

# On the Timing Analysis of the Dynamic Segment of FlexRay

Unmesh D. Bordoloi    Bogdan Tanasa    Petru Eles    Zebo Peng  
Linköpings Universitet, Sweden  
e-mail: {unmesh.bordoloi, bogdan.tanasa, 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. A communication cycle in FlexRay consists of an event-triggered component known as the dynamic (DYN) segment, apart from a time-triggered segment. Predicting the worst-case response time of messages transmitted on the DYN segment is a difficult problem. This is because a set of complex rules, apart from the priorities of the messages, govern the DYN segment protocol. In this paper, we survey techniques for the timing analysis of the DYN segment. We discuss the challenges associated with the timing analysis of the FlexRay protocol, the proposed techniques and their limitations.

## I. INTRODUCTION

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 [6] of automotive companies. In fact, cars equipped with FlexRay are already in the streets or in production [7]. 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.

It should be noted here that the argument for Ethernet as an automotive communication protocol is also gaining traction [1]. However, it is not expected to replace FlexRay. Rather, Ethernet and domain specific protocols like FlexRay are expected to co-exist in automotive electronic systems and inter-connected via gateways. Hence, timing analysis and scheduling for FlexRay continues to generate significant research interest.

FlexRay is a hybrid communication protocol, 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 ST segment is divided into several slots that appear in pre-defined temporal points. Each message to be transmitted over the ST segment is assigned a unique slot. A message may be transmitted only during its slot and this assures predictability of the response times of the messages. In contrast, the DYN segment resolves conflicts between messages based on priorities. Unlike the ST segment, the delay suffered by a message depends on the interferences by its higher priority messages. Computing the interferences for the higher priority messages is a challenging problem for the DYN segment of FlexRay.

In fact, Pop et al. have shown that it is like the bin covering problem, which is an NP-hard problem in the strong sense,

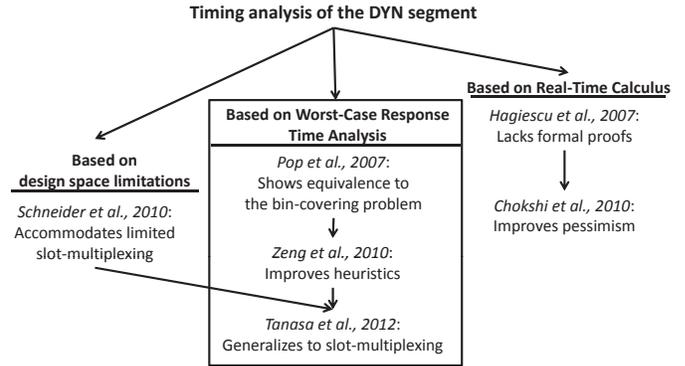


Fig. 1. Recent work on the timing analysis of the DYN segment.

in one of the first known work on formal timing analysis of the DYN segment of FlexRay [12], [11]. They also proposed heuristics to compute upper bounds on the worst-case response times of the messages which have been improved later on by Zeng et al [18]. These techniques have been built on top of worst-case response time analysis that iteratively computes the interference from the higher priority messages until a fixed point is reached. A separate thread of work (see Figure 1) by Hagiescu et al. [8] and, by Chokshi and Bhaduri [5] have attempted to compute the delays of messages on the DYN segment based on the Real-Time Calculus framework [4]. Section IV of this paper provides a more detailed discussion on both threads of work mentioned above.

The timing analysis of the DYN segment is even more difficult if slot multiplexing is considered. Slot multiplexing refers to the fact that two different messages can share the same priority. This feature of FlexRay will be discussed in detail in Section II. The initial papers [11], [12], [18], [8], [5] on FlexRay ignored this feature. Very recently, however, there have been attempts to address this issue. Schneider et al. [14], [15] proposed an approach that accommodates slot multiplexing by restricting the priorities that may be assigned to messages and thereby enforcing that the interference from the higher priority messages is limited to one cycle. This is a very pessimistic approach and recently, we overcome this limitation [16]. Our technique [16] is quite general and it can estimate message delays that span over multiple cycles. For the case of slot multiplexing, the timing analysis problem can not be transformed in to the traditional bin covering problem. Rather, the problem becomes what we call the *bin covering problem with conflicts* [16]. Moreover, we showed that, even

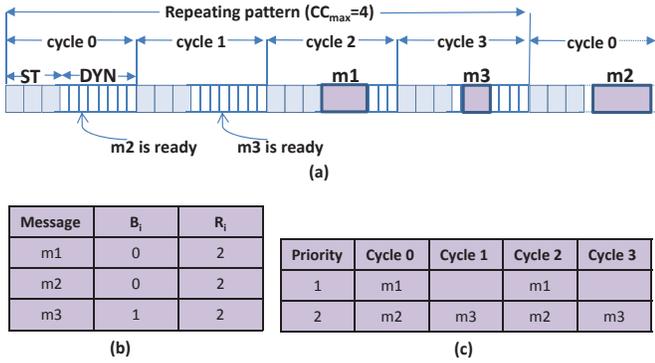


Fig. 2. Example 1: Messages  $m_1$  and  $m_2$  are multiplexed in FlexRay DYN segment.

for the case where slot multiplexing is ignored, the results obtained by our scheme [16] are significantly better than the state-of-the-art [18].

In this paper, our thrust will be on the thread of work that proposes heuristics for the bin covering problem, as highlighted by a box in Figure 1. We will also mention other techniques and discuss their limitations. It should be noted here that Schmidt and Schmidt [13] have also proposed an Integer Linear Programming (ILP) based formulation of the timing analysis problem in order to compute the response time of messages on the DYN segment. However, we will not discuss this here because they did not propose any heuristic for the bin covering problem. ILP-based solutions help in obtaining the optimal solution, but they suffer from scalability problems because the bin covering problem is NP-hard.

## II. THE FLEXRAY DYNAMIC SEGMENT

The FlexRay communication protocol [6] is organized as a periodic sequence of communication cycles with fixed length,  $l_{FC}$ .

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 2(a) illustrates a FlexRay communication pattern with  $CC_{max} = 4$ . In the figure, the cycle counter starts from 0, goes till  $CC_{max} - 1 = 3$ , and then, it is reset to 0.

Each message 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. For the FlexRay cycle illustrated in Figure 2(a), let us consider three messages  $m_1$ ,  $m_2$  and  $m_3$  to be transmitted over the DYN segment. Let the base cycles be  $B_1 = B_2 = 0$  and  $B_3 = 1$  and let the repetition rates be set to  $R_1 = R_2 = R_3 = 2$ . These parameters are listed in Figure 2(b). Figure 2(c) shows the cycles where

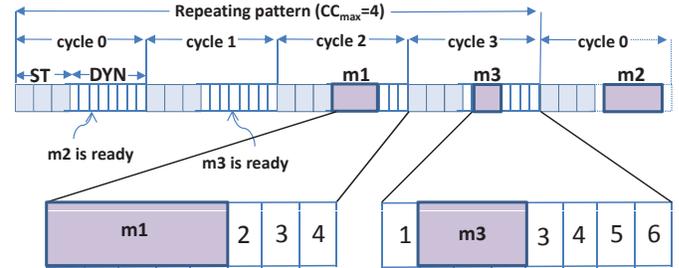


Fig. 3. Illustrating the incrementing slot counter for the DYN segment. A minislot expands into one larger slot if the message with corresponding priority is transmitted.

$m_1$ ,  $m_2$  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 may be transmitted every alternate cycle.  $m_1$  may be transmitted in the same cycles as  $m_2$  because they have the same repetition rate and base cycle.

Each communication cycle is further subdivided into a ST and a DYN segment. The ST segment follows a time-triggered communication paradigm. In the following we discuss the DYN segment in more detail. Conflicts between messages mapped to the same DYN segment are resolved using priorities as each message is assigned a fixed priority. In the above example,  $m_1$  has the highest priority while  $m_2$  and  $m_3$  have lower priority. Messages that may be sent in different cycles may be assigned the same priority and this is called slot-multiplexing. In the above example,  $m_2$  and  $m_3$  are said to be slot multiplexed.

According to the FlexRay standard, the base cycle  $B_i \in [0 \dots CC_{max} - 1]$ , and  $B_i < R_i$ . The relation  $B_i \in [0 \dots CC_{max} - 1]$  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”. A slot counter counts the number of slots in the DYN segment. At the beginning of each DYN segment, the message with priority 1 gets access to the bus. It occupies the required number of minislots on the bus according to its size and the slot counter increments only by one. 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 this case as well, the slot counter is incremented by one. The bus is then given to the next highest-priority message (with priority 2) if it is ready and the same process is repeated until the end of the DYN segment. Further, at most one instance of each message is allowed to be transmitted in each FlexRay cycle. Consider our running example that is now shown in Figure 3. 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.

The slot counter, as shown in the figure, is incremented by one after  $m_1$  is transmitted. In cycle 3, however, there is no message with priority 1 that is ready and hence, one minislot is wasted. Then, the slot counter is incremented to 2.  $m_3$  with priority 2 is ready and hence, it may be now transmitted and it occupies 3 minislots.

**Challenges:** Compared to other fixed priority based protocols, like the CAN [3] bus, timing analysis of the DYN segment is inherently difficult. This is because, in the DYN segment, there is the possibility that, even if a message is ready and the bus is idle, the message is not given access to the bus. This is not the case in protocols like CAN, and is possible in FlexRay because of the following features.

First, at most one instance of a message can be sent in each DYN segment. Second, if a DYN segment message is generated by its sender task after the slot has started, it has to wait until the next bus cycle starts to get access to the bus. Finally, a message can be sent only if it fits into the remaining portion of the current DYN segment, i.e., a message can not straddle two communication cycles.

### III. SYSTEM MODEL

In this paper, we assume that system model consists of the specification of the FlexRay bus and the set of messages to be transmitted on the DYN segment.

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  $l_{FC} = l_{ST} + l_{DYN}$ .

We assume that the set of messages  $\Gamma$  that will be transmitted on the FlexRay DYN segment is known. Any message  $m_i \in \Gamma$ , 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 given.
- 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 known. A higher value implies a lower priority.

### IV. TIMING ANALYSIS METHODS

In this section, we will discuss the timing analysis of the DYN segment when the feature of slot multiplexing is not used, i.e., the parameters  $B_i = 0$  and  $R_i = 1$  for all messages  $m_i$ . This essentially means that a message can be transmitted in any cycle, provided it is ready and it may fit into the

DYN segment bandwidth remaining after its higher priority messages have been transmitted in that cycle.

First, we will have a short discussion on the approaches based on Real-Time Calculus. This will be followed by a more detailed discussion on the approaches based on worst-case response time analysis.

#### A. Real-Time Calculus

Real-Time Calculus [4] uses abstract models to capture the timing properties of event streams, like periodically triggered messages and the capabilities of processing resources, like bus/processors. Timing properties of message arrivals are modeled by *arrival curves*, whereas the capabilities of the bus are represented by *service curves*. An arrival curve  $\alpha(\Delta)$  of an event stream is defined as an upper bound on the number of events seen in the stream within any time interval  $\Delta$ . The processing capabilities of a communication bus (or a processor) are usually expressed in number of bus (processor) cycles per time unit. Thus, a service curve  $\beta(\Delta)$  is defined as a lower bound on the number of cycles available to an event stream within any time interval  $\Delta$ . Using analytical equations from Real-Time Calculus, that are based on min-max algebra [2], an upper bound on the delay may be computed. This delay is essentially the worst-case response time which may be experienced by a message on the communication resource.

For a typical fixed priority based communication system, computing the service curves for messages follows directly from Real-Time Calculus fundamentals. The service curves for any message  $m_i$  is computed by an analytical expression as follows.

$$\beta_i(\Delta) = \sup_{0 \leq \lambda \leq \Delta} \{\beta_{i-1}(\lambda) - \alpha_{i-1}(\lambda)\} \quad (1)$$

For details, we refer the interested reader to [4]. Here, we only note that it is a closed form equation that takes as input the service  $\beta_{i-1}(\lambda)$  available to the higher priority message  $m_{i-1}$  and the arrival rate  $\alpha_{i-1}$  of the higher priority message  $m_{i-1}$ . Based on this, the service available to all messages from the highest priority to the lowest priority messages may be computed iteratively.

For modeling the DYN segment with Real-Time Calculus, however, this equation is no longer directly applicable and computing the available service  $\beta_i$  to a message  $m_i$  becomes a challenging problem. This is because of the following reason. In Real-Time Calculus abstraction it is assumed that active events, i.e., messages in the case of FlexRay, are processed in a greedy fashion in FIFO order by the resource, where the processing is restricted by the availability of resources. This means that if an instance of a message is ready to be transmitted on the bus, and resource is available, the instance of the message will be transmitted.

As we discussed in Section II, this property is not true for the DYN segment. Prior work [8], [5] on FlexRay attempted to circumvent this issue by proposing a set of algorithmic transformations to the service curve  $\beta_{i-1}$  to obtain the service

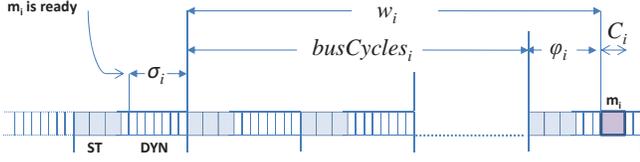


Fig. 4. The worst-case response time consists of three distinct components.

$\beta_i$ , available for the message  $m_i$ .

**Limitations:** However, the proposed transformations are not based on the max-min algebra which is at the root of the Real-Time Calculus. Rather, the transformations work on the curve  $\beta_{i-1}$ , taken as a geometric representation in Cartesian co-ordinates. [8], [5] claimed that the resulting service curve may be used as any other service curve within the Real-Time Calculus framework for the purposes of computing delay. However, they did not provide a formal proof that the transformed curve safely bounds the resource available from the DYN segment and hence, the correctness of their model cannot be formally guaranteed.

### B. Response Time Analysis

Advances on timing analysis of DYN segment have also been made based on the worst-case response time analysis approach [17], as discussed in Section I. In the following, we will focus on this line of work.

Computing the worst-case response time of a message transmitted on the FlexRay bus consists of several components [12], [18]. For simplicity of exposition, in this paper, we assume that  $D_i \leq T_i$ . However, this is not a restriction on the proposed methods and the details may be found in [12], [18].

The worst-case response time  $\mathcal{WCR}T_i$  of a message  $m_i$  consists of the following components. This is illustrated in Figure 4.

$$\mathcal{WCR}T_i = \sigma_i + w_i + C_i \quad (2)$$

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  $\mathcal{WCR}T_i$ , we are interested in the scenario where  $\sigma_i$  is maximum. Let the set of high priority messages be denoted as  $hp(m_i) = \{m_1, m_2, \dots, m_N\}$ . Now the worst-case scenario occurs if  $m_i$  arrives just after its 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:

$$\sigma_i = l_{FC} - (l_{ST} + (ID_i - 1)l_{MS}) \quad (3)$$

Thus,  $\sigma_i$  can be easily computed with the straightforward algebraic equation above that is based on parameters specified in 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:

$$w_i = busCycles_i + lastCycle_i \quad (4)$$

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.

The last component  $C_i$  of  $\mathcal{WCR}T_i$ , as shown in Equation 2, 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  $C_i = l_{MS} \times W_i$ .

**A Bin Covering Problem:** In the above discussion,  $busCycles_i$  is the only component for which we have not presented the computation technique. This will be detailed in the following. 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  $\mathcal{WCR}T_i$  remain exactly same as described in Equations 2 to 4 in both cases — with and without slot multiplexing.

At any iteration, the problem of filling  $l$  cycles is essentially a bin covering problem. This was shown by Pop et al [11] in the first paper to have addressed the timing analysis of the DYN segment. The bin covering problem is to maximize the number of bins that can be filled to a fixed minimum capacity using a given set of items, where each item is associated with a weight. Each message must be considered as a separate item and the number of instances that are ready as the number of copies of the same item. Each message is considered as a separate item. The minimum capacity of the bin that must be filled is  $\phi_{m_i}$ . It is defined as the minimum amount of communication  $\phi_{m_i}$  (in minislots) that needs to exist 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.

$$\phi_{m_i} = N_{MS} + 2 - (W_{m_i} + F_{m_i}) \quad (5)$$

Finally, the objective of this bin covering problem is to maximize the total number of bins that can be covered.

Following this observation, [12] used known heuristics for computing upper bounds of bin covering problems. These heuristics were originally presented for the bin covering problem and were presented in Labbe et al [10]. However, directly applying heuristics for the bin covering problem may lead to very pessimistic and potentially wrong results.

Consider, for example, the fact that, from the FlexRay protocol specification (see Section II), not more than one instance of the same message may be transmitted in the same DYN segment. The classic bin covering problem does not consider such constraints. Pop et al. [12] ignored this constraint in their method. This problem was identified and addressed by [18]. They consider each message as a separate item and the number of instances that are ready as the number of copies of the same item. Further, to accurately model the FlexRay DYN segment problem, the equivalent bin covering problem must have the condition that not more than one copy of the same item may be packed into the same bin.

**An iterative procedure:** For the classic bin covering problem, the number of items is fixed and the problem is to maximize the number of bins. For the problem of timing analysis of the DYN segment, however, the number of items, i.e., the number of instances of the higher priority messages depends on the number of bins, i.e., the number of cycles. This is because a given number of cycles corresponds to a particular time interval and hence, the time interval increases for each additional cycle/bin that is considered. The number of instances of each message depends on the time interval under consideration. Hence, the number of items must be recomputed for each additional cycle/bin. To accommodate this, the timing analysis for FlexRay DYN segment follows an iterative procedure as described in Algorithm 1.

Recall that  $busCycles_i$  denotes the maximum number of cycles that a message  $m_i$  may be delayed by the higher priority messages. An outline of an 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 8). If the algorithm cannot fit all the instances within  $dCycle_i$  cycles for any message  $m_i$ , then it terminates and declares that the given message set  $\Gamma$  is not schedulable (lines 14 to 15).  $dCycle_i$  is computed directly from the deadline as an upper bound the relative number of cycles based on the length 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, 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) for the next iteration.

The details of how the bin covering heuristic is solved may be found in [11], [18] and [16], where each has reported improvements over the previous one. The details of the algorithms are not the focus of this paper and we refer the interested readers to the papers for them.

**Limitations:** First, we note that [12] directly used the bin covering heuristics. As discussed above, this might lead to in-accurate results. Secondly, both [12] and [18] ignored slot multiplexing and this will be discussed in the following section.

---

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

---

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

- 1: **for all**  $m_i \in \Gamma$  **do**
- 2:    $schedulable = \text{false}$
- 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:        $k_h^l = \left\lceil l \frac{l_{FC}}{T_h} \right\rceil$
- 7:     **end for**
- 8:     Solve the bin covering problem
- 9:     Let  $P$  be the solution of the bin covering problem
- 10:     **if**  $P < l$  **then**
- 11:        $schedulable = \text{true}; busCycles_i = l - 1$
- 12:     **end if**
- 13:   **end for**
- 14:   **if**  $schedulable == \text{false}$  **then**
- 15:     The set  $\Gamma$  is not schedulable
- 16:   **end if**
- 17: **end for**

---

## V. GENERALIZATION TO SLOT MULTIPLEXING

In this section, we will discuss two recently proposed techniques that assume slot multiplexing is utilized.

### A. Restricted Approach

Schneider et al. [14] proposed a method to synthesize message schedules for the DYN segment of FlexRay. In essence, this implies that they were interested in synthesizing the parameters  $B_i, R_i, ID_i$  for each message  $m_i$  with the goal of optimizing certain cost functions.  $B_i, R_i$  are the base cycle and repetition rate of the message  $m_i$  as discussed in Section II.  $ID_i$  refers to the priority of the message. Thus, they focused on a design space exploration problem. However, at the core of their design space exploration problem, they performed timing analysis of the DYN segment in order to guarantee schedulability. This model of timing analysis of the DYN segment incorporated slot multiplexing but it was simplistic in the following sense.

The technique synthesizes message schedules that allocate only those priorities  $ID_i$  where message transmissions are guaranteed without the risk of displacement. Towards this, they compute a slot called  $S_{max}$ , which is the last slot in the DYN segment that may be assigned as a priority to any message. By assigning priorities  $ID_i \leq S_{max}$ , the schedule guarantees that the delay is safely bounded.  $S_{max}$  is a loose

upper bound that is computed as the sum of the message sizes that can be potentially mapped to that cycle of the DYN segment. While it is safe upper bound, this approach has two significant drawbacks.

**Limitations:** First, based on this timing model, any message with priority greater than the stipulated threshold  $S_{max}$ , will be assigned to have infinite delay. This is a very pessimistic approach because it is possible for several such messages to have finite delay and possibly, even schedulable. Secondly, the design space exploration scheme based on such models will lead to bandwidth wastage because the bandwidth beyond  $S_{max}$  will always remain unutilized.

Recently, we overcame this limitation for timing analysis of the DYN segment by accounting for slot multiplexing. We showed how the problem can be transformed into a general version of the bin covering problem and proposed a heuristic to solve the problem [16].

### B. New Approach

In Section IV-B, we discussed that the problem of computing  $busCycles_i$  can be converted into a 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 III) 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).

The transformation of messages and cycles into items and bins remains similar as discussed in Section IV-B. In the context of slot multiplexing, however, there is an additional constraint that 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 5 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 6. 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

	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

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. Once again, let us refer to Figure 5. 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 start from cycle 2. In this case, we need to verify 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. For each of these three cases the number of each type of bins that occur is not same. If in any of 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 never be more than 64 [6]. Moreover, extracting the minimum number of bins to be 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 the set of the types of different bins with  $G$ . Thus,  $G = \{g_1, g_2, \dots, g_P\}$  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  $h_{l,3} = 1$ .

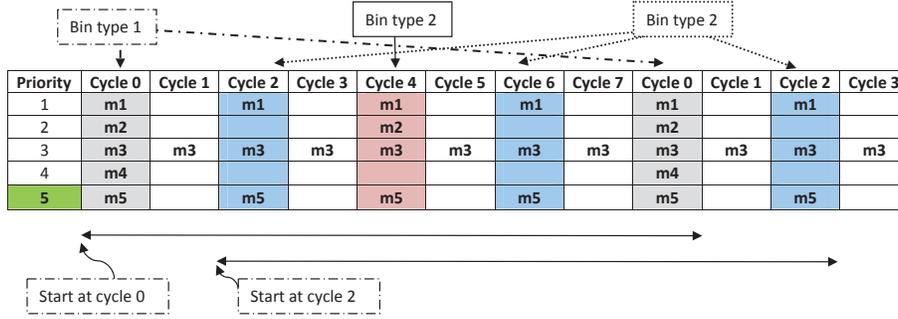


Fig. 5. The cycles where messages are allowed to be transmitted.

The previous values correspond to the case when the worst case delay of message  $m_5$  is assumed to start with cycle 0. Consider starting point as cycle 2. For this case, to check if 9 cycles can be filled, the number bins of each type that must be filled, now changes. Thus, in this case, we will have  $h_{l,1} = 1, h_{l,2} = 3$  and  $h_{l,3} = 1$ .

We proposed an algorithm to solve the problem of bin covering with conflicts. We refer the interested reader to [16] for the details. Our algorithm, 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 [9].

## VI. QUANTITATIVE COMPARISONS

### A. Quality of results

We provide a brief description of the quality of results for the three approaches that were discussed in the previous section.

First, we note that the results reported by Zeng et al. [18] that compared their heuristic with the one proposed by Pop et al [11], [12]. The response time computed by Pop et al. [11], [12] were reported to be about 8 times larger than the optimal value. The optimal value was computed by an ILP implementation. As a comparison, the heuristic by Zeng et al.[18] had an average of 0.67% error with a maximum of 15% error on the same case study.

We now discuss results comparing the quality of our heuristic [16] with Zeng et al [18]. For comparing the quality of the results we chose  $\epsilon = 1/16$  for our algorithm. The rationale behind this is that for this value of  $\epsilon$ , our algorithm can run within a matter of few minutes and is scalable. We provide details on the running times in the next section.

Since the computation of the  $busCycles_i$  is the most important component in the timing analysis of the DYN segment for our technique and the one by Zeng et al. [18], we compare  $busCycles_i$  for both techniques. Note that for comparison with previous work we assume no slot multiplexing for these experiments. We report the worst-case delays reported by both the frameworks for the lowest priority message in a message set of size 30. 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

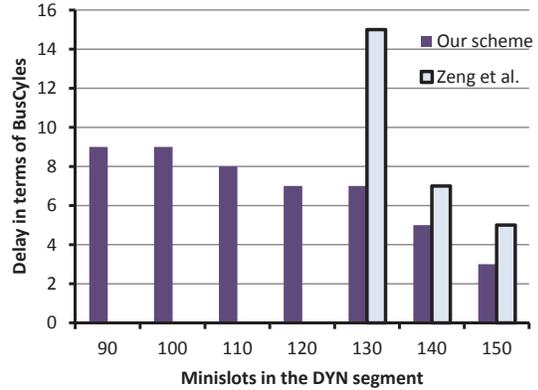


Fig. 6. Comparing the quality of results between our approach [16] and [18]. For minislot 90, 100, 110, and 120 the delay reported by [18] was infinity and is not plotted.

DYN segment, both methods report lesser worst case delay. In particular, the existing method [18] 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.

The test cases have been randomly generated by varying the message parameters like the periods and lengths, in order to cover a wide range of possible scenarios. In all experiments we have assumed that the deadlines are equal to 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. We have assumed that the length of one minislot is equal to 12  $\mu s$ .

### B. Running times

Our algorithm [16] takes  $\epsilon$  as an input from the system designer. Different values of  $\epsilon$  would lead to different running times. We ran the experiments with the values of  $\epsilon$  as  $1/32, 1/16, 1/8, 1/4$  and  $1/2$ . The results show how that the running times decrease progressively for higher values of  $\epsilon$ . These running times are plotted in Figure 7.

Note that, for the value of  $1/32$  for  $\epsilon$ , our technique will yield even better results than the ones we presented in the previous section (with  $\epsilon = 1/16$ ), in terms of the quality of

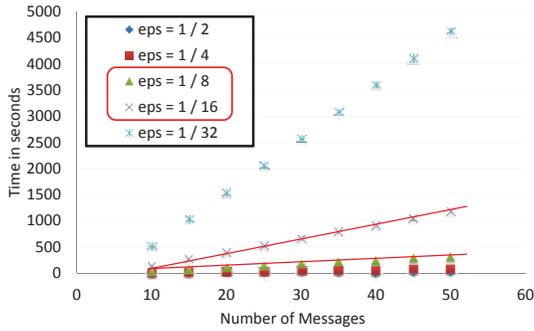


Fig. 7. The running times of our proposed algorithm for five different values of  $\epsilon$ .

the results. However, as seen from Figure 7, our algorithm does not scale well with  $\epsilon = 1/32$ . On the other hand, from our experiments we know that when  $\epsilon$  is set to  $1/2$ , or  $1/4$ , our results are, in general, pessimistic compared to the known heuristic [18]. Hence, we believe  $1/2$ ,  $1/4$ , and,  $1/32$  are not good values for  $\epsilon$ .

For a value of  $\epsilon$  set to  $1/8$ , our results are very comparable to those reported by prior work [18]. In the previous section, we already discussed that with  $\epsilon$  set to  $1/16$ , our algorithm outperforms the existing approaches from perspective of the quality of the results. Hence, from our experiments, we believe that an  $\epsilon$  value of  $1/16$  or  $1/8$  strikes the right balance between efficiency and quality.

We conclude by stating that 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. In short, our proposed scheme provides a *knob* in the form of  $\epsilon$  to the designer that allows him/her to tune the running times and the quality of solutions.

## VII. CONCLUSION AND FUTURE WORK

We conclude this paper with a short discussion on some open issues. In this paper, we have focused on the timing analysis for the DYN segment. We note that Schneider et al. [14] have focused on synthesizing message schedules, instead of the timing analysis problem. However, they used a simplistic analysis model within the synthesis framework. In future, it will be interesting to integrate our framework into such a synthesis scheme.

The timing analysis problem discussed here dealt with the *worst-case* response times. As such, our results are useful for hard real-time systems. Note that FlexRay consists of a ST segment as well. If the ST segment is used to accommodate messages from hard real-time applications, the DYN segment may be deployed for transmitting messages belonging to soft real-time applications. For such messages, the worst-case response time is not a critical performance metric. Instead,

it will be interesting to have a probabilistic analysis of the response times for the messages on the DYN segment.

It will also be worthwhile to develop a fault-tolerant message scheduling scheme on the DYN segment of the FlexRay. Fault-tolerance issues for FlexRay are a significant concern in the context of safety-critical applications that are being deployed on the cars. Soft errors induced by electro-magnetic interferences may corrupt the messages being transmitted over the FlexRay bus. Such errors can be handled by re-transmission of messages but this makes the problem of timing analysis of the DYN segment even more difficult.

## REFERENCES

- [1] L. Lo Bello. The case for ethernet in automotive communications. *SIGBED Review - Special Issue on the 10th International Workshop on Real-time Networks*, 8(4):7–15, 2011.
- [2] J.-Y. Le Boudec, P. Thiran, and F. Worm. Network calculus applied to optimal smoothing. In *INFOCOM*, 2001.
- [3] CAN Specification, Ver 2.0, Robert Bosch GmbH. [www.semiconductors.bosch.de/pdf/can2spec.pdf](http://www.semiconductors.bosch.de/pdf/can2spec.pdf), 1991.
- [4] S. Chakraborty, S. Künzli, and L. Thiele. A general framework for analysing system properties in platform-based embedded system designs. In *DATE*, 2003.
- [5] D. B. Chokshi and P. Bhaduri. Performance analysis of FlexRay-based systems using real-time calculus, revisited. In *Symposium on Applied Computing*, 2010.
- [6] The FlexRay Communications System Specifications, Ver. 2.1. [www.flexray.com](http://www.flexray.com).
- [7] E. Fuchs. FlexRay beyond the consortium phase. In *FlexRay, Special Edition Hanser Automotive*, 2010.
- [8] 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.
- [9] K. Jansen and R. Solis-Oba. An asymptotic fully polynomial time approximation scheme for bin covering. *Theor. Comput. Sci.*, 306(1-3), 2003.
- [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. In *Euromicro Conference on Real-Time Systems*, 2006.
- [12] 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.
- [13] K. Schmidt and E. G. Schmidt. Schedulability analysis and message schedule computation for the dynamic segment of FlexRay. In *Vehicular Technology Conference*, 2010.
- [14] 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.
- [15] 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.
- [16] B. Tanasa, U. D. Bordoloi, S. Kosuch, P. Eles, and Z. Peng. Interactive schedulability analysis. In *Real Time Technology and Applications Symposium*, 2012.
- [17] K. Tindell, A. Burns, and A. Wellings. Calculating Controller Area Network (CAN) message response times. *Control Engineering Practice*, 3(8):1163–1169, 1995.
- [18] 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.