $N$ optimization variables compared with the initial number - $M$. What we have to solve is:

$$\begin{aligned} \text{minimize:} \quad & \sum_{n=1}^{N} \frac{q_{n,c}}{k_n^l} p_n \\ \text{such that:} \quad & \sum_{n=1}^{N} q_{n,c}(W_n - 1) \geq \phi_m \\ & \sum_{i=1}^{n-1} q_{i,c}(W_i - 1) \leq \phi_{m_n} - 1, \forall n \in \{2, 3, \cdots, N\} \\ & q_{n,c} \in \{0, 1\} \end{aligned} \tag{15}$$

This is a multiple-constraint knapsack problem and has several constraints that correspond to FlexRay specific details. As mentioned in the beginning of this section the problem of computing the worst case delay of a message $m$ is solved by doing a logical transformation into the bin covering problem. The core idea is that cycles can be logically transformed into bins. The constraints regarding how items should be packed into bins are given by the FlexRay specifications. In order to make the transformation reversible we have to take into consideration all such constraints. The first constraints ensures that the message under analysis will be displaced. The second constraint represents a set of $N-1$ constraints which ensures that each message $m_h \in hp(m)$ will not be displaced by its own set of high priority messages. The last constraint ensures that at the maximum only one instance of a given message will be transmitted in one cycle.

We will explain the significance of the second set of constraints with the help of one example. In our example presented in Figure 4 we show 4 messages. We are interested in the worst-case delay of message $m_4$. The figure shows 2 scenarios.

The first case presents a bin covered with copies of $m_1, m_2$ and $m_3$. We can see that this bin configuration cannot be transformed back into a FlexRay cycle because message $m_3$ is displaced by message $m_1$ and $m_2$ and this violates the FlexRay constraints. Note that $m_3$ is not the message under analysis but it is still important to encapsulate such FlexRay constraints. On the other hand, the configuration which corresponds to case 2 is a reversible one because message $m_2$ and $m_3$ are not displaced beyond $\phi_{m_2}$ and $\phi_{m_3}$ respectively.

**Discussion:** The overall complexity of the Algorithm 1 is:

$$\mathcal{O}\left(N \times \left(\frac{1}{\epsilon^2} + \log N\right) \times \Omega(N)\right) \tag{16}$$

where $\Omega(N)$ is the complexity of solving the above knapsack problem with $N$ elements where the knapsack has a capacity value equal with $\phi_m$. Note that this multiple-constraint knapsack problem, that appears at the heart of the overall problem, is solved optimally by a dynamic programming algorithm that runs in pseudo-polynomial time [9]. Note that that we can bound the number of such knapsack problems to be solve with $\mathcal{O}\left(N \times \left(\frac{1}{\epsilon^2} + \log N\right)\right)$.

As a detail in our algorithm we would like to note that for each simplex $\sum_{c=1}^{M} x_c = l$ a column reduction process is conducted because each simplex has exactly $N$ linear constraints. Therefore the maximum number of non-zero components (the maximum number of variables $x_c^l \neq 0$) cannot exceed the value of $N$. A way to transform a solution of the simplex into a one with at most $N$ non-zero components can be done using the *Singular Value Decomposition - SVD* algorithm [15].

Finally, note that our analysis will provide a safe result from the point of view that the actual worst-case delay of a message $m_i$ will always be smaller or equal compared with the results provided by our analysis. Broadly, allowing $x_c$ to have non-integral values and thereafter, not generating all the bin configurations are the only approximation schemes in our approach. The first, i.e., an LP relaxation that we perform always leads to pessimistic results. It is known from the theory of linear programming that for a maximization problem if one removes the integrality constraints will always get a solution which is an upper bound for the integral solution of the same problem [1]. Secondly, even if we generate only a few bin configurations, we follow the price decomposition method that guarantees upper bounds as proved previously [8].

## VII. Timing Analysis: With Slot Multiplexing

In the previous section, we transformed the problem of computing $busCycles_i$ into the bin covering problem [10]. However, for the case of slot multiplexing, the computation of $busCycles_i$ can not be transformed into the traditional bin covering problem. Rather, the computation of $busCycles_i$ becomes a problem that we call as the *bin covering problem with conflicts*. This is a direct consequence of the fact that the repetition rates of messages (see Section IV) allow each message to be transmitted only in certain FlexRay cycles within the repeating pattern of $CC_{max}$ cycles where the messages (items) have no *conflicts* with the cycles (bins).

To solve this problem, we need to suitably adapt the technique by Jansen and Solis-Oba [7]. First, we must transform this problem into a bin covering problem with conflicts. The transformation of messages and cycles into items and bins remains similar to the one in Section VI. In the context of slot multiplexing, this constraint becomes a conflict between an item (message) and a bin (cycle). In this sense, all bins are not of the same type — unlike the bins in the traditional case. Thus, there are conflicts between items and bin types, and it is under this condition that the number of bins that can be filled must be maximized.

Let us consider an example with 5 messages. The values of the relevant parameters for these 5 messages are presented in Table I. Following these parameters Figure 6 shows the cycles where the 5 messages may be submitted. We are interested in computing the value of $busCycles_5$, i.e., we want to compute the number of cycles that message $m_5$ can be delayed in the worst-case by higher priority messages. Let us consider that the length of the FlexRay cycle is $l_{FC} = 4$ ms, and that in the present iteration of our algorithm , we want to check whether $m_5$ will be delayed for 9 cycles, i.e., $l = 9$.

We start by observing that an instance of $m_5$ can be sent on the bus only in cycles 0, 2, 4, and 5. This follows from the specifications in Table I. Secondly, we observe that the cycles with same counter that appear in two different DYN segments are similar. For instance, cycle 0 in both DYN cycles in the figure are similar from the point of view that only instances of messages $m_1, m_2, m_3$ and $m_4$ are allowed to be sent. Similarly, we see that cycles 2 and 6 are similar from the perspective that only instances of messages $m_1$ and $m_3$ are allowed to be sent. Finally, in cycle 4 only instances of messages $m_1, m_2$ and $m_3$ will be sent.

When connecting this observations to the bin covering problem with conflicts we have the following: cycles 0 will be identified as bin type 1, cycles 2 and 6 will represent the bin type 2 while cycle 4 will be of bin type 3. In the case without slot multiplexing, the decision problem of whether the message will be displaced by 9 cycles was same as whether 9 bins can be filled. In case of slot multiplexing, the question whether the message will be displaced by 9 cycles can be filled is equivalent to the question of whether different types of bins can be filled up to a minimum number or not. To understand this, let us look at Figure 6. Starting from cycle 0 (where $m_5$ is allowed) till cycle 0 in the next DYN segment, the message $m_5$ can be displaced for 9 cycles. Within this time interval, there are 2 bins of type 1, 2 bins of type 2 and one bin of type 3. However, $m_5$ displacement might also starting from cycle 2. In this case, we need to see if 3 bins of type 2 and one bin of type 1 and type 3 can be filled in order for the displacement to span 9 cycles. Hence, the decision problem must be solved for $m_5$ considering that the worst-case might occur while starting from any of the types of bin where $m_5$ is allowed. or each of these three cases the number of each type of bins that occur is not same. For example, If in any of

| Priority | Cycle 0 | Cycle 1 | Cycle 2 | Cycle 3 | Cycle 4 | Cycle 5 | Cycle 6 | Cycle 7 | Cycle 0 | Cycle 1 | Cycle 2 | Cycle 3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | m1 | | m1 | | m1 | | m1 | | m1 | | m1 | |
| 2 | m2 | | | | m2 | | | | m2 | | | |
| 3 | m3 | m3 | m3 | m3 | m3 | m3 | m3 | m3 | m3 | m3 | m3 | m3 |
| 4 | m4 | | | | | | | | m4 | | | |
| 5 | m5 | | m5 | | m5 | | m5 | | m5 | | m5 | |

Fig. 5: The cycles where messages are allowed to be transmitted.

these three cases the bins can be covered, we say that $m_5$ can be delayed for 9 cycles by higher priority messages.

We emphasize that the number of types of bin is limited by a constant number because the FlexRay standard limits the number of cycles allowed within a repeating pattern i.e., $CC_{max}$. This constant can be never more than 64 [3]. Moreover, extracting the minimum number of bins to covered for each type is straightforward given the system model.

To formally denote the distinct types of bins based on the repetition rates of the higher priority messages let us denote with $G$ the set of the types of different bins. Thus, $G = \{g_1, g_2, \cdots, g_P\}$ assuming there are $P$ types of bins. Each element $g_i \in G$ is associated with a value $h_{l,i}$ denoting for how many times this bin needs to be covered in order to have a total delay of $l$ cycles. As discussed, this is easily computed from the system model. For the previous example we have $G = \{g_1 = \{m_1, m_2, m_3, m_4\}, g_2 = \{m_1, m_3\}, g_3 = \{m_1, m_2, m_3\}\}$ with the associated variables $h_{l,1} = 2, h_{l,2} = 2$ and $h_{l,3} = 1$. The previous values correspond to the case when the worst case delay of message $m_5$ is assumed to start with cycle 1. At the same time if we want to see if the message can experience an higher worst case delay we will have to check for example the situation when we have as a starting point cycle 3. For this case, assuming that we want to answer the same question which is if 9 cycle can be filled the number of different types of bins is the same but how many bins of each type we have changes. In this case we will have $h_{l,1} = 1, h_{l,2} = 3$ and $h_{l,3} = 1$.

Once we have the problem reformulated as a bin covering problem, the first step of our algorithm is similar to the Step 1 as discussed in Section VI. Thus, the corresponding *convex block-angular resource sharing problem* has the following structure:

$$\lambda^* = \min \left\{ \lambda \middle| \begin{array}{l} \sum_{p=1}^{P} \sum_{c=1}^{M} \frac{q_{n,c}^p}{k_n} x_m^p \leq \lambda, \forall n \in \{1, 2, \cdots, N\} \\ \sum_{m=1}^{M} x_m^1 = h_{l,1} \\ \sum_{m=1}^{M} x_m^2 = h_{l,2} \\ \cdots \\ \sum_{m=1}^{M} x_m^P = h_{l,P} \end{array} \right\} \quad (17)$$

Similar to Section VI this problem can be now solved using the parallel price directive decomposition. Step 2 in Section VI, however, solves only one knapsack which is not the case any more in the context of slot multiplexing. In contrast, in this case, we now need to solve $P$ knapsack problems where

| | Period | Repetition Rate | Base Cycle |
|---|---|---|---|
| $m_1$ | 10 ms | 2 cycles | 1 |
| $m_2$ | 18 ms | 4 cycles | 1 |
| $m_3$ | 8 ms | 1 cycle | 1 |
| $m_4$ | 48 ms | 8 cycle | 1 |
| $m_5$ | 12 ms | 2 cycle | 1 |

TABLE I: Message Parameters

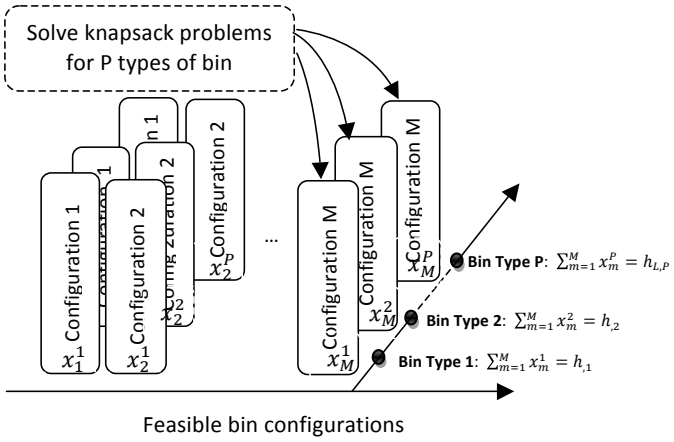


Fig. 6: In the case of slot multiplexing there are *types* of bins that influence the knapsack problems that need to solved.

$P$ is number of the types of bins as shown in the Figure 6.

## VIII. Discussion

We presented a schedulability analysis for messages on the DYN segment of FlexRay based on response time analysis. For the simplicity of exposition we assumed that the deadline of each message is less than its period. However, our analysis can be extended to the general case as well. This is similar to the analysis of delays of CAN bus messages [2] and previous papers on analyzing messages on DYN segment have followed this approach as well.

Towards this, we note that the response time analysis of the DYN segment messages is based on the calculation of the level-$i$ busy period. The busy period is the worst-case time in which the bus is always busy transmitting messages with priority higher than or equal to $m_i$. The busy period starts from the critical instant for an instance of when $m_i$ is queued at t=0 with jitter $J_i$. The busy period ends when all instances of messages with priority higher than or equal to $m_i$ that were queued during the busy period have been transmitted. The number of instances $Q_i$, of message $m_i$ that become ready

for transmission before the end of busy period is given by

$$Q_i = \left\lceil \frac{t_i + J_i}{T_i} \right\rceil \tag{18}$$

The worst-case response time of the message $m_i$ is the maximum value from among the response time of each of these $Q_i$ instances of $m_i$. The first instance in the busy period be $q = 0$ and the last instance is $q = Q_i - 1$. If the response time of instance $q$ is given by

$$\mathcal{WCRT}_i = \max_{q=0...Q-1}(\mathcal{WCRT}_i(q)) \tag{19}$$

The response time, $\mathcal{WCRT}_i(q)$ of the $q$th instance is given by:

$$\mathcal{WCRT}_i(q) = w_i(q) - (qT_i - J_i) + C \tag{20}$$

In the above equation, $w_i(q)$ is the longest time from the start of busy period to the time when the instance $q$ begins it transmission on the bus. $qT_i - J_i$ is the time interval relative to the start of the busy period after which the $q$th instance starts transmission. Thus, $w_i(q) - (qT_i - J_i)$ denotes the time that the $q$th instance is queued before transmission. $C_i - l_{MS}$ denotes the transmission time of the message on the bus. In this equation, $q$ is the instance number and $T_i$, $J_i$, $C_i$ and $C$ are components of the system model and hence, these values are known to us. The computation of the busy period and the computation of $w_i(q)$ essentially follows the same approach as listed in Algorithm 1. A complete description of the details is out of scope of this paper.

## IX. Experimental Results

**Experimental setup:** We implemented our proposed framework in Matlab. All the experiments were conducted on a Windows XP machine running a 4-core Xeon(R) 2.67 GHz processor. The test cases have been randomly generated by varying the message parameters like the periods, lengths, repetitions rates and base cycles in order to cover a wide range of possible scenarios. In all experiments we have assumed that the deadlines are equal with the periods. The length of the ST segment was set to be equal to 2 $ms$, while the number of minislots inside the dynamic segment was varied between 50 and 150 minislots. These values are in conformance with the FlexRay specification. We have assumed that the length of one minislot is equal to 12 $\mu s$. The repetitions rate have been generated such that the values represent a power of 2.

Broadly, we conducted two sets of experiments. For the first set, we compare the quality of results of our method and in the second set of experiments we evaluate the running times of our algorithm.

### A. Quality of our results

As discussed in Section I, the best known results on timing analyis of DYN segment were reported by Zeng et al. [16]. Hence, we compare our results with the framework presented by Zeng et al. [16]. Note, however, our scheme can yield results with varying degree of pessimism based on the input $\epsilon$. For large values of $\epsilon$, our algorithm returns more pessimistic values although it can run faster. On the other hand, for smaller values of $\epsilon$, the results are more accurate but it incurs longer running times. We consider this to be a significant advantage over existing techniques for timing analysis for FlexRay DYN segment.

| 20 Messages | Minislots | Our scheme | Previous work [16] |
|---|---|---|---|
| | 90 | 3 | Inf |
| | 100 | 2 | 7 |
| | 110 | 2 | 4 |
| | 120 | 2 | 3 |
| 25 Messages | Minislots | Our scheme | Previous work [16] |
| | 90 | 4 | Inf |
| | 100 | 3 | Inf |
| | 110 | 3 | 12 |
| | 120 | 3 | 7 |
| 30 Messages | Minislots | Our scheme | Previous work [16] |
| | 90 | 5 | Inf |
| | 100 | 5 | Inf |
| | 110 | 3 | Inf |
| | 120 | 3 | Inf |
| | 130 | 2 | 15 |
| | 140 | 2 | 7 |
| | 150 | 2 | 5 |
| 35 Messages | Minislots | Our scheme | Previous work [16] |
| | 90 | 6 | Inf |
| | 100 | 5 | Inf |
| | 110 | 5 | Inf |
| | 120 | 3 | Inf |
| | 150 | 2 | 14 |

TABLE II: In order to evaluate the quality of the results obtained by our framework, we compare the results with an existing method.

For comparing the quality of the resuts we choose $\epsilon = 1/8$. The rationale behind this is that for this value of $\epsilon$, our algorithm can run within a matter of few minutes and is scalable. Further experimental results on the running times will be reported in the next section.

Since the computation of the $busCycles_i$ is the most important component in the timing analysis of FlexRay DYN segment for our technique and the one by Zeng et al. [16], we compare $busCycles_i$ for both techniques. In Table II, we report the worst-case delays reported by both the frameworks for the lowest priority message in 4 different message sets whose sizes ranged from 20 to 35. The first observation from the table is that our scheme always performs better than the previous algorithm. Secondly, note that for each message set, as we increase the bandwidth, i.e., the number of minislots that are in the DYN segment, both methods report lesser worst case delay. In particular, the existing method reports infinite worst-case delay for several instances of the problem. However, in such cases, our algorithm returns a finite number. These results show that as the problem becomes tight, our algorithm will be able to find solutions while previous algorithms will be pessimistic and return non-schedulable solutions.

As discussed in Section I, the only known work to analyze delay in presence of slot multiplexing has been reported by Schneider et al. [13], [14]. However, their technique is severely limited because they cannot compute delays for messages that

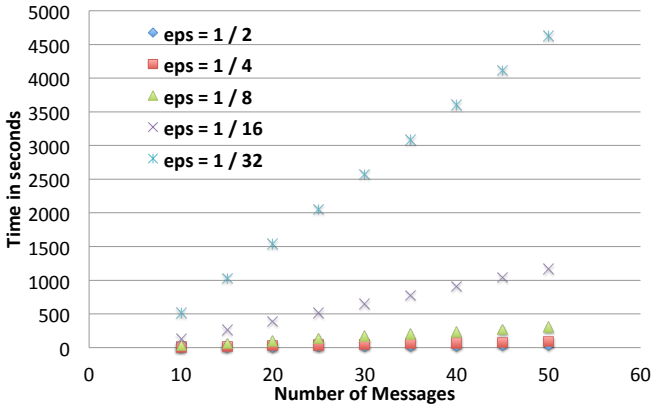


Fig. 7: The running times of our proposed algorithm for various values of $\epsilon$.

are displaced over multiple cycles and report such messages to be unschedulable. For example, all the results that we reported above had displacements of over one cycle and the technique proposed by Schneider et al. [13], [14] would have reported negative results. Note that the case of no slot multiplexing is essentially a special case of slot multiplexing where the repetition rate for each message is one and the base cycle is the first cycle in the FlexRay communication cycle.

We note that the focus of their papers [13], [14] is on synthesizing message schedules and evaluating other properties and they use a simplistic delay analysis model within such a framework. In future, it will be interesting to integrate the framework proposed by our paper into such existing schemes that rely on a pessimistic analysis.

### B. Running times

As mentioned in the previous section, our algorithm takes $\epsilon$ as an input from the system designer as an input and that various values of $\epsilon$ would lead to different running times. In this section, we increase the value of $\epsilon$ from $1/32$ to $1/2$ with 3 other values in between and show how the running times decrease. These running times are shown in Figure 7. As clearly observed here values of $\epsilon$ values of $1/8$, $1/4$ and $1/2$ scale quite well with the problem size. Note that values of $1/16$ and $1/32$ will yield better results than the ones we presented in the previous section where with $1/8$ our technique outperformed existing methods by significant margin. Hence, from our experiments, we believe that an $\epsilon$ value of 1/8 strikes the right balance between efficiency and quality.

## X. Concluding Remarks

In this paper, we deployed new theoretical results and applied it to perform schedulability analysis for the DYN segment of FlexRay. Our proposed scheme now allows us to perform schedulability analysis in presence of slot multiplexing and this opens up the possibility of configuring new parameters like the repetition rate. This gives the designer more flexibility to optimize his design. For illustration, let us consider the second example that was shown in Figure 2(a). In this example, if we change the repetition rate of $m_2$ from 1 to 2, $m_2$ is not allowed to be transmitted in cycle 1 and cycle 3. Other parameters in the example remain the same. Now, let us consider the worst-case response time of message $m_3$. Unlike the scenario in Figure 2(b), $m_2$ now cannot be transmitted in cycle 3. Thus, $m_3$ will be transmitted in cycle 3 thus suffering less delay compared to the case where $m_2$ had the repetition rate of 1. Of course, this also means that $m_3$ but it shows that the designer has more flexibility of configuring the delays of the messages.

## References


[1] M. S. Bazaraa, J. J. Jarvis, and H. D. Sherali. *Linear Programming and Network Flows*. Wiley-Interscience, 2004.

[2] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien. Controller area network (CAN) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems*, 35(3):239–272, 2007.

[3] The FlexRay Communications System Specifications, Ver. 2.1. www.flexray.com.

[4] E. Fuchs. FlexRay beyond the consortium phase. In *FLEXRAY, Special Edition Hanser automotive*, 2010.

[5] M. D. Grigoriadis and L. G. Khachiyan. Coordination complexity of parallel price-directive decomposition. *Math. Oper. Res.*, 21, 1996.

[6] A. Hagiescu, U. D. Bordoloi, S. Chakraborty, P Sampath, P. V. V. Ganesan, and S. Ramesh. Performance analysis of FlexRay-based ECU networks. In *DAC*, 2007.

[7] K. Jansen and R. Solis-Oba. An asymptotic fully polynomial time approximation scheme for bin covering. *Theor. Comput. Sci.*, 306(1-3), 2003.

[8] K. Jansen and H. Zhang. Approximation algorithms for general packing problems with modified logarithmic potential function. In *International Conference on Theoretical Computer Science*, 2002.

[9] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack problems*. Springer, 2004.

[10] M. Labbe, G. Laporte, and S. Martello. An exact algorithm for the dual bin packing problem. *Operations Research Letters*, 17(1), 1995.

[11] 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.

[12] K. Schmidt and E. G. Schmidt. Schedulability analysis and message schedule computation for the dynamic segment of FlexRay. In *Vehicular Technology Conference*, 2010.

[13] R. Schneider, U. D. Bordoloi, D. Goswami, and S. Chakraborty. Optimized schedule synthesis under real-time constraints for the dynamic segment of FlexRay. In *International Conference on Embedded and Ubiquitous Computing*, 2010.

[14] R. Schneider, D. Goswami, S. Chakraborty, U. D. Bordoloi, P. Eles, and Z. Peng. On the quantification of sustainability and extensibility of FlexRay. In *DAC*, 2011.

[15] L.N. Trefethen and D. Bau. *Numerical linear algebra*. 1997.

[16] H. Zeng, A. Ghosal, and M. D. Natale. Timing analysis and optimization of FlexRay dynamic segment. In *International Conference on Computer and Information Technology*, 2010.