2013 25th Euromicro Conference on Real-Time Systems

Probabilistic Timing Analysis for the Dynamic Segment of FlexRay

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

***Abstract*—We propose an analytical framework for probabilistic timing analysis of the event-triggered Dynamic segment of the FlexRay communication protocol. Specifically, our framework computes the Deadline Miss Ratios of each message. The core problem is formulated as a Mixed Integer Linear Program (MILP). Given the intractability of the problem, we also propose several techniques that help to mitigate the running times of our tool. This includes the re-engineering of the problem to run it on GPUs as well as re-formulating the MILP itself.**

I. Introduction

Modern automobiles are, nowadays, equipped with an embedded computing system that consists of several processing units interconnected by a fieldbus like FlexRay. In the near future, it is expected that applications from increasingly diverse domains will empower such automobiles. The diverse range of functional domains would include x-by-wire safety-critical applications (like engine control and control of car stability), driver assistance applications (like night vision service and lane departure warning), telematics (like navigation systems and traffic information), entertainment applications (like multimedia programs, personal connectivity and rear seat entertainment), and so on. In spite of their varied characteristics, to reduce design costs, it is important to map such applications into a shared hardware resource.

This is challenging because each functional domain features different requirements and specific constraints on its respective communication data. For example, x-by-wire applications impose hard real-time deadlines while other applications, such as control applications related to body control, must meet soft deadlines, specified by constraints like deadline miss ratios.

Hybrid protocols like FlexRay are well-poised to serve as a communication backbone to such systems with mixed constraints. FlexRay 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. Given the deterministic and predictable nature of the ST segment, it is considered very suitable for hard real-time systems. In fact, several tools have been introduced to schedule and map messages, characterized by hard deadlines, to the ST segment and the ST segment has been used in cars already deployed on the road. On the other hand, the DYN segment is, potentially, an attractive choice as a communication platform for soft-real time applications.

Our Contributions and Related Work: In spite of the above, however, there has been no *systematic* study on probabilistic timing analysis of the DYN segment. On the contrary, the related literature [11], [8], [6] has heavily focused on computing the worst-case response time of messages transmitted on the DYN segment which is relevant only for hard deadlines. There is only one exception [4], which however, suffers from several serious drawbacks. First, it makes the assumption that the probability of a message to be transmitted on the DYN segment is associated with the load on the buffer which is at odds with the real behavior of the system. Second, unlike our paper, it does not associate any stochastic behavior to the message triggering patterns like jitter. Third, the experiments were conducted with exactly the same probability for all the messages which severely restricts the conclusions that may be drawn. Finally, they only compute the probability that a message would be delayed beyond a single cycle and this provides no information regarding the overall delay or the relation of the delays with the deadlines. In this paper, we attempt to bridge this gap in the literature regarding the probabilistic analysis of the DYN segment. We hope that our results on the probabilistic timing analysis of the DYN segment can serve as a stepping stone to build frameworks that schedule and map messages from diverse applications, with both hard and soft constraints, in a systematic manner.

The goal of our framework is to compute the Deadline Miss Ratio (DMR) of each message that is transmitted over the DYN segment. We assume that messages are triggered by their parent tasks that generate messages in a periodic fashion. However, various factors such as variable execution times of the task or the interference from the other higher priority tasks may lead to randomness in the queuing jitter of a message. Given such a random jitter as an input, our framework, first, builds a transition graph where each vertex of the graph represents a unique backlog (see Section V). Each transition is associated with the probabilistic characteristics needed to compute the DMR. We rely on a Mixed Integer Linear Program (MILP) formulation to build this graph. Once the graph has been built, we compute the DMR using a GPU-based engine.

In the real-time system community, there has been a lot of work devoted towards probabilistic analysis. Our work is in line with such efforts [7], [2], [5]. However, most of them focused on a task set running on computational cores as opposed to our work that considers a communication system. In fact, they considered scheduling policies like fixed-priorities or EDF and hence, such existing frameworks may not be trivially extended to a probabilistic analysis of the DYN segment. It should be noted here that Zeng et al. [10] recently proposed a framework for stochastic analysis of the CAN bus protocol. However, the

1068-3070/13 $26.00 © 2013 IEEE
DOI 10.1109/ECRTS.2013.24

CAN bus protocol is based on a fixed-priority scheduling policy which has major differences compared to the protocol of the DYN segment of FlexRay.

Given the fact that even the worst-case response time analysis of the DYN segment is an NP-hard problem and that probabilistic analysis of relatively simpler schedulers, such as fixed-priority or EDF, are already intractable, the problem of probabilistic analysis of the DYN segment of FlexRay cannot be expected to be solved efficiently. As such, we propose two novel techniques to tackle the scalability issues — (i) re-engineering the problem to run it on GPUs as well as (ii) re-formulation of the MILP itself. Note that the use of GPUs, for solving schedulability analysis problems [1], [3], has gained traction in recent years.

II. FlexRay Communication

Communication in FlexRay is organized as a periodic sequence of communication cycles with a fixed length, l_{fc}. As discussed in Section I, each communication cycle is further subdivided into a ST segment and a DYN segment. It also consists of two segments : a network idle time and a symbol window that are used for administrative purposes but not for communication. In this paper, we denote them together as a time interval called IDLE.

DYN Segment: The DYN segment is partitioned into equal-length slots which are referred to as "minislots". A minislot counter tracks the current minislot of the DYN segment. An instance of a message is given access to the DYN segment when the minislot counter has the same value as the frame identifier of the message. Note that the instance of the message occupies the required number of minislots on the bus according to its size. Thus, at the beginning of each DYN segment, first, the highest priority message gets access to the DYN segment. 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 allowed to be transmitted in each FlexRay cycle.

Notations: The length of the ST segment is denoted by l_{st}. The length of one minislot in the DYN segment is denoted with l_{ms} and the total number of minislots is denoted with n_{ms}. The length of the DYN segment is thus $l_{dyn} = l_{ms} \times n_{ms}$. The length of the IDLE interval is assumed to be l_{idle}. Hence, the length of the FlexRay cycle (denoted by FC) is $l_{fc} = l_{st} + l_{dyn} + l_{idle}$.

III. System Model

As mentioned in Section I, we consider that messages are triggered by periodic tasks that are running on ECUs. We assume that messages are triggered at the point of task completion and hence, the messages inherit the periods of their parent tasks on the ECUs . However, due to the variable execution times of the tasks combined with the interferences due to other higher priority tasks, the messages may suffer from varying queuing jitter which leads to the stochastic nature of the message response times.

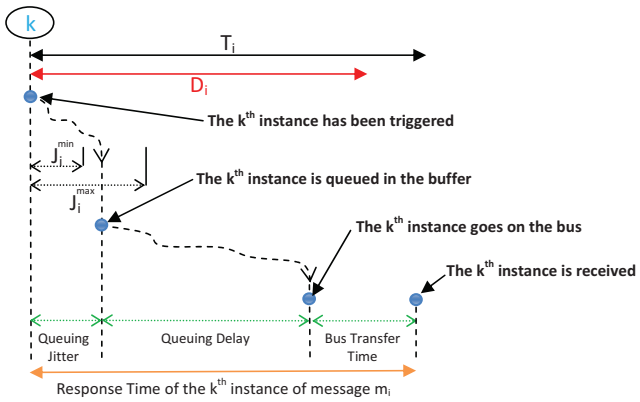

Fig. 1. The k^{th} instance of message m_i, from its triggering moment until the completion of its transmission.

Definition 1: The *queuing jitter* of an instance of a given message m_i is the time interval since the given instance has been triggered by an event until the moment it has been queued in the buffer and starts competing for bus access. □

To encapsulate queuing jitter as well as other characteristics of the messages, we define each message m_i as a n-tuple of 8 parameters. We assume that a set Γ of N periodic messages are designated to be transmitted on the DYN segment of FlexRay. Thus $\Gamma = \left\{m_i | m_i = \left(O_i, T_i, D_i, W_i, F_i, J_i^{min}, J_i^{max}, f_i\right), i = \overline{1, N}\right\}$.

1) The **offset -** O_i is the relative time with respect to the starting point of the FlexRay communication when the first instance of a message m_i will be triggered.
2) The **period -** T_i is the rate at which all the instances of a message m_i are being triggered. After the offset O_i elapses the triggering events of the instances of the message m_i are assumed to be periodic with period T_i.
3) The **deadline -** D_i is the relative time since the triggering event of a given instance of a message m_i until the time by which the transmission of it must end.
4) The **length -** W_i is the number of minislots that each instance of a message m_i will occupy when transmitted on the DYN segment of FlexRay.
5) The **priority -** F_i is the minislot identifier used by the FlexRay communication controller to transmit all the instances of message a m_i on the DYN segment.
6) The **minimum jitter -** J_i^{min} is the minimum queuing jitter that each instance of a message m_i can have.
7) The **maximum jitter -** J_i^{max} is the maximum queuing jitter that each instance of a message m_i can have.
8) The **probability density function -** $f_i : [J_i^{min}..J_i^{max}] \rightarrow \mathcal{R}_+$ of the jitter of a message m_i describes the relative likelihood for this jitter to take on a given value in the interval $[J_i^{min}..J_i^{max}]$. We do not impose any restriction with regards to the nature of this function.

The instances of a message are triggered periodically but due to the queuing jitter, an instance competes for the bus only after suffering an additional delay. Thereafter, an instance might be further delayed by instances of higher priority messages as well as prior instances of the same message that are in the queue.

This time interval — since the queuing of the instance in the transmission buffer until the moment when the given instance is sent on the bus — may be referred to as the queuing delay of an instance of a message m_i. Once the instance gains access to the bus, the time required by W_i minislots occupied by an instance of a message may be called the bus transfer time. The above is also illustrated in Figure 1 for the k^{th} instance of a message m_i.

Given the above, the response time of an instance of a message m_i is the time interval since the triggering moment of the given instance until the moment when the instance completes its transmission. An instance of a message m_i is said to suffer a deadline violation if its response time is greater than its deadline D_i.

IV. Problem Formulation

We now state the problem of probabilistic timing analysis of the DYN segment as the problem of computing the *Deadline Miss Ratio* (DMR) of a given message m_i when the system identified by the set of messages Γ is analyzed for an infinite amount of time. Formally, for a finite interval of time Δt, the $\mathcal{DMR}^i_{\Delta t}$ of a message m_i represents the expected value of the discrete random variable $\mathcal{Z}^i_{\Delta t}$. The discrete random variable $\mathcal{Z}^i_{\Delta t}$

$$\mathcal{Z}^i_{\Delta t} = \begin{pmatrix} \frac{0}{s_i} & \frac{1}{s_i} & \cdots & \frac{k}{s_i} & \cdots & \frac{s_i}{s_i} \\ \xi_0 & \xi_1 & \cdots & \xi_k & \cdots & \xi_{s_i} \end{pmatrix} \tag{1}$$

gives the probability ξ_k that exactly k instances of message m_i, out of the total number of instances s_i sent during the time interval Δt , may have a response time greater than the deadline D_i. Thus,

$$\mathcal{DMR}^i_{\Delta t} = E\left[\mathcal{Z}^i_{\Delta t}\right] = \sum_{k=0}^{s_i} \frac{k}{s_i} \times \xi_k \tag{2}$$

Definition 2: For an infinite time interval, the $\mathcal{DMR}^i$ of a message m_i is defined as:

$$\mathcal{DMR}^i = \lim_{\Delta t \to \infty} \mathcal{DMR}^i_{\Delta t} \quad \square \tag{3}$$

V. Overview of our Solution

Our method starts by computing the possible backlog vectors (which represents the accumulated messages in the buffers) at the end of a finite time interval Δt (the quantum of which, we will discuss later). As mentioned in Section I, we construct a transition graph with one vertex for each unique backlog vector. Our technique, then, computes the probability that for a message m_i, exactly k instances violate the deadline D_i at each transition. Once this is computed, we apply the process of convolution in order to find the DMR as defined in Equation 3. The above process consists of three major steps as described in Section V-A, V-B, and V-C. Towards this, we introduce a few definitions in the following.

Definition 3: A message m_i is said to have an input backlog of u_i instances at the start of the time interval Δt, if u_i instances of message m_i have been triggered during the FlexRay cycles before Δt and have not been transmitted until the start of the time interval Δt. $\square$

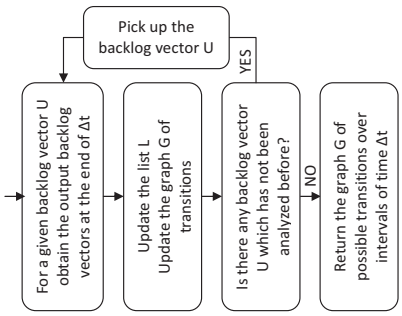

Fig. 2. The algorithmic flow of Step I of our proposed framework.

Note that the u_i instances of message m_i referred in Definition 3 are a subset of all the instances triggered during all the FlexRay cycles before the start of the time interval Δt.

Definition 4: A message m_i is said to have an output backlog of v_i instances at the end of the time interval Δt if there exists an accumulation of v_i instances of message m_i triggered (i) during the time interval Δt and (ii) during the cycles before Δt but still waiting to be transmitted during the FlexRay cycles after the completion of Δt. $\square$

We would like to emphasize that the above definition of backlog also includes an instance that has been triggered during Δt but has **not** been queued in the transmission buffer because of its own queuing jitter within Δt. Thus, physically, such an instance is **not** in the buffer but since its periodic event of triggering has elapsed within Δt, it contributes to the output backlog according to the above definition. Of course, an instance that has been triggered and has been queued in the transmission buffer during Δt but is delayed, also contributes to the output backlog defined above.

Given the above definitions for backlogs for each message, we can now extend it to consider the backlog of all messages as encapsulated by a vector.

Definition 5: The vector $\mathcal{U} = (u_1, u_2, \cdots, u_N)$ is defined as the input backlog vector at the start of the time interval Δt, where u_i has been previously introduced as the input backlog of message m_i at the start of Δt. $\square$

Definition 6: The vector $\mathcal{V} = (v_1, v_2, \cdots, v_N)$ is defined as the output backlog vector at the end of the time interval Δt, where v_i has been previously introduced as the output backlog of message m_i at the end of Δt. $\square$

A. Step I - Generating the transition graph

In Step I, our framework builds a transition graph $\mathcal{G}$ where each vertex represents a unique backlog vector and each edge is a transition as defined below.

Definition 7: A transition $\mathcal{U} \overset{\Delta t}{\Rightarrow} \mathcal{V}$ over a finite time interval Δt is defined as the sequence of events connecting the backlog vectors $\mathcal{U}$ and $\mathcal{V}$ at the boundaries of Δt. $\square$

Our algorithm is sketched in Figure 2. First, the transition graph $\mathcal{G}$ is initialized with a vertex representing the input backlog vector $\mathcal{U} = (0, 0, \cdots, 0)$ (corresponding to the start of the FlexRay communication). Then, our algorithm proceeds, iteratively, in the following fashion. Given an input backlog vector $\mathcal{U}$ at the start of a time interval Δt, it computes the list $\mathcal{L}$ of all possible output backlog vectors which can be

TABLE I
THE PARAMETERS OF ALL MESSAGES IN THE RUNNING EXAMPLE.

	$\frac{O}{ms}$	$\frac{T}{ms}$	$\frac{D}{ms}$	W	F	$\frac{J^{min}}{\mu s}$	$\frac{J^{max}}{\mu s}$	a
m_1	.10	4.5	4.5	12	1	45	900	472.5
m_2	.25	3.0	3.0	10	2	30	600	315.0
m_3	.18	3.0	3.0	9	3	30	600	315.0
m_4	.40	4.0	4.0	5	4	40	800	420.0
m_5	.28	4.5	4.5	4	5	45	900	472.5

reached at the end of Δt. If any backlog vector $\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{V}_l$ from the list $\mathcal{L}$ is not present in the graph $\mathcal{G}$, then a new vertex is added corresponding to that backlog vector. We also update the graph with all the outgoing transitions from the vertex representing backlog $\mathcal{U}$. The algorithm terminates when there are no new backlog vectors in the list $\mathcal{L}$ otherwise it repeats the above phases by setting one of the new vectors as the $\mathcal{U}$ for the next iteration. We would like to note that our algorithm has been designed to terminate even if the size of the transition graph $\mathcal{G}$ is infinite. It can be easily shown that the transition graph $\mathcal{G}$ is finite as long as there is no infinite accumulation in any of the backlog vectors. Hence, our algorithm checks if the transition graph $\mathcal{G}$ is infinite by imposing some simple conditions on the nature of backlog accumulated in the transitions. Due to space limitations, we omit the details here.

Choice of the time interval Δt: We choose Δt as the hyper-period of all periods and length of the FlexRay cycle: $\Delta t = \text{lcm}\{l_{fc}, T_1, T_2, \cdots, T_n\}$. It is a natural choice of Δt because the triggering patterns of all the messages repeat in the same fashion with respect to the FlexRay cycles after a hyper-period.

Note that, for a given input backlog vector $\mathcal{U}$, several output backlog vectors are possible at the end of the time interval Δt and we denote that with the list $\mathcal{L} = \{\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{V}_l\}$. This is due to the random behavior of the queuing jitter of all $u_i + p_i$ (defined in the following) instances of a message m_i which have to be considered during the time interval Δt. Here u_i is the input backlog for message m_i at the start of Δt while p_i is the number of triggered instances during Δt. The total number of instances of a message m_i which will be triggered during an arbitrarily hyper-period Δt is $p_i = \left\lceil \frac{\Delta t - O_i}{T_i} \right\rceil$.

Each input backlog vector needs to be analyzed separately to compute the output backlog vectors that might result from it. This, however, is a complex function of the FlexRay parameters and the parameters of the messages. To compute this list, we formulate a MILP model (see Section VI).

Running Example: In order to better illustrate our framework, we use a running example that is shown in Table I. In this example, the set Γ contains 5 messages. The length of the ST segment is $l_{st} = 1200\mu s$, the number of minislots inside the DYN segment is $n_{ms} = 24$, the length of one minislot is $l_{ms} = 10\mu s$ and the length of the IDLE interval is $l_{idle} = 160\mu s$. This gives us the length of FlexRay cycle $l_{fc} = 1600\mu s$ and thus the length of the time interval $\Delta t = 45 \times l_{fc}\ \mu s$ (because Δt is chosen to be equal to the hyper-period). We assume that the likelihood of the jitter to occur in the $[J_i^{min}..J_i^{max}]$ is defined by a truncated Weibull distribution. The parameter a of Weibull distribution is shown in the last column in the table and b is set to 4 for all messages. We re-iterate that our framework is not restricted to any particular distribution and Weibull has been chosen only as an illustration.

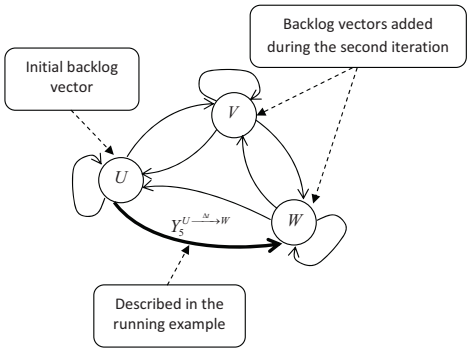

Fig. 3. The complete graph $\mathcal{G}$ of transitions over intervals of time Δt for the example presented in Table I.

We describe Step I (Figure 2) with respect to the above example. The algorithm starts by initializing $\mathcal{U} = (0, 0, 0, 0, 0)$. Here, $\mathcal{U}$ is the backlog vector at the start of the system and contains the backlog (which is 0 when the system starts) for each of the 5 messages. This initial backlog vector is added as the first vertex $\mathcal{U}$ in the transition graph (Figure 3). Then the algorithm proceeds as shown in Figure 2, where we invoke a MILP solver with $\mathcal{U}$ as the input. The MILP solver now updates the list $\mathcal{L}$ with two new vertexes $\mathcal{V} = (0, 0, 0, 1, 0)$ and $\mathcal{W} = (0, 0, 0, 1, 1)$. These backlog vectors show that the messages m_4 and m_5 accumulate a backlog of 1 after two different transitions. Our algorithm now adds two new vertices in the transition graph (Figure 3). We also add the transition from vertex $\mathcal{U}$ to itself and the transition from vertex $\mathcal{U}$ to vertices $\mathcal{V}$ and $\mathcal{W}$.

Thereafter, our algorithm iterates again because $\mathcal{V}$ and $\mathcal{W}$ are new vertices in the list $\mathcal{L}$. Thus, the MILP solver is invoked again, once with $\mathcal{V}$ and once with $\mathcal{W}$ as the input. However, this time the MILP solver does not return any new backlog vectors and hence, the algorithm does not iterate any further. However, it updates the graph with new transitions and this results in the final graph that is shown in Figure 3. The probability for these transitions to occur, depending on the input backlog vector, is computed by Step II of our framework.

B. Step II - Computing the probabilities at transitions

Knowing the set of all possible transitions $\mathcal{U} \overset{\Delta t}{\rightarrow} \mathcal{V}$, in Step II, our method will compute for a message m_i, based on its own set of higher priority messages, the probability ξ_k that exactly k instances suffer a deadline violation out of the total number of instances s_i sent during the current transition. For a given transition $\mathcal{U} \overset{\Delta t}{\rightarrow} \mathcal{V}$, when the message under analysis is m_i, we will attach to it a discrete random variable $Y_i^{\mathcal{U} \overset{\Delta t}{\rightarrow} \mathcal{V}}$ with no more than $(s_i + 1)$ realizations. Thus, in Step II, we will compute the probability ξ_k that exactly k deadline violations occur during Δt when k takes values in the set $\{0, 1, \cdots, s_i\}$. This probability is obtained by analyzing in how many ways exactly k instances out of s_i instances can violate their deadline

TABLE II
THE PROBABILITIES THAT EXACTLY k INSTANCES OF MESSAGE m_5 VIOLATED THE DEADLINE D_5.

k	ξ_k	k	ξ_k
0	0	4	7.739474E-04
1	2.371244E-03	**5**	**1.778538E-04**
2	1.216082E-03	6	1.805873E-08
3	1.406886E-04	7	5.622012E-05

D_i (see Section VII for details on how this is done). Thus, as an output of Step II we obtain the following.

$$Y_i^{\mathcal{U}\overset{\Delta t}{\rightarrow}\mathcal{V}} = \begin{pmatrix} 0/s_i & 1/s_i & \cdots & s_i/s_i \\ \xi_0 & \xi_1 & \cdots & \xi_{s_i} \end{pmatrix} \tag{4}$$

At the end of Step II, for each message m_i, we will have assigned to all possible transitions $\mathcal{U} \overset{\Delta t}{\rightarrow} \mathcal{V}$ a discrete random variable $Y_i^{\mathcal{U}\overset{\Delta t}{\rightarrow}\mathcal{V}}$. It would be appropriate to mention here that the number of instances s_i of a message m_i which have been sent during the time interval Δt is computed in Step I and is discussed in Section VI.

Note: We have mentioned above that during Step I, for a possible transition $\mathcal{U} \overset{\Delta t}{\rightarrow} \mathcal{V}$, $u_i + p_i$ instances of a message m_i are involved. This is in contrast with the fact that for the same transition, during Step II, only $s_i \leq u_i + p_i$ instances have to be analyzed. The remaining v_i instances (see Equation 10) will be analyzed in the context of the next transition which takes the output backlog vector $\mathcal{V}$ as an input backlog vector.

Running Example: For the example presented in Table I, considering that the message under analysis is m_5, we show the discrete random variable $Y_5^{\mathcal{U}\overset{\Delta t}{\rightarrow}\mathcal{W}}$ in Table II. The number of instances of message m_5 triggered during Δt in the context of the transition $\mathcal{U} \overset{\Delta t}{\rightarrow} \mathcal{W}$ is $p_5 = 16$ as given by $p_i = \left\lceil \frac{\Delta t - O_i}{T_i} \right\rceil$, where Δt is the length of the hyper-period. Out of them only 15 are sent on the FlexRay bus (as computed during Step II using Equation 10) and thus k varies from 0 to 15. The remaining ones will be sent during the next time interval Δt. In Table II, it can be seen, for example, that the probability that exactly 5 instances of message m_5, out of the total which have been sent during Δt, may suffer a deadline violation is 1.778538E-04. Note that for each k from 8 to 15 the associated probability value is 0 and hence, it is not shown in the above table.

C. Step III - Computing the DMR

Step III of our solution is to compute the DMR as defined in Equation 3. Note that the exact value of the $\mathcal{DMR}^i$ of a message m_i is obtained when the process of convoluting the discrete random variables (defined in Equation 4) is let to run for an infinite amount of time. However, in practice, it is typically approximated by terminating the convolutions after a desired level of convergence. Hence, we proceed as in the following.

The starting point of the convolution process is the vertex with the backlog vector $\mathcal{U} = (0, 0, \cdots, 0)$. The process of approximating the $\mathcal{DMR}^i$ of message m_i is iterative. First the $\mathcal{DMR}^i$ of message m_i will be approximated for one time interval Δt, i.e., one transition starting from vertex $\mathcal{U}$. Then, the process is repeated for two consecutive intervals of time Δt (i.e., two consecutive transitions starting from vertex $\mathcal{U}$) and, based on the designers preference of accuracy, it iterates for three, four and possibly for more consecutive intervals of time Δt until the desired level of convergence is achieved.

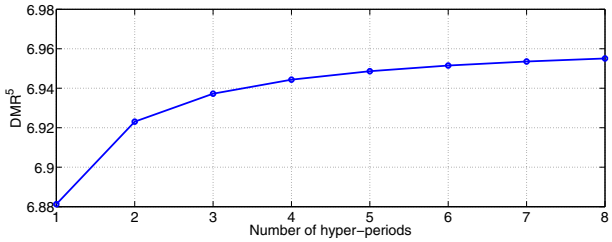

Fig. 4. The convolution process converges rapidly while computing the $\mathcal{DMR}^5$ value of the message m_5.

We would like to remind that, the convolution process first provides as an output the discrete random variable $\mathcal{Z}^i_{n\times\Delta t}$ (see Section IV). Here n signifies the number of consecutive hyper-periods Δt. Based on it, the $\mathcal{DMR}^i$ of a message m_i is approximated by computing the expected value $E\left[\mathcal{Z}^i_{n\times\Delta t}\right]$.

Running Example: For the example presented in Table I the convolution process is described below. The graph of transitions returned by Step I is a complete graph with 3 vertices (see Figure 3). For one time interval Δt ($n = 1$) the $\mathcal{DMR}^i$ of message m_i is approximated by analyzing all possible transitions ($\mathcal{U} \overset{\Delta t}{\rightarrow} \mathcal{U}$,$\mathcal{U} \overset{\Delta t}{\rightarrow} \mathcal{V}$ and $\mathcal{U} \overset{\Delta t}{\rightarrow} \mathcal{W}$) starting from the backlog vector $\mathcal{U} = (0, 0, 0, 0, 0)$. By collapsing the discrete random variables $Y_i^{\mathcal{U}\overset{\Delta t}{\rightarrow}\mathcal{U}}$, $Y_i^{\mathcal{U}\overset{\Delta t}{\rightarrow}\mathcal{V}}$ and $Y_i^{\mathcal{U}\overset{\Delta t}{\rightarrow}\mathcal{W}}$ (that were computed in Step II) into one we get the discrete random variable $\mathcal{Z}^i_{1\times\Delta t}$.

For two consecutive intervals of time Δt the approximation of the $\mathcal{DMR}^i$ of message m_i is obtained by analyzing all possible transitions $\mathcal{U} \overset{\Delta t}{\rightarrow} \mathcal{U} \overset{\Delta t}{\rightarrow} \mathcal{U}$,$\mathcal{U} \overset{\Delta t}{\rightarrow} \mathcal{U} \overset{\Delta t}{\rightarrow} \mathcal{V}, \cdots, \mathcal{U} \overset{\Delta t}{\rightarrow} \mathcal{W} \overset{\Delta t}{\rightarrow} \mathcal{W}$. A transition of the form $\mathcal{U} \overset{\Delta t}{\rightarrow} \mathcal{V} \overset{\Delta t}{\rightarrow} \mathcal{W}$ will have associated a discrete random variable $Y_i^{\mathcal{U}\overset{\Delta t}{\rightarrow}\mathcal{V}\overset{\Delta t}{\rightarrow}\mathcal{W}}$ obtained as the convolution of the discrete random variables $Y_i^{\mathcal{U}\overset{\Delta t}{\rightarrow}\mathcal{V}}$ and $Y_i^{\mathcal{V}\overset{\Delta t}{\rightarrow}\mathcal{W}}$. Similarly the discrete random variable $\mathcal{Z}^i_{2\times\Delta t}$ is obtained by collapsing together all the associated random variables to the possible transitions for two consecutive intervals of time Δt. In Figure 4, we show how the $\mathcal{DMR}^5$ of message m_5 converges with increasing number of iterations. It is important to note that, even if this is a potentially computationally expensive step due the possibly large number of paths to be explored, in practice the convergence is achieved very fast. In fact, as our results will later demonstrate, Step III incurs the least running time amongst the three steps in our framework.

Recall that computing the DMR was the goal of our framework and with the above example from Step III, where we discussed how we obtained $\mathcal{DMR}^5$, we may now conclude the overview of our overall framework. In the following, we would like to provide more details on two components of our framework. First, in Section VI, we will discuss the MILP model that is used in first phase of the algorithm (see Figure 2) in Step I. Second, in Section VII, we will discuss the details on how Step II computes the discrete random variables (Equation 4) associated with each transition.

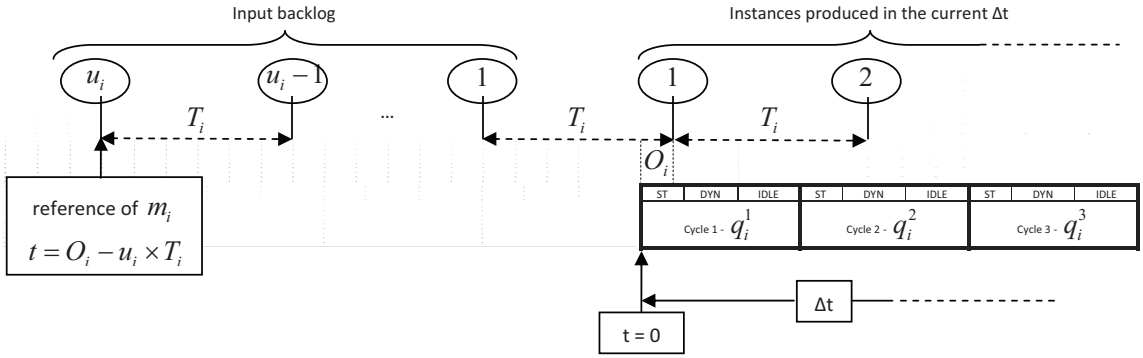

Fig. 5. Modeling the input backlog at the start of one time interval Δt.

VI. Modeling the FlexRay Communication

In what follows we will model the temporal behavior of the FlexRay communication for a finite interval of time Δt, chosen to be equal to one hyper-period, by formulating a MILP problem which takes as an input (i) the parameters of all messages, (ii) the FlexRay parameters and (iii) the input backlog vector $\mathcal{U}$. As an output, the MILP formulation provides the set of output backlog vectors reached from $\mathcal{U}$ at the end of time interval Δt.

Let us first introduce the variables required to model a particular message m_i associated with priority F_i. For each message m_i we introduce a set of Boolean variables q_i^c, $q_i^c = 1$ if an instance of this message has been transmitted in cycle c. Instances will be sent in the order they have been triggered. The total number of cycles in one hyper-period is $h = \frac{\Delta t}{l_{fc}}$.

Essentially the linear constraints of our MILP problem will model the arrival times of instances of message m_i in the transmission buffer as well as their possible transmission times. We will denote with α_i^c (see Equation 5) the queuing moment in the transmission buffer of the instance which has to be considered for transmission in cycle c.

$$\alpha_i^c = \overbrace{\underbrace{O_i - \underbrace{u_i \times T_i}_{\text{input backlog}}}_{\text{reference of } m_i} + \underbrace{\left(\sum_{j=1}^{c-1} q_i^j\right)}_{\text{sent until cycle } c} \times T_i}^{\text{periodic moments of triggering}} + X_i^c \quad (5)$$

The above equation is explained with the help of Figure 5, where we show how we model, in our MILP formulation, the input backlog of a message m_i with u_i instances triggered during the cycles before Δt which are still waiting to be transmitted. These instances are indexed by the set $\{u_i, u_i - 1, \cdots, 1\}$. Without loss of generality we can consider that each interval of time Δt starts at moment $t = 0$. Thus, if a message has an input backlog of u_i instances, the first instance that has to be considered for transmission in the first cycle of Δt is the one triggered at moment $O_i - u_i \times T_i$. This moment is denoted as the reference point of message m_i. We also show how instances will be triggered during Δt. The first instance will be triggered with a given offset O_i with respect to the starting moment of Δt. For a cycle c the instance which has to be *considered for transmission* is given by the summand $\sum_{j=1}^{c-1} q_i^j$. This value tells us how many instances of message m_i have been transmitted in the previous cycles since the beginning of cycle 1 of the current time interval Δt until the beginning of cycle c.

The variable X_i^c in the definition of α_i^c represents the queuing jitter of the instance which has to be considered for transmission in the current cycle c. It is required to attach a queuing jitter variable even to the instances which represent the input backlog because of the fact that it is possible to have instances which have been triggered during the cycles before Δt but have not arrived yet in the transmission buffer until the start of Δt (or until the start of the current hyper-period).

β_i^c (see Equation 6) represents the *possible* time, relative to starting moment of cycle c, when the instance that needs to be considered for transmission in this cycle can be transmitted. The qualifier *possible* is used to denote the following characteristic of FlexRay communication in the DYN segment: the transmission of an instance does not depend only on the fact that the given instance has to be ready in the transmission buffer before its own minislot has arrived but also there must be enough space for it inside the DYN segment. Otherwise the given instance will be delayed until the next cycle when the same arbitration process starts again. To capture this, we introduce γ_i^c (see Equation 7), which is the displacement of minislot F_i inside the DYN segment of cycle c based on the set of messages with higher priorities compared to message m_i.

$$\beta_i^c = \overbrace{(c-1) \times l_{fc}}^{c-1 \text{ cycles since } t=0} + \overbrace{l_{st} + \gamma_i^c \times l_{ms}}^{\text{inside cycle } c} \quad (6)$$

$$\gamma_i^c = F_i - 1 + \sum_{j=1}^{i-1} q_j^c \times \overbrace{(W_j - 1)}^{\text{the load of } m_j, \text{ if transmitted}} \quad (7)$$

Having the previous equations (Equations $5, 6, 7$) an instance of message m_i can be transmitted in a cycle c iff:

$$(q_i^c = 1) \Leftrightarrow \overbrace{(\alpha_i^c \leq \beta_i^c)}^{\text{part 1}} \wedge \overbrace{(\gamma_i^c + W_i \leq n_{ms})}^{\text{part 2}} \quad (8)$$

The **first part** of Equation 8 checks if the instance of message m_i, which has to be considered for transmission, is queued

in the transmission buffer before the possible transmission time β_i^c associated with cycle c. The **second part** enforces the condition that the given instance of message m_i has to fit inside the DYN segment of cycle c. In other words, (i) the displacement of minislot F_i based on the set of messages with higher priority than message m_i plus (ii) the number of minislots which the current instance of message m_i will occupy when transmitted cannot exceed the total number of minislots n_{ms} of the DYN segment.

As discussed, the variable X_i^c of a cycle c captures the queuing jitter of an instance of message m_i. Hence, the variable X_i^c for any cycle c is constrained (by our MILP model) within the bounds $[J_i^{min}..J_i^{max}]$ provided as an input in our system model. However, X_i^c for an instance of a message m_i may be constrained within a tighter interval by the MILP constraints that we have described above. When a message is not transmitted in cycle c, these tighter constraints must remain valid in the next cycle $c+1$.Towards this, we introduce the following additional MILP constraints.

$$\begin{cases} X_i^{c+1} \leq X_i^c + q_i^c \times M \\ X_i^{c+1} \geq X_i^c - q_i^c \times M \end{cases} \tag{9}$$

, where M is large enough constant.

For example, if $q_i^c = 0$, then the instance of the message m_i has not been sent in cycle c. This implies this instance will contend for transmission in cycle $c+1$. Hence, if the X_i^c value of this instance was evaluated by the MILP to be associated with tighter constraints, then this must be propagated to X_i^{c+1}. To handle this scenario, if $q_i^c = 0$, the above inequalities will be reduced to the equality $X_i^c = X_i^{c+1}$. The opposite case for $q_i^c = 1$ is similarly handled with the fact that M is a large constant.

When invoked with the above constraints, the MILP solver returns a feasible (i.e., satisfying the above constraints) assignment of the q_i^j variables. The summation $\sum_{j=1}^{h} q_i^j$ is equal to s_i which gives us the number of instances sent in Δt. Now, using Equation 10, we can obtain v_i for each message m_i and thereby, obtain the output backlog vector $\mathcal{V}$.

$$\overbrace{v_i}^{\text{output backlog}} = \overbrace{u_i}^{\text{input backlog}} + \overbrace{p_i}^{\text{triggered}} - \overbrace{s_i}^{\text{sent}} \tag{10}$$

Next, the MILP solver iterates to produce a new $\mathcal{V}'$, if it exists. To obtain this, it is invoked with the constraint that at least for one message, the backlog $v_i \in \mathcal{V}$ from the previous iteration must be assigned a different value in $\mathcal{V}'$. These iterations continue as long as a new backlog vector is produced after which our algorithm (Figure 2) in Step I proceeds to its second phase.

Note: The MILP model is composed of the linear in-equations described in this section. We followed standard techniques to transform the logical operator AND of Equation 8 into linear constraints [9]). Let us denote the above MILP problem with MILP$_h$, where $h = \frac{\Delta t}{l_{fc}}$ represents the number of consecutive FlexRay cycles in Δt when Δt was chosen to be equal to the hyper-period. We also note that our MILP formulation does have an optimization function because in this problem we are looking for any solution that satisfies the constraints presented above.

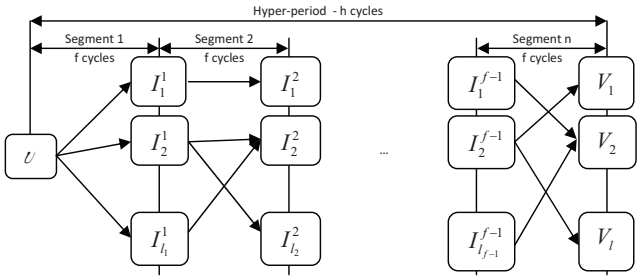

Fig. 6. The hyper-period is divided into equal lengths and a new MILP problem is formulated based on these smaller time intervals. The MILP produces a set of intermediary solutions based on which the list of solutions for the initial MILP problem is computed.

A. Towards scalability : Re-engineering the MILP$_h$ problem

The above MILP formulation is invoked repeatedly in algorithm (see Figure 2) used in Step I. Given the intractability of the problem, this might lead to a serious performance bottleneck. Hence, in the following, we propose a method to tackle the scalability problem by re-engineering the MILP$_h$ problem into a set of smaller MILP problems. By *smaller* MILP, we imply the fact that these MILP problems consider a smaller time interval Δt than the hyper-period. To obtain the smaller time interval, we divide the original time interval Δt into a number of n segments all containing the same number of cycles $f = \frac{h}{n}$. For a given segment $l, l = \overline{1,n}$, we now formulate a new but smaller MILP problem based on the same model presented in Section VI considering a shorter time interval $\Delta t = f \times l_{fc}$ chosen to represent f consecutive FlexRay cycles. We denote with MILP$_f^l$ the new smaller MILP problem.

In Figure 6 we show that starting from an input backlog vector $\mathcal{U}$ a MILP$_f^1$ problem is solved for the first segment of the hyper-period. By solving the MILP$_f^1$ problems, we mean obtaining the output vectors at the end of the first segment (i.e., at the end of f cycles). They are represented by $\{\mathcal{I}_1^1, \mathcal{I}_2^1, \cdots, \mathcal{I}_{l_1}^1\}$. Each of these output backlog vector will be analyzed separately by the corresponding MILP problem for the next segment. This process is repeated until all the intermediary solutions are processed.

Since $f \leq h$ the number of optimization variables of the MILP$_f^l$ model is reduced compared to the MILP$_h$ model. Thus the MILP$_f^l$ problem can be solved faster. The list of the possible output backlog vectors $\{\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{V}_l\}$ at the end of the hyper-period, based on the input backlog vector $\mathcal{U}$, is obtained when the last segment has been analyzed. When compared to output backlog vectors provided by solving the MILP$_h$, this set of solution is a superset and hence, leads to pessimism. This pessimism propagates to the Step II and Step III. This is discussed further in the experimental results.

VII. Computing the probabilities at transitions

Above, we discussed the MILP formulation used in Step I of our overall framework. Step II computes the probabilities of all possible realizations (for k deadline violations) of the

discrete random variable $Y_i^{\mathcal{U}\overset{\Delta t}{\rightarrow}\mathcal{V}}$ associated with the transition $\mathcal{U} \overset{\Delta t}{\rightarrow} \mathcal{V}$, when the time interval Δt is one hyper-period. In the light of the new approach to our MILP$_h$ problem, as discussed in Section VI-A, we first note that, in fact, we solve a set of smaller MILP$_f^l$ problems by dividing Δt into a number of segments. The variable $Y_i^{\mathcal{U}\overset{\Delta t}{\rightarrow}\mathcal{V}}$ is computed by convoluting the variables on the intermediary paths which lead to the output backlog vector $\mathcal{V}$. Hence, we must first compute the corresponding random variables from the intermediate transitions obtained from the smaller MILP$_f^l$ problems and this procedure is the subject of discussion in the following.

Let us consider the case of an intermediary segment l for which the input and the output backlog vectors, $\mathcal{I}_a^{l-1}$ and $\mathcal{I}_b^l$, are known by solving the corresponding MILP$_f^l$ problem. We also consider that the message under analysis is m_i. In what follows we will present the method to compute the discrete random variable $Y_i^{\mathcal{I}_a^{l-1}\overset{\Delta t}{\rightarrow}\mathcal{I}_b^l}$ associated with the transition $\mathcal{I}_a^{l-1} \overset{\Delta t}{\rightarrow} \mathcal{I}_b^l$ in the context of the segment l, when Δt is chosen to be equal to the length of the given segment. This method will be iteratively called for all the transitions $\mathcal{I}_a^{l-1} \overset{\Delta t}{\rightarrow} \mathcal{I}_b^l$.

Towards this, we formulate a new MILP model and we denote this as eMILP$_f^{l,k}$. This model is exactly the same model as presented in Section VI but with two sets of additional constraints. The first set of constraints are imposed in order to evaluate the intermediate transition $\mathcal{I}_a^{l-1} \overset{\Delta t}{\rightarrow} \mathcal{I}_b^l$ obtained after solving the MILP$_f^l$ in Section VI-A. To achieve this, we use Equation 10, instantiated with the values obtained after solving the MILP$_f^l$, as an additional constraint. This constraint is required because we are interested in computing the $Y_i^{\mathcal{I}_a^{l-1}\overset{\Delta t}{\rightarrow}\mathcal{I}_b^l}$ associated with this transition for this iteration.

The second additional set of constraints is to capture the sequence of events that can lead to a scenario where exactly k instances of message m_i, out of the total which are sent, may violate the deadline D_i during the given intermediate transition. To achieve this, we introduce a set of Boolean variables y_i^c in the eMILP$_f^{l,k}$ model, where $y_i^c = 1$ if the instance of a message m_i sent in cycle c may violate its deadline D_i. Hence, to analyze if exactly k instances can violate their deadlines we impose the condition $\sum_{c=1}^{f} y_i^c = k$.

A solution of the eMILP$_f^{l,k}$ problem is given by the integer values of the Boolean variables q_j^c ($\forall j \leq i, \forall c \leq f$) and the bounds on the continuous values of variables X_j^c ($\forall j \leq i, \forall c \leq f$) representing the queuing jitter. Using this solution, we may now compute the probability that k deadline violations for a message m_i could have occurred with the particular scenario represented by the eMILP$_f^{l,k}$ solution. This probability computation is shown in Algorithm 1. Given a particular solution found by the eMILP$_f^{l,k}$ problem, it computes the probability of that this solution (with k deadline violations) could have occurred, based on the bounds on the variable X_j^c ($\forall j \leq i$). The algorithm represents these bounds as a and b. Recall that these bounds are, initially, within J_i^{min} and J_i^{max} but the eMILP$_f^{l,k}$ model could have imposed tighter constraints. The algorithm (i) iterates over all the cycles f in the segment l and then (ii) iterates in order to consider the message m_i and its higher priority messages.

Data: A fixed assignment of the integer variables q_j^c.
Result: The probability ψ for the given sequence of events to happen.
`/* Initialize` ψ `*/`
$\psi = 1$;
for $c = 1 \rightarrow f$ **do**
  for $j = 1 \rightarrow i$ **do**
    if $q_j^c == 1$ **then**
      get the bounds a and b on the variable X_j^c ;
      `/* Update the probability` ψ `*/`
      $\psi = \psi \times \int_a^b f_j(x)dx$;
    end
  end
end

Algorithm 1: The algorithm of computing the probability of a sequence of events given by the a fix assignment of the integer variables q_j^c.

The above description was regarding the computation of the probability ψ of one particular solution of the eMILP$_f^{l,k}$ problem. In order to compute the overall probability ξ_k, we iterate this for all solutions of the eMILP$_f^{l,k}$ problem. The probability ξ_k, that exactly k instances of message m_i violate the deadline D_i, is obtained by summing all the probabilities ψ for all the assignments of the integer variables q_j^c.

However, exploring the set of all possible solutions of the eMILP$_f^{l,k}$ problem is a computational bottleneck. This is despite the fact that the MILP$_f^l$ problem, on which the eMILP$_f^{l,k}$ problem has been built upon, can be solved very fast in practice. This is because of the additional constraints and additional variables, introduced above, when compared to the MILP$_f^l$ model. More importantly, it is slower because we need to find all the feasible solutions and not just one particular solution.

A. Towards scalability: A GPU-based engine

In the light of the above, we propose to harness the computational power of the GPUs to address the scalability issue. However, taking the problem as it is, and utilizing the GPUs might deteriorate the performance. In order to fully exploit the computational power of the GPUs it is important to carefully reformulate the problem. In the following, we describe our algorithm towards this. The pseudo-code for our algorithm is listed in Algorithm 2.

As a first step, for each $Y_i^{\mathcal{I}_a^{l-1}\overset{\Delta t}{\rightarrow}\mathcal{I}_b^l}$, we invoke the MILP solver i times for each message m_1, m_2, ..., m_i. At the jth invocation, the MILP solver is asked to give us all possible assignments of the variables $\{q_j^1, q_j^2, \cdots, q_j^f\}$ corresponding to the message m_j. The multiple invocations of these eMILP$_f^{l,k}$ problems runs significantly faster compared to invoking the solver to obtain all the solutions using one eMILP$_f^{l,k}$ invocation. This is due to two reasons. First, note that $f << i$ because f is the number of cycles in each segment l which is typically very small compared to the total number of messages in a realistic system. Second, in contrast to exploring all the solutions from the eMILP$_f^{l,k}$ problems, the above formulation only explores the solutions corresponding to each message m_j individually. We emphasize that this formulation, of course, takes into account the interference by the higher priority messages.

Now that we have obtained a set of values for the variables

$\{q_j^1, q_j^2, \cdots, q_j^f\}$, we are finally, in a position to formulate the appropriate problem, denoted with pGPU$_f^{l,k}$ to be solved by the GPUs. Towards this, we introduce the definition of a column. A column is an assignment of the variables $\{q_j^1, q_j^2, \cdots, q_j^f\}$ that was found using the above step. It represents one particular instance of how the instances of a message m_j were sent during f consecutive cycles in the segment l. We then define with $\mathcal{C}^j$ the matrix of all possible columns for the message m_j. The matrix $\mathcal{C}^j$ represents all possible ways in which the instances of a message m_j may be sent during f consecutive cycles in the segment l. For each matrix $\mathcal{C}^j$ we introduce a set of Boolean variables $Z^j = \{\zeta_1^j, \zeta_2^j, \cdots \zeta_{n_j}^j\}$, where n_j is the number of columns of matrix j. Now, we introduce the constraints $\zeta_1^j + \zeta_2^j + \cdots + \zeta_{n_j}^j = 1, \forall j \leq i$ to capture the fact that only one column is chosen to form the solution of the pGPU$_f^{l,k}$ problem. This is because we want to choose one particular way in which the instances of each message may be sent in the segment l.

We obtained the column of a matrix $\mathcal{C}^j$ while ignoring the assignment of the remaining Boolean variables corresponding to the rest of the messages. However, this does not result in any inaccuracy in our framework because the pGPU$_f^{l,k}$ problem imposes the relevant constraints regarding the temporal behavior of the DYN segment (e.g., an instance of a message has to fit into the available minislots). Given the above, our framework suffers from no loss of accuracy at this step.

It is important to note that $\mathcal{C}^j$ of a message $m_j, j = \overline{1,i}$ has been obtained such that exactly k instances of the message under analysis m_i violate the deadline D_i out of the total number of instances sent during f consecutive FlexRay cycles as part of intermediary transition $\mathcal{I}_a^{l-1} \overset{\Delta t}{\rightarrow} \mathcal{I}_b^l$ for the segment l. Thus, if we find all possible solutions to the pGPU$_f^{l,k}$ problem, we would have found all possible solutions to the original eMILP$_f^{l,k}$ problem.

The maximum number of solutions the pGPU$_f^{l,k}$ problem can have is $\prod_{j=1}^{i} n_j$. To explore this vast solution space, we use GPUs and exploit the data parallelism. The GPU framework explores, in parallel, the set of all possible combinations of the columns by checking their feasibility. By feasibility, we refer to (i) the FlexRay related constraints and (ii) the constraints on the intervals of queuing jitter of all the instances of all messages. Finally, the algorithm for probability calculation (Algorithm 1) is also implemented on the GPUs. We omit a detailed discussion on our GPU algorithms due to limitations on space.

VIII. Experimental Results

We implemented our framework on the Nvidia Fermi machine with a TeslaM2050 GPU. It has 14 streaming multiprocessors which together have 448 cores running at 1147MHz. The host contains 2 Intel CPUs Xeon E5520, with 8 cores in total and each clocked at 2.27 GHz. Our framework is partially implemented in Matlab and partially in C++. In order to efficiently solve the MILP problems we used CPLEX. For the reformulated MILP problems, we used Jacket to explore the set of all possible combinations of columns on the GPUs. These tools are integrated with Matlab through MEX files.

Data: The transition graph $\mathcal{G}$ with the intermediate transitions (see Figure 6).
Result: The discrete random variables $Y_i^{\mathcal{U} \overset{\Delta t}{\rightarrow} \mathcal{V}}$ for message m_i.
for $l = 1 \rightarrow n$ **do**
  forall the *MILP$_f^l$ problems at segment l* **do**
    $\mathcal{I}_a^{l-1}$ - the input backlog vector of segment l ;
    forall the *solutions of the MILP$_f^l$ problem* **do**
      $\mathcal{I}_b^l$ - the output backlog vector of segment l ;
      get s_i - instances sent during the segment l ;
      for $k = 0 \rightarrow s_i$ **do**
        formulate the eMILP$_f^{l,k}$ problem ;
        for $j = 1 \rightarrow i$ **do**
          build the matrix of columns $\mathcal{C}^j$;
        end
        formulate the pGPU$_f^{l,k}$ problem ;
        solve the pGPU$_f^{l,k}$ problem on GPUs ;
        `/* Implemented in parallel where each thread calls Algorithm 1 */`
        get ξ_k, - the probability that exactly k instances violate the deadline D_i ;
      end
      get $Y_i^{\mathcal{I}_a^{l-1} \overset{\Delta t}{\rightarrow} \mathcal{I}_b^l}$ - the discrete random variable associated with the intermediary transition $\mathcal{I}_a^{l-1} \overset{\Delta t}{\rightarrow} \mathcal{I}_b^l$;
    end
  end
end

Algorithm 2: The algorithm described in Section VII-A.

The test cases have been generated randomly by varying the message parameters like the periods, deadlines, lengths and priorities. In all experiments we assumed that the length of ST segment is $l_{st} = 675\mu s$ and the length of one minislot $l_{ms} = 10\mu s$. The total number of minislots n_{ms} and the length of the IDLE segment l_{idle} have been varied such that the length of the FlexRay cycle l_{fc} remained constant $l_{fc} = 1500\mu s$. All generated periods represent harmonic numbers such that the length of the hyper-period Δt does not exceed 24 FlexRay cycles of length l_{fc}. As discussed in Section VI-A, we divide the time interval Δt into a number of segments. For our experiments, each hyper-period consists of 6 segments and each segment has 4 FlexRay cycles.

The intervals of queuing jitter $[J_i^{min}..J_i^{max}]$ have been generated with respect to the periods T_i of the messages. In all of our experiments J_i^{min} has been set to represent 1% of the period T_i while J_i^{max} has been varied up to a maximum value of 15% of the same period. The probability density functions assigned to the messages have been chosen from a list of well known distributions like the Uniform, Normal and Weibull distribution. In our experiments we used the truncated version of this distributions to the given interval $[J_i^{min}..J_i^{max}]$.

A. Running Times with Increasing Number of Messages

In Figure 7 we show the running times of our tool for a varying number of messages. The jitter J_i^{max} was set to 15% for all the messages in all the sets used in this experiment. The running times for each step (Step I, Step II and Step III) of our tool is depicted separately.

The blue line represents the running taken by Step I to compute the transition graph $\mathcal{G}$, the red line represents the time needed by the GPU (Step II) to compute the probability of the transitions and the green line represents the time needed for the convolutions (Step III) to approximate the $\mathcal{DMR}^i$ of a

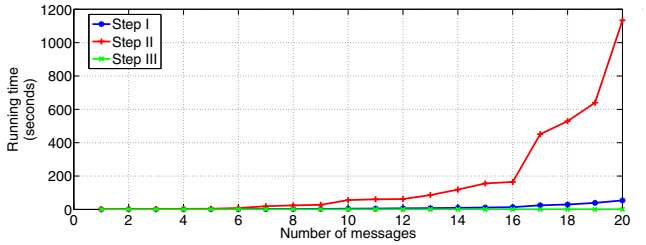

Fig. 7. The running times needed by each step.

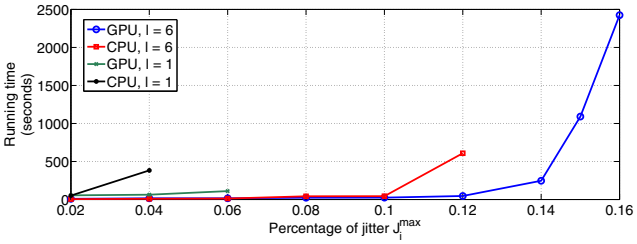

Fig. 8. The running time needed by Step II, highlighting the importance of our proposed scalability techniques.

message m_i. The message m_i was always chosen to be the lowest priority message because it leads to the highest running times. In Figure 7 it can be seen that the time needed by Step II to finish dominates over the time needed by the other two steps. The running times have been reported for our tool that includes the optimizations for scalability, i.e., division of the hyper-period into several segments (Section VI-A) and the use of GPUs (Section VII-A). We report that, in the absence of these optimizations, it was not possible to obtain the results even for a small set containing 2 messages. We provide more comparisons regarding our scalability optimizations in the following.

B. Running Times with Jitter Variation

The computation time of the $\mathcal{DMR}^i$ of a message m_i is also dictated by the length of the queuing jitter intervals $[J_j^{min}..J_j^{max}], \forall j \leq i$, of the messages with higher priority than message m_i and message m_i itself. Large intervals may result in a large number of columns to be explored by the pGPU$_f^{l,k}$ problems.

In Figure 8, we show how the time needed by the Step II depends on the length of the queuing jitter intervals $[J_i^{min}..J_i^{max}]$. We vary the J_i^{max} between 2% and 16%. We consider the same set of 20 messages as considered in Figure 7. Once again, these experiments were for the message with the lowest priority.

The Figure 8 shows the impact of (i) dividing of the MILP$_h$ into several smaller problems (Section VI-A) and (ii) the use of GPUs (Section VII-A). In the figure, $l = 1$ implies that we invoked the MILP$_h$ problem (i.e., the hyper-period is a single segment) which processes Step I. Thereafter, Step II is either performed on the CPU or on the GPU. As seen from the graph, without the GPU, Step II on the CPU (line CPU, $l = 1$) cannot scale beyond 4% of J_i^{max}. However, with MILP$_h$, Step II with the GPU (line GPU, $l = 1$) cannot scale beyond 6% of J_i^{max}. This is because the MILP$_h$ generates such a large number of combinations of columns that it cannot accomodated in the GPU memory. This, once again, highlights the importance of breaking down the MILP$_h$.

As shown in the graph, with $l = 6$, when we divide the hyper-period into 6 segments, now Step II can scale further. Thus, with the MILP$_f^l$ problem (instead of MILP$_h$), Step II on the CPU (line CPU, $l = 6$) scales up to 12%. The best performance is delivered when we use MILP$_f^l$ for the Step I and the GPU for Step II, observed in the graph (line GPU, $l = 6$).

C. Quality of the results

The low running times achieved by Step I are due to the division of the time interval Δt into a number of segments as presented in Section VI-A. As discussed in Section VI-A, this division can possibly introduce pessimism in computing the discrete random variable $Y_i^{\mathcal{U} \stackrel{\Delta t}{\Rightarrow} \mathcal{V}}$. In order to experimentally verify the degree of pessimism we conducted a set of experiments. This was conducted for a set of 20 messages with jitter varying from 2% to 6%. It is not possible to compare beyond this range because with MILP$_h$ formulation, the tool does not scale beyond 6% as shown in Figure 8. Our results, interestingly, reported no deviations at all between the solutions from MILP$_h$ and MILP$_f^{l,k}$. The reason for this is that with small jitters, the intermediary graphs generated by MILP$_f^l$ can capture the same temporal behavior as covered by MILP$_h$.

IX. Conclusion

We have proposed a method towards probabilistic analysis of the DYN segment of FlexRay while addressing the scalability of the problem by novel techniques. The experimental results highlight the significance of our proposed scheme.

References

[1] M. Ahmed, N. Fisher, and D. Grosu. A parallel algorithm for edf-schedulability analysis of multi-modal real-time systems. In *RTCSA*, 2012.

[2] J.L. Diaz, D.F. Garcia, Kanghee Kim, Chang-Gun Lee, L. Lo Bello, J.M. Lopez, Sang Lyul Min, and O. Mirabella. Stochastic analysis of periodic real-time systems. In *RTSS*, 2002.

[3] J. Feng, S. Chakraborty, B. Schmidt, W. Liu, and U. D. Bordoloi. Fast schedulability analysis using commodity graphics hardware. In *RTCSA*, 2007.

[4] B. Kim and K. Park. Analysis of frame delay probability in the FlexRay dynamic segment. In *International Conference on Industrial Informatics*, 2008.

[5] S. Manolache, P. Eles, and Z. Peng. Schedulability analysis of applications with stochastic task execution times. *Trans. Embed. Comput. Syst.*, 3(4), November 2004.

[6] M. Neukirchner, M. Negrean, R. Ernst, and T.T. Bone. Response-time analysis of the FlexRay dynamic segment under consideration of slot-multiplexing. In *Symposium on Industrial Embedded Systems*, 2012.

[7] T. Nolte, H. Hansson, and C. Norström. Probabilistic worst-case response-time analysis for the controller area network. In *RTAS*, 2003.

[8] B. Tanasa, U.D. Bordoloi, S. Kosuch, P. Eles, and Z. Peng. Schedulability analysis for the dynamic segment of FlexRay: A generalization to slot multiplexing. In *RTAS*, 2012.

[9] H. P. Williams. *Model Building in Mathematical Programming*. Wiley, 1999.

[10] H. Zeng, M. Di Natale, P. Giusto, and A. Sangiovanni-Vincentelli. Stochastic analysis of CAN-based real-time automotive systems. *Transactions onIndustrial Informatics*, 5(4):388 –401, 2009.

[11] 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.