Timing Analysis of Cyber-Physical Applications for Hybrid Communication Protocols

Alejandro Masrur*, Dip Goswami*, Samarjit Chakraborty*
Jian-Jia Chen†, Anuradha Annaswamy‡ and Ansuman Banerjee§
*TU Munich, Germany; †Karlsruhe Institute of Technology, Germany
‡Massachusetts Institute of Technology, USA; §Indian Statistical Institute, Kolkata, India

***Abstract*—Many cyber-physical systems consist of a collection of control loops implemented on multiple electronic control units (ECUs) communicating via buses such as FlexRay. Such buses support hybrid communication protocols consisting of a mix of time- and event-triggered slots. The time-triggered slots may be perfectly synchronized to the ECUs and hence result in zero communication delay, while the event-triggered slots are arbitrated using a priority-based policy and hence messages mapped onto them can suffer non-negligible delays. In this paper, we study a switching scheme where control messages are dynamically scheduled between the time-triggered and the event-triggered slots. This allows more efficient use of time-triggered slots which are often scarce and therefore should be used sparingly. Our focus is to perform a schedulability analysis for this setup, i.e., in the event of an external disturbance, can a message be switched from an event-triggered to a time-triggered slot within a specified deadline? We show that this analysis can check whether desired control performance objectives may be satisfied, with a limited number of time-triggered slots being used.**

I. Introduction

In this paper we are concerned with distributed cyber-physical architectures where multiple control applications are mapped into spatially distributed electronic control units (ECUs) communicating via a hybrid communication protocol such as FlexRay [1]. We address the *semantic* gap arising from the communication delay experienced by the control messages while being transmitted over time- (TT) or event-triggered (ET) segments of the bus. A *zero/negligible* communication delay may be achieved when all the control messages are mapped onto the static TT segment of the bus with perfectly synchronized TT slots and ECUs. Clearly, the controller based on such zero communication delay leads to a good control performance. However, such TT implementations might be overly expensive because of their high communication bandwidth requirements. On the other hand, priority-driven ET implementations suffer from the usual temporal non-determinism, i.e., the communication delay varies with the priority and the current scheduling situation on the bus. In such ET scheme, a controller is designed based on the *worst-case delay* and might results in a poor control performance.

In this paper we investigate an intermediate possibility where the aim is to achieve control performance close to a purely TT implementation, but using *fewer* TT slots than what would be necessary for purely TT communication. Towards this, we exploit the fact that the time required by a control application to reject an external disturbance (or *response time*) is considerably lesser with the controller based on TT communication compared to the one based on ET communication. To meet a specified response time requirement of a control application, we appropriately switch between the TT and ET modes as originally proposed in [2]. A *schedulability analysis* is necessary, since the number of allocated TT slots is less than what is required for *all* control messages to be accommodated. Hence, in the event of a disturbance, an application might have to wait (depending on whether its associated TT slot is occupied or not) before it may switch from an ET to a TT mode. Designing and analyzing such a *control performance-oriented scheduling* is the topic of this paper.

978-3-9810801-8-6/DATE12/ © 2012 EDAA

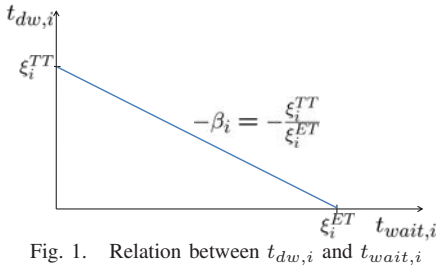

Fig. 1. Relation between $t_{dw,i}$ and $t_{wait,i}$

Our contributions and related work: There are two broad classes of schedulability analysis techniques within the real-time systems literature – response time analysis [3] and the demand-bound criteria [4]. In this paper, we lift the classical response time analysis technique to a control-theoretic setting. In particular, we propose a switching scheme between the TT mode (zero-delay controller and TT communication) and ET mode (worst-case delay controller and ET communication). In this switching scheme, multiple control applications *share* the same TT slot and they request to move to TT mode whenever an external disturbance arrives. When multiple applications experience disturbance simultaneously, a control application C_i might have to wait $t_{wait,i}$ time units (as its associated TT slot is occupied) before it moves to the TT mode. Once a control application is in TT mode, it requires $t_{dw,i}$ time units to spend in that mode for complete disturbance rejection. While C_i is waiting in the ET mode – during $t_{wait,i}$, a fraction of disturbance already gets rejected and hence the controller needs lesser time in the TT mode. That is, $t_{dw,i}$ gets shorter with increase in $t_{wait,i}$ (see Fig. 1 – parameters are explained later). Such waiting time leads to the higher response time and hence, the higher possibility of violating the desired response time requirements. In this paper, we present a formal schedulability analysis to compute the necessary number of static TT slots such that the response time requirements of a given set of control applications are met.

While there has been previous work on timing analysis of both TT [5] and a mix of TT and ET systems [6], the questions addressed were typically the following. (i) How to compute upper bounds on communication delays? (ii) How to synthesize TT schedules (see also [7] for TT schedule synthesis for FlexRay)? There has also been some work on partitioning system functionality into TT and ET activities. However, the schedulability analysis problem arising from dynamically switching messages between TT and ET modes, and in particular, analysis with control performance objectives has not been sufficiently addressed so far. Notable exceptions to this are [8] and [9]. The work presented in [8] studied how the performance of multiple control loops may be optimized

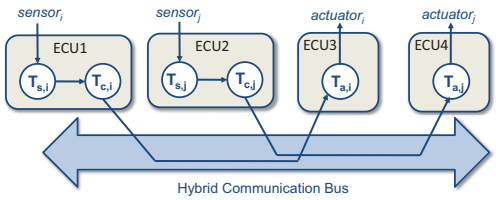

Fig. 2. The distributed cyber-physical architecture in this paper

while still ensuring schedulability in CAN networks. Similarly, the schedulability region that guarantees control performance has been computed in [9]. Controller-scheduling co-synthesis has also been studied in [10], [11]. Our approach follows this line of work and specifically computes the necessary number of TT slots, while maintaining the desired response times for control applications (which requires a schedulability analysis).

Schedulability analysis in the context of hybrid TT and ET protocols like FlexRay have also been addressed in the past using techniques like model checking [12], [13]. This requires a state-based model of the system under analysis along with explicit specification of deadlines as properties in real-time fragments of temporal logic. A recent research [12] reports the use of the SPIN model checker for jointly optimizing the static task and bus access schedule for TT systems. It is theoretically possible to cast our problem also as a model checking exercise. However, the underlying model checker has to explore a significantly large state space arising out of the inter-leavings possible in the underlying system due to the switching activity. This is aggravated by the fact that a precise schedulability analysis has to consider all possible patterns for disturbance arrival and possible job migration between the TT and ET slots. In this paper we therefore, pursue a classical algebraic, rather than a model checking approach.

Organization: The rest of this paper is organized as follows. We formally formulate the schedulability problem in Section III. Subsequently, Section II provides the formal characterization of the control applications in the context of the presented schedulability analysis. This is followed by the discussion on the proposed scheduling algorithm in Section IV. We illustrate the applicability of our algorithm with a case study in Section V.

II. Problem Formulation

We consider a set of multiple control applications C_i with sampling period p_i ($i \in \{1, 2 \dots n\}$) that run on a distributed architecture of the form shown in Fig. 2. The control applications are represented by:

$$x[k+1] \quad = \quad A_i x[k] + B_i u[k], \tag{1}$$

where $x[k]$ is the plant state, $u[k]$ is the control input and A_i, B_i are the system matrices of C_i. Each C_i is composed of three tasks $T_{s,i}$ (measures $x[k]$), $T_{c,i}$ (computes $u[k]$) and $T_{a,i}$ (applies $u[k]$ to the actuator/plant)—see (1). Such tasks are then mapped onto spatially distributed ECUs which are connected via a hybrid communication bus as shown in Fig. 3.

Each communication cycle on the bus is divided into time-triggered (or static) and event-triggered (or dynamic) segments. On the TT segment, the tasks are given access to the bus (or allowed to send messages) only at their predefined *slots*. On the other hand, the tasks are assigned *priorities* in order to arbitrate for the access to the ET segment. Further, we consider a distributed setup with the following properties:

- The tasks $T_{s,i}$ and $T_{c,i}$ are mapped onto the same ECU which is attached to the corresponding sensors. $T_{s,i}$ triggers $T_{c,i}$ after measuring the states $x[k]$. Our analysis can also be extended to other task mappings as well.
- The tasks $T_{s,i}$ and $T_{a,i}$ that belong to a particular control application are triggered *periodically* with the same period (which is dictated by the sampling time p_i). The triggering of $T_{s,i}$ and $T_{a,i}$ is synchronized with a given slot on the static segment of the bus.
- The execution times of $T_{s,i}$, $T_{c,i}$ and $T_{a,i}$ (in the order of a few μs) are negligible compared to the sampling period p_i (in the order of tens of ms).
- Every controller task $T_{c,i}$ can send messages (to $T_{a,i}$) either over the static or the dynamic segment of the bus. The transmission rate in FlexRay is usually 10 Mbit/s. As a result, the transmission time of messages over the bus are generally in the order of μs which is negligible compared to the sampling periods of common control applications which are in the order of ms. We further assume that the slot length on the static segment has been chosen such that every possible message fits (entirely) into one slot. Therefore, we can consider that the transmission time of messages is zero (i.e., negligible with respect to the sampling period). On the dynamic segment, $T_{c,i}$'s messages experience a maximum communication delay τ_i. This is due to the contention among messages with different priorities [14]. We assume that the priority assigned to every $T_{c,i}$ on the dynamic segment guarantees that $0 < \tau_i \leq p_i$ holds for the corresponding C_i.

A controller $u[k]$ aims to achieve asymptotic stability (or stable *regulation*), i.e., $x[k] \rightarrow$ *reference* as $k \rightarrow \infty$. Without loss of generality, we assume that the reference is *zero*. In a steady-state, the values of every element of the vector $x[k]$ are close to zero (or reference) and hence, $x[k]'x[k]$ is small. In the context of stability of a control application, $x[k]'x[k]$ often acts as a measure of the system state or energy level. The deviation of $x[k]'x[k]$ from the reference that is tolerated by the designer in steady-state is E_{th}. In this work, anything that causes $x[k]'x[k] > E_{th}$ is referred to as a *disturbance*. The control goal is to bring back the system to steady-state (by making $x[k]'x[k] \leq E_{th}$) within a finite amount of time from the point at which the disturbance occurred. The amount of time required by a control application to achieve a steady-state from the point at which a disturbance has occurred, is referred as response time ξ_i.

We consider a *state-feedback controller* with communication delay in the feedback signals, i.e., $u[k] = Kx[k - \Delta]$, where K is the *state-feedback gain* [15] and Δ is the communication (feedback) delay measured in number of samples. $\Delta = 0$ implies the ideal case with zero communication delay while $\Delta = 1$ indicates communication delay of one sampling interval. For $\Delta = 0$, it is possible to adapt well-known optimal control approaches such as Linear Quadratic Regulator (LQR) [16] to derive the optimal feedback gain $K = K_{opt}$. In the case of $\Delta = 1$, the design of K relies on non-optimal *pole placement* technique [16] (i.e., $K = K_1$) as $u[k]$ has an older state $x[k-1]$ in feedback rather than the current state $x[k]$.

A controller implemented over a purely ET communication is essentially based on the worst-case delay, i.e., $u[k] = K_1 x[k-1]$. Such a controller is often quite pessimistic

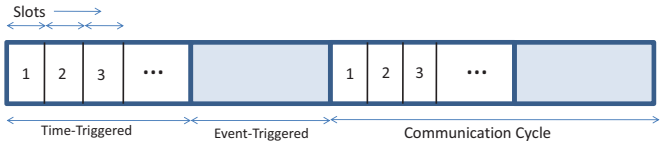

Fig. 3. Hybrid communication protocol: *time-* and *event-triggered* segments

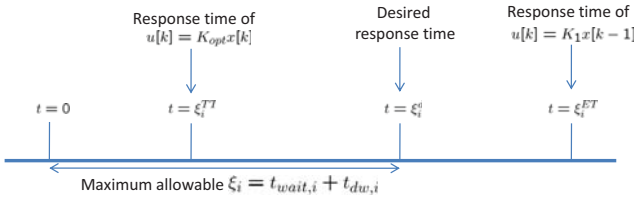

Fig. 4. Relation among ξ_i^{TT}, ξ_i^{ET}, ξ_i, ξ_i^d, $t_{dw,i}$ and $t_{wait,i}$

(illustrated in Section III). On the other hand, we normally have a limited number of TT slots which makes a zero-delay controller, i.e., $u[k] = K_{opt}x[k]$, be expensive. By switching between the zero- and worst-case delay controllers, we can countervail some of the pessimism incurred in a worst-case delay design and, at the same time, economize TT slots. The proposed switching scheme is described next:

- A control application C_i can either apply a zero-delay (i.e., $u[k] = K_{opt}x[k]$) or a worst-case delay controller (i.e., $u[k] = K_1x[k-1]$). In both cases, the asymptotic stability is guaranteed, i.e., $x[k] \rightarrow$ *reference* as $k \rightarrow \infty$. However, the response time ξ_i is lower in the case of using $K_{opt}x[k]$ and higher with $K_1x[k-1]$.
- Every control application is associated with a desired response time or deadline ξ_i^d. That is, after the occurrence of any disturbance, the control application must get back to steady-state within ξ_i^d.
- To meet the response time requirement ξ_i^d in the presence of disturbances, the control application C_i needs to apply $u[k] = K_{opt}x[k]$ for $t_{dw,i}$ time. That is, C_i requires to send $\frac{t_{dw,i}}{p_i}$ messages with zero delay. The application of only $u[k] = K_1x[k-1]$ causes a violation of the response time requirement ξ_i^d—see Fig. 4.
- The value of $t_{dw,i}$ (i.e., the number of necessary zero-delay messages to meet ξ_i^d) depends on the time $t_{wait,i}$. This accounts for the time that C_i spends using $u[k] = K_1x[k-1]$ after a disturbance and, hence, sending delayed messages. In general, the zero-delay and the delayed controller both tend to stabilize the system—with the distinction that the delayed controller is slower in meeting the performance requirements. Therefore, the time spent by C_i with the delayed controller already allows *rejecting* some amount of disturbance. This disturbance rejection by the delayed controller reduces the amount of work that needs to be done by the zero-delay controller. The relation between $t_{dw,i}$ and $t_{wait,i}$ is illustrated in Fig. 1 (explained in Section III). The slope β_i can closely be approximated as the ratio between the response time ξ_i^{TT} of a purely TT (i.e., zero-delay) controller and the response time ξ_i^{ET} of a purely ET (i.e., delayed) controller. Hence, $t_{dw,i}$ is given by $\xi_i^{TT} - \beta_i t_{wait,i}$. Since ξ_i^{TT} is strictly less than ξ_i^{ET}, $\beta_i < 1$ holds. Note that a $t_{wait,i} = \xi_i^{ET}$ results in a $t_{dw,i} = 0$; however, this also implies a *deadline violation* as shown in Fig. 4.

TABLE I
ILLUSTRATIVE EXAMPLE

$Case$	$t_{wait,i}(samples)$	$t_{dw,i}(samples)$	$\xi_i(samples)$
1	1	14	15
2	3	13	16
3	5	12	17
4	9	11	20
5	13	10	23

Clearly, if every C_i has its own slot on the static segment, then all of them will be able to meet their response time requirements ξ_i^d because there will be no contention for the static segment. However, this leads to a poor overall bus utilization and an expensive design. Since $\xi_i^{TT} < \xi_i^d$ normally holds, applications can tolerate some contention $\xi_i^d - \xi_i^{TT}$ and still meet their deadlines. Hence, we propose allocating multiple applications to the same TT slot. Now, the access to these *shared* slots needs to be arbitrated which leads us to the following schedulability problem.

Problem statement: We consider n control applications C_i with ξ_i^{TT}, ξ_i^{ET} and ξ_i^d where $i \in \{1, 2 \ldots n\}$. Given a bound on the disturbances for each application, we intend to compute the minimum number of static segment slots m ($m \leq n$) to ensure that all control applications meet their response time requirements ξ_i^d.

III. DETAILS OF CONTROL APPLICATIONS

In this section, we illustrate the behavior of the control applications C_i ($i \in \{1, 2 \ldots n\}$) shown in Fig. 2 using a second-order discrete-time plant of the form (1) and,

$$A = \begin{bmatrix} 0.4 & -1.2 \\ -2.56 & -1.9 \end{bmatrix}, B = \begin{bmatrix} 0.1 \\ 0.4 \end{bmatrix}. \quad (2)$$

We consider the case where $u[k] = K_1x[k-1]$ is applied for $t_{wait,i}$ time units after the occurrence of disturbance and subsequently, the controller switches to $u[k] = K_{opt}x[k]$ and is applied for $t_{dw,i}$ time units to bring back the control application to a steady-state. Table I shows various cases with different $t_{wait,i}$ and the corresponding $t_{dw,i}$ and ξ_i for the discrete-time plant (2) with initial conditions $x_1[0] = x_2[0] = 20$ and $E_{th} = 0.1$. It may be noticed that $t_{dw,i}$ decreases with an increase in $t_{wait,i}$ and their relation can closely be approximated as:

$$t_{dw,i} = \xi_i^{TT} - \beta_i t_{wait,i}, \quad (3)$$

Eq. (3) can be interpreted as follows: A fraction of the disturbance is already rejected by $u[k] = K_1x[k-1]$ during $t_{wait,i}$ and hence $u[k] = K_{opt}x[k]$ needs less time to bring back the system to steady-state (i.e., shorter $t_{dw,i}$).

The basic design consideration is that the response time ξ_i^{TT} of a purely zero-delay controller is considerably lesser than the response time ξ_i^{ET} of a purely worst-case delay controller. The slope β_i essentially captures this property and can therefore be approximated by $\beta_i = \frac{\xi_i^{TT}}{\xi_i^{ET}} < 1$. Further, the response time ξ_i is given by:

$$\begin{aligned} \xi_i &= t_{wait,i} + t_{dw,i}, \\ &= \xi_i^{TT} + (1 - \beta_i)t_{wait,i}. \end{aligned} \quad (4)$$

Based on the fact that $\beta_i < 1$, we can notice that ξ_i increases with an increase in $t_{wait,i}$ which is also supported by various cases shown in Table I. Thus, there is an upper-bound of $t_{wait,i}$ to meet a given deadline ξ_i^d. As

per (4), the upper-bound on $t_{wait,i}$ is longer for a shorter $t_{dw,i}$. That is, the control applications are allowed to spend more time in the ET mode for a shorter $t_{dw,i}$. Naturally, a shorter $t_{dw,i}$ is more desirable from schedulability perspective.

Stability in presence of switching: The shifting between the zero-delay and the worst-case delay controllers essentially results in a switched system. Once the controller switches from $u[k] = K_1 x[k-1]$ to $u[k] = K_{opt} x[k]$, it stays in TT mode for $t_{dw,i}$ time units. Here, $t_{dw,i}$ is chosen sufficiently long to bring the system back to steady-state from any initial condition at the point of switching. Now, consider the time interval from the point at which the disturbance occurred until it is fully rejected, i.e., ξ_i. During ξ_i, the controller switches only once after $t_{wait,i}$ time units. If it is assumed that each control application gets enough time to reject a disturbance before the next one arrives, then the system energy never becomes unbounded. This essentially avoids the possibility of instability arising from such switching control strategy.

Detailed problem statement: Given the above-described properties, there are two possible ways to choose $t_{dw,i}$ for a control application. First, $t_{dw,i}$ can be chosen the maximum time required by the zero-delay controller to reject a disturbance after switching, for all $t_{wait,i}$. For example, we can choose $t_{dw,i} = 14$ samples as in Case *1* (Table I) for all the cases. The schedulability analysis with such $t_{dw,i}$ will certainly provide a *safe* result. However, the actual $t_{dw,i}$ is shorter than 14 samples for the Cases *2-5*. Hence, the second possibility is to choose the actual $t_{dw,i}$ as per (3) to avoid pessimism. In this work, we address the schedulability analysis problem described in the previous section considering the actual $t_{dw,i}$.

IV. Slot Sharing and Schedulability

To determine the number of necessary slots on the static segment, we first need to decide on how control applications will access slots. In general, similar to scheduling real-time tasks on processors, there are two different ways of implementing a slot-sharing strategy in our setup. First, applications can be assigned fixed slots in a partitioned scheme. Second, applications can be dynamically scheduled on slots in a global scheme. In this paper, we focus on the partitioned scheme, i.e., each application is assigned to a single slot such that it always uses the same slot when transmitting over the static segment. To determine the necessary number of slots, we need to analyze the schedulability of a set of applications on one slot. Based on such a schedulability analysis, we can allocate applications to one or more slots accordingly.

A. Schedulability Analysis

The schedulability analysis on one slot requires two inputs: **(i)** the performance-related requirements derived from the control design, **(ii)** the disturbance arrival pattern.

In principle, a TT slot behaves as a *processor* with a certain processing capacity. The control applications C_i requesting for zero-delay transmission behave like *tasks* running on the TT slot. At a disturbance, a control application requests the TT slot for a given amount of time $t_{dw,i} \leq \xi_i^{TT}$. $t_{dw,i}$ here behaves as the *execution time* of C_i. A TT slot or processor must then provide this amount of *service* to C_i within a *deadline* ξ_i^d.

A request for zero-delay communication $t_{dw,i}$ coming from a C_i depends on the disturbance arrival pattern of C_i, which we characterize in the next paragraph.

Disturbance model: For a control application C_i, disturbances may arrive sporadically with a minimum inter-arrival time denoted by r_i. In this paper, we consider the case where $\xi_i^d \leq r_i$ holds for every C_i in the system. That is, any control application is assumed to have enough time to recover from a disturbance before the next one arrives. The sources of disturbance are assumed to be independent of each other. Consequently, the worst-case disturbance arrival pattern happens when disturbances occur simultaneously with their respective minimum inter-arrival times r_i for all C_i in the system.

From the previous discussion, we know that C_i needs to recover from disturbances within ξ_i^d time units. For this purpose, a C_i has to send $\frac{t_{dw,i}}{p_i}$ zero-delay messages. However, as discussed previously, $t_{dw,i}$ varies with the time $t_{wait,i}$ that C_i remains in the ET regime (sending delayed messages). This behavior requires special attention.

In order to schedule a number of control applications C_i on the same TT slot, we implement a priority-based slot sharing. All C_i sharing one slot on the static segment are assigned priorities according to their criticality. For this purpose, we make use of the Deadline Monotonic (DM) policy, i.e., the shorter the deadline of a C_i, the higher its priority on the given slot. As mentioned before, the deadline of a C_i here is given by its desired response time ξ_i^d. Note that the technique proposed in this paper trivially extends to other priority assignments.

From our previous discussion, we know that a control application can be switched at most once between the ET and TT regime during a disturbance. Otherwise, the stability of the switching would be compromised. That is, once an application C_i has access to a TT slot, it requires blocking the slot for $t_{dw,i}$ amount of time (i.e., until it finishes transmitting $\frac{t_{dw,i}}{p_i}$ messages). As a result, the scheduling of a sequence of messages on the static segment must be implemented in a non-preemptive manner.

Independent of its priority, an application C_i will have to wait to have access to the TT slot, if this is being used by another application. This increases its waiting time $t_{wait,i}$ of C_i. Hence, its demand for zero-delay communication $t_{dw,i}$ decreases as discussed before—see Fig. 1. However, as discussed before, the overall response time of the application increases with $t_{wait,i}$ (see Eq. (4)). This is because the parameter $\beta_i = \frac{\xi_i^{TT}}{\xi_i^{ET}}$ is always less than one.

The schedulability of a control application C_i on a shared TT slot will then be guaranteed, if the following condition holds for every possible $t_{wait,i}$: $\xi_i^d \geq (1-\beta_i) t_{wait,i}$.

Hence, to test the schedulability of C_i, we need to find the greatest possible $t_{wait,i}$ (denoted by $\hat{t}_{wait,i}$) which leads to the worst-case response time of C_i (denoted by $\hat{\xi}_i$). This occurs when C_i suffers the maximum possible interference due to higher-priority applications. For this, we will consider that all higher-priority applications C_j interfering with C_i require their maximum possible transmission time on the shared slot, i.e., $t_{dw,j} = \xi_j^{TT}$. This assumption is pessimistic since $t_{dw,j}$ actually decreases with the blocking time suffered by C_j. However, this allows us to simplify the analysis and leads to a *safe* schedulability condition. Under this assumption, the worst-case interference on C_i clearly occurs when it needs to have access to the TT slot together with all higher-priority C_j (sharing the same slot). This again happens when all higher-priority C_j and C_i undergo disturbances at the same time.

(Recall that the sources of disturbance are independent of each other.) Since the scheduling is non-preemptive, C_i may also suffer some blocking time due to lower-priority applications.

Computing $\hat{t}_{wait,i}$ and $\hat{\xi}_i$ here has some similarities with computing the worst-case response time in a fixed-priority non-preemptive scheduling like the one of CAN [17], [18]. That is, we need to compute the response times of all *jobs* of that task within its *maximum busy period* [18].

In our case, the task is given by a control application C_i sending a certain number of consecutive messages over a shared slot. The maximum busy period $t_{max,i}$ of a C_i is then the largest time interval in which the shared slot is constantly being used by higher-priority control applications and by C_i itself. For ease of exposition, we assume that $t_{max,i} \leq r_i$ holds for all C_i in this paper, i.e., there is only one transmission of $\frac{t_{dw,i}}{p_i}$ messages of C_i within its busy period $t_{max,i}$. This way, we only need to compute the response time ξ_i of the sole *job* of C_i within $t_{max,i}$ to obtain its worst-case response time $\hat{\xi}_i$, which can be done in the following manner:

$$\xi_i = \xi_i^{TT} + (1-\beta_i)b_i + (1-\beta_i)\sum_{j=1}^{i-1}\left\lceil \frac{\xi_i}{r_j} \right\rceil \xi_j^{TT}, \quad (5)$$

where $b_i = \max_{k=i+1}^{n}(\xi_k^{TT})$ denotes the maximum possible blocking time due to lower-priority applications suffered by C_i and n is the number of applications. Without loss of generality, we assume in Eq. (5) and in the remainder of the paper that applications are sorted in order of decreasing priority (i.e., C_j has higher priority than C_i and C_i has higher priority than C_k for $1 \leq j < i < k \leq n$). Eq. (5) can be solved starting from $\xi_i = \xi_i^{TT} + (1-\beta_i)b_i$ and computing it iteratively until ξ_i becomes greater than ξ_i^d or converges to a certain value. Clearly, if ξ_i exceeds ξ_i^d, C_i is not schedulable on the shared slot. On the other hand, if there is a convergence value prior to ξ_i^d, then C_i can meet its deadline and is schedulable.

B. Reducing lower-priority blocking time

For the sake of stability, a control application can only be switched once from ET to TT during a disturbance rejection, which resulted in a non-preemptive scheduling studied in the previous section.

Under the considered DM policy, lower-priority applications have longer deadlines and normally higher communication requirements, i.e., greater ξ_i^{TT}. Hence, in a non-preemptive scheme, a higher-priority C_j may be blocked for a long time to have access to the TT shared slot.

At the other extreme, in every communication cycle, it can be decided which application starts transmitting next on the TT slot. As long as every C_i is switched only once from TT to ET, we can use this fact to mitigate the blocking time suffered by higher-priority applications.

First, we compute the maximum possible blocking time $\hat{b}_i$ an application can withstand without missing its deadline. This can be done replacing ξ_i by ξ_i^d (i.e., the deadline) in Eq. (5) and then resolving for b_i. The resulting value is $\hat{b}_i$:

$$\hat{b}_i = \frac{\xi_i^d - \xi_i^{TT}}{1-\beta_i} - \sum_{j=1}^{i-1}\left\lceil \frac{\xi_i^d}{r_j} \right\rceil \xi_j^{TT}. \quad (6)$$

An application C_i can wait (or be blocked by a lower-priority C_k) up to $\hat{b}_i$ time units. After that time, it needs to switch to the TT regime in order to meet its deadline under all possible conditions (i.e., considering the maximum possible interference by higher-priority C_j according to Eq. (6).

If we compute $\hat{b}_i$ from the lowest- to the highest-priority application, we can then calculate t_i in the following manner:

$$t_i = \hat{b}_i - \max_{k=i+1}^{n}\left(\xi_k^{TT} - \beta_k t_k\right), \quad (7)$$

which stands for the time C_i can wait at maximum before switching to TT minus the maximum blocking time by lower-priority C_k (which also wait for a time t_k to switch to TT).

When a disturbance arrives, we now force all applications C_i to wait in the ET regime up to t_i time units before switching to TT. As a result, the maximum blocking time suffered by any C_i will be given by:

$$b_i' = \max_{k=i+1}^{n}\left(\xi_k^{TT} - \beta_k t_k\right). \quad (8)$$

As it can be seen from Eq. (8), b_i' is always less or equal to b_i. Hence, the blocking time due to lower-priority applications can be reduced this way.

Replacing b_i by b_i' in Eq. (5) and proceeding as explained before, we can then test the schedulability of applications under this new scheme with reduced lower-priority blocking.

Algorithm 1 Computation of the number of slots

Require: Set of control applications C_i with ξ_i^d and ξ_i^{TT}
Require: The minimum disturbance inter-arrival time r_i for every C_i
1: number_slots=1
2: Sort C_i according to decreasing priority
3: **for** $i = 1$ to n **do**
4: **for** $s = 1$ to number_slots **do**
5: **if** Schedulable(C_i,slot(s)) **then**
6: Allocate C_i to slot(s)
7: **else if** s==number_slots **then**
8: number_slots = number_slots + 1
9: Allocate C_i to slot(number_slots)
10: **end if**
11: **end for**
12: **end for**
13: Return number_slots

C. Allocation Algorithm

The problem of finding the minimum number of slots (that guarantees the response time requirements of all C_i) is clearly an allocation problem. Often such problems are NP-hard in the strong sense, i.e., finding an optimal solution results in exponential complexity.

The technique proposed in this paper is based on the well-known First Fit (FF) heuristic. FF leads to a number of slots that is acceptably close to the optimum and has polynomial complexity. Our algorithm (Alg. 1) first sorts the control applications C_i according to increasing priority (i.e., in the case of DM, according to increasing values of ξ_i^d). Then, it iterates over the sorted set of C_i and tries to allocate them in the minimum possible number of slots.

The algorithm we propose starts with only one slot and allocates the control applications C_i to it as long as they are schedulable on that slot (line 5). A C_i is schedulable on one slot if it can meet its timing requirement ξ_i^d when assigned to that slot. To test this, the proposed algorithm makes use of the schedulability analysis presented in the previous section.

Our algorithm allocates all C_i to one or more slots in the list of existing slots (line 4 to 11). If a C_i could not be scheduled on any of the exiting slots, it then adds a slot to the list (line 8). The algorithm concludes when all C_i have been

TABLE II
CONTROL APPLICATIONS: PARAMETERS IN MS

C_i	r_i	ξ_i^d	ξ_i^{TT}	ξ_i^{ET}	$\hat{\xi}_i(b_i)$	$\hat{\xi}_i(b_i')$
C_1	2000	150	50	200	87.50	98.70
C_2	2000	500	200	550	327.27	263.63
C_3	1500	150	50	200	87.50	136.20
C_4	2000	300	200	400	300.00	280.35
C_5	5000	1000	800	2000	800.00	800.00
C_6	600	600	300	700	300.00	414.28

allocated and returns the number of slots that were necessary for accommodating all of them (line 13).

V. EXPERIMENTAL RESULTS

In this section, we evaluate the proposed switching scheme through an illustrative example. We consider six control applications with the parameters shown in Table II. The communication protocol is assumed to be FlexRay with a cycle length of 5 ms. The static segment has 2 ms length and it is divided into 10 slots. The rest of the cycle is assigned to the dynamic segment.

Results and discussion: Given the six control applications shown in Table II, we apply the non-preemptive scheduling analyzed in Section IV-A and determine the necessary number of slots that guarantee all requirements using Alg. 1.

For the considered example, we obtained four slots with the following partitioning: $\{C_1, C_3\}$, $\{C_4, C_2\}$, $\{C_6\}$ and $\{C_5\}$. On the other hand, using the technique described in Section IV-B to reduce the lower-priority blocking time, we can allocate the six applications into three slots: $\{C_1, C_3, C_2\}$, $\{C_4, C_6\}$ and $\{C_5\}$. Although we can improve (i.e., reduce) the number of necessary slots, this is normally achieved at the cost of higher response times—see column $\hat{\xi}_i(b_i')$, i.e., $\hat{\xi}_i$ obtained with b_i' of Eq. (8). This is because the technique presented in Section IV-B forces applications to wait for t_i time (see Eq. (7)) before switching to the TT slot. However, the resulting response times of applications depend also on the other applications that are mapped to the same slots.

The experiments demonstrate that the proposed switching scheme allows saving TT slots with respect to a purely TT scheme, which requires six slots. Using the non-preemptive scheduling of Section IV-A, it is possible to reduce the number of necessary TT slots by two. On the other hand, using the technique of Section IV-B to reduce lower-priority blocking, it is further possible to save up an additional slot, i.e., in this latter case, we used half of the TT slots that a purely TT solution requires.

Finally, Fig. 5 shows the *schedulability region* for the applications C_1, C_2, C_3 and C_4. For C_4 given as in Table II, we vary the disturbance arrival rates of C_1, C_2 and C_3. As it can be noticed, for $r_1 = 250$ and $r_2 = 1000$ ms, these applications are only schedulable for an $r_3 = 10$ ms. Further, an $r_3 = 300$ ms is then possible if we decrease r_2 to 700 ms.

VI. CONCLUDING REMARKS

In this paper we proposed a switching strategy for distributed control applications communicating via a hybrid event-/time-triggered protocol. The response times of the control applications are considerably shorter in the TT compared to the ET mode. However, a TT implementation essentially results in poor bus utilization and hence in an expensive design. The approach we followed in this paper allows for a performance close to that of a purely TT scheme using

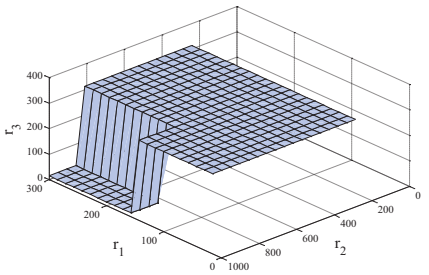

Fig. 5. Schedulability region

fewer TT slots. Towards this, we proposed a priority-based slot sharing scheme, for which we presented and analyzed its schedulability. The novelty of our approach lies in the formal characterization of control performance requirements in the context of schedulability analysis. As a part of future work, we plan to investigate the impact of different kinds of disturbance models (rather than always assuming the *worst-case disturbance arrival*) and extend our analysis to handle them in a conservative fashion.

REFERENCES

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

[2] D. Goswami, R. Schneider, and S. Chakraborty, "Re-engineering cyber-physical control applications for hybrid communication protocols," in *Design, Automation and Test in Europe (DATE)*, Grenoble, France, 2011.

[3] K. Tindell, A. Burns, and A. J. Wellings, "An extendible approach for analyzing fixed priority hard real-time tasks," *Real-Time Systems*, vol. 6, no. 2, pp. 133–151, 1994.

[4] S. Baruah, "Dynamic- and static-priority scheduling of recurring real-time tasks," *Real-Time Systems*, vol. 24, no. 1, pp. 93–128, 2003.

[5] P. Pop, P. Eles, and Z. Peng, "Schedulability-driven communication synthesis for time triggered embedded systems," *Real-Time Systems*, vol. 26, no. 3, pp. 297–325, 2004.

[6] T. Pop, P. Eles, and Z. Peng, "Design optimization of mixed time/event-triggered distributed embedded systems," in *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2003.

[7] M. Lukasiewycz, M. Glaß, J. Teich, and P. Milbredt, "Flexray schedule optimization of the static segment," in *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2009.

[8] A. Martinez and P. Tabuada, "On the benefits of relaxing the periodicity assumption for networked control systems over CAN," in *Real-Time Systems Symposium (RTSS)*, 2009.

[9] F. Zhang, K. Szwaykowska, W. Wolf, and V. J. Mooney, "Task scheduling for control oriented requirements for cyber-physical systems," in *Real-Time Systems Symposium (RTSS)*, 2008.

[10] S. Samii, A. Cervin, P. Eles, and Z. Peng, "Integrated scheduling and synthesis of control applications on distributed embedded systems," in *Design Automation and Test in Europe (DATE)*, 2009.

[11] D. Fontanelli, L. Palopoli, and L. Greco, "Deterministic and stochastic QoS provision for real-time control systems," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2011.

[12] Z. Gu, X. He, and M. Yuan, "Optimization of static task and bus access schedules for time-triggered distributed embedded systems with model-checking," in *Design Automation Conference (DAC)*, 2007.

[13] Z. Gu, "Solving real-time scheduling problems with model-checking," in *IEEE International Conferences on Embedded Software and Systems (ICESS)*, 2005.

[14] H. Zeng, A. Ghosal, and M. Di Natale, "Timing analysis and optimization of flexray dynamic segment," in *IEEE International Conference on Computer and Information Technology (ICCIT)*, 2010.

[15] W. Jiang, E. Fridman, A. Kruszewski, and J. Richard, "Switching controller for stabilization of linear systems with switched time-varying delays," in *IEEE Conference on Decision and Control (CDC)*, 2009.

[16] K. Zhou, J. C. Doyle, and K. Glover, *Robust and Optimal Control*. Prentice Hall, New Jersey, 1996.

[17] K. Tindell, H. Hansson, and A. Wellings, "Analysing real-time communications: Controller area network (CAN)," in *Real-Time Systems Symposium (RTSS)*, 1994.

[18] R. Davis, A. Burns, R. Bril, and J. Lukkien, "Controller area network (CAN) schedulability analysis: Refuted, revisited and revised," *Real-Time Systems*, vol. 35, no. 3, pp. 239–272, 2007.