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 nondeterminism, 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.

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

*978-3-9810801-8-6/DATE12/ © 2012 EDAA
where the architecture of the form shown in Fig. 2. The control applica-
tion approach. The rest of this paper is organized as follows.
precise schedulability analysis has to consider all possible
the switching activity. This is aggravated by the fact that a
has to explore a significantly large state space arising out of
theoretically possible to cast our problem also as a model
fragments of temporal logic. A recent research [12] reports
a state-based model of the system under analysis along with
protocols like FlexRay have also been addressed in the past
10, [11]. Our approach follows this
has been computed in [9]. Controller-scheduling co-synthesis
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 characteriza-
tion 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 \( p_i \) (\( i \in \{1, 2, \ldots, n\} \)) that run on a distributed
architecture of the form shown in Fig. 2. The control applica-
tions are represented by:

\[
x[k+1] = Ax[k] + Bu[k],
\]

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 con-
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 \( \mu s \)) are negligible compared to the sampling period
\( p_i \) (in the order of tens of \( \mu s \)).

- 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 \( \mu s \) which is negligible
compared to the sampling periods of common control
applications which are in the order of \( \mu s \). 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 trans-
munication 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 \( \tau_i \). 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 \( 0 < \tau_i < p_i \) 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, \( x[k] \rightarrow x[k] \) is small.
In the context of stability of a control application, \( x[k] \rightarrow x[k] \)
often acts as a measure of the system state or energy level.
The deviation of \( x[k] \rightarrow x[k] \) from the reference that is tolerated
by the designer in steady-state is \( E_{th} \). In this work, anything
that causes \( x[k] \rightarrow x[k] > E_{th} \) is referred to as a disturbance.
The control goal is to bring back the system to steady-state
(by making \( x[k] \rightarrow x[k] 

\leq E_{th} \)) within a finite amount of time
from the point at which the disturbance has 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 \( \xi_i \).

We consider a state-feedback controller with communica-
tion delay in the feedback signals, i.e., \( u[k] = K_e x[k] - \Delta \),
where \( K \) is the state-feedback gain [15] and \( \Delta \) 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. 2. The distributed cyber-physical architecture in this paper](image-url)
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., $u[k] = K_1 x[k-1]$). In both cases, the asymptotic stability is guaranteed, i.e., $x[k] \rightarrow 0$ as $k \rightarrow \infty$. However, the response time $\xi$ is lower in the case of using $K_{opt}[x[k]]$ and higher with $K_1 x[k-1]$.

- Every control application is associated with a desired response time or deadline $\xi^d$. That is, after the occurrence of any disturbance, the control application must get back to steady-state within $\xi^d$.

- To meet the response time requirement $\xi^d$ in the presence of disturbances, the control application $C_i$ needs to apply $u[k] = K_{opt}[x[k]]$ for $t_{d_{wait},i}$ time. That is, $C_i$ requires to send $t_{d_{wait},i}$ messages with zero delay. The application of only $u[k] = K_1 x[k-1]$ causes a violation of the response time requirement $\xi^d$—see Fig. 4.

- The value of $t_{d_{wait},i}$ (i.e., the number of necessary zero-delay messages to meet $\xi^d$) depends on the time $t_{d_{wait},i}$. This accounts for the time that $C_i$ spends using $u[k] = K_1 x[k-1]$ 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 $t_{d_{wait},i}$ and $t_{d_{wait},i}$ is illustrated in Fig. 1 (explained in Section III). The slope $\beta_i$ can closely be approximated as the ratio between the response time $\xi^{TT}$ of a purely TT (i.e., zero-delay) controller and the response time $\xi^{ET}$ of a purely TT (i.e., delayed) controller. Hence, $t_{d_{wait},i}$ is given by $t_{d_{wait},i} = \xi^{TT} - \xi^{ET}$. Since $\xi^{TT}$ is strictly less than $\xi^{ET}$, $\beta_i < 1$ holds. Note that a $t_{d_{wait},i} = \xi^{TT}$ results in $t_{d_{wait},i} = 0$; however, this also implies a deadline violation as shown in Fig. 4.

### Table I: Illustrative Example

<table>
<thead>
<tr>
<th>Case</th>
<th>$t_{wait,i}$ (samples)</th>
<th>$t_{d_{wait},i}$ (samples)</th>
<th>$\xi_i$ (samples)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>13</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>12</td>
<td>17</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>11</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>13</td>
<td>10</td>
<td>21</td>
</tr>
</tbody>
</table>

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 $\xi^d$ because there will be no contention for the static segment. This leads to a poor overall bus utilization and an expensive design. Since $\xi^{TT} < \xi^d$ normally holds, applications can tolerate some contention $\xi^d - \xi^{TT}$ 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 $\xi^{TT}$, $\xi^{ET}$ and $\xi^d$ 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 $\xi^d$. 

### 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,

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

We consider the case where $u[k] = K_1 x[k-1]$ is applied for $t_{d_{wait},i}$ time units after the occurrence of disturbance, and subsequently, the controller switches to $u[k] = K_{opt}[x[k]]$ and is applied for $t_{d_{delay},i}$ time units to bring back the application to a steady-state. Table I shows various cases with different $t_{d_{wait},i}$ and the corresponding $t_{d_{delay},i}$ and $\xi_i$ for the discrete-time plant (2) with initial conditions $x[0] = x[2][0] = 20$ and $E_{th} = 0.1$. It may be noticed that $t_{d_{delay},i}$ decreases with an increase in $t_{d_{wait},i}$ and their relation can closely be approximated as:

$$t_{d_{delay},i} = \xi^{TT}_i - \beta_i t_{d_{wait},i}.$$  (3)

Eq. (3) can be interpreted as follows: A fraction of the disturbance is already rejected by $u[k] = K_1 x[k-1]$ during $t_{d_{wait},i}$ and hence $u[k] = K_{opt}[x[k]]$ needs less time to bring back the system to steady-state (i.e., shorter $t_{d_{delay},i}$).

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

$$\xi_i = t_{d_{wait},i} + t_{d_{delay},i},$$  (4)

$$= \xi^{TT}_i + (1 - \beta_i) t_{d_{wait},i}.$$  (4)

Based on the fact that $\beta_i < 1$, we can notice that $\xi_i$ increases with an increase in $t_{d_{wait},i}$ which is also supported by various cases shown in Table I. Thus, there is an upper-bound of $t_{d_{wait},i}$ to meet a given deadline $\xi^d$. As
per (4), the upper-bound on $t_{\text{wait},i}$ is longer for a shorter $t_{\text{dw},i}$. That is, the control applications are allowed to spend more time in the ET mode for a shorter $t_{\text{dw},i}$. Naturally, a shorter $t_{\text{dw},i}$ 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 $u[k] = K_i x[k] - 1$ to $u[k] = K_{\text{opt}} x[k]$, it stays in TT mode for $t_{\text{dw},i}$ time units. Here, $t_{\text{dw},i}$ 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., $\xi_i$. During $\xi_i$, the controller switches only once after $t_{\text{wait},i}$ 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 $t_{\text{dw},i}$ for a control application. First, $t_{\text{dw},i}$ can be chosen the maximum time required by the zero-delay controller to reject a disturbance after switching, for all $t_{\text{wait},i}$. For example, we can choose $t_{\text{dw},i} = 14$ samples as in Case 1 (Table I) for all the cases. The schedulability analysis with such $t_{\text{dw},i}$ will certainly provide a safe result. However, the actual $t_{\text{dw},i}$ is shorter than 14 samples for the Cases 2-5. Hence, the second possibility is to choose the actual $t_{\text{dw},i}$ as per (3) to avoid pessimism. In this work, we address the schedulability analysis problem described in the previous section considering the actual $t_{\text{dw},i}$.

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 $t_{\text{dw},i} \leq \xi_i^T$. $t_{\text{dw},i}$ 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 $\xi_i^T$.

A request for zero-delay communication $t_{\text{dw},i}$ 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 $\xi_i^d$ time units. For this purpose, a $C_i$ has to send $\frac{t_{\text{dw},i}}{r_i}$ zero-delay messages. However, as discussed previously, $t_{\text{dw},i}$ varies with the time $t_{\text{wait},i}$ 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 $\xi_i^d$. 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 $t_{\text{dw},i}$ amount of time (i.e., until it finishes transmitting $\frac{t_{\text{dw},i}}{r_i}$ 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 $t_{\text{wait},i}$ of $C_i$. Hence, its demand for zero-delay communication $t_{\text{dw},i}$ decreases as discussed before—see Fig. 1. However, as discussed before, the overall response time of the application increases with $t_{\text{wait},i}$ (see Eq. (4)). This is because the parameter $\beta = \frac{t_{\text{wait}}}{t_{\text{wait},i}}$ 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 $t_{\text{wait},i}$: $\xi_i^d \geq (1 - \beta) t_{\text{wait},i}$. Hence, to test the schedulability of $C_i$, we need to find the greatest possible $t_{\text{wait},i}$ (denoted by $t_{\text{wait},i}$) which leads to the worst-case response time of $C_i$ (denoted by $\xi_i$). 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., $t_{\text{dw},j} = \xi_j^T$. This assumption is pessimistic since $t_{\text{dw},j}$ 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 $t_{\text{wait,}i}$ and $t_i$ 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 $t_{\text{max,}i}$ 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 $t_{\text{max,}i} \leq r_i$ holds for all $C_i$; in this paper, i.e., there is only one transmission of $\frac{c_i}{p_i}$ messages of $C_i$ within its busy period $t_{\text{max,}i}$. This way, we only need to compute the response time $t_i$ of the sole job of $C_i$ within $t_{\text{max,}i}$ to obtain its worst-case response time $\hat{t}_i$, which can be done in the following manner:

$$\hat{t}_i = t_i + (1 - \beta_i)b_i + 1 - \beta_i \sum_{j=1}^{i-1} \left[ \frac{t_i}{r_j} \right] \xi^T_j,$$

where $b_i = \max_{n=1}^{n}\left(\xi^T_k\right)$ 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 $\hat{t}_i = t_i + (1 - \beta_i)b_i$ and computing it iteratively until $\hat{t}_i$ becomes greater than $\xi^d$ or converges to a certain value. Clearly, if $\hat{t}_i$ exceeds $\xi^d$, $C_i$ is not schedulable on the shared slot. On the other hand, if there is a convergence value prior to $\xi^d$, 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 $\xi^T_j$. Hence, in a non-preemptive scheme, a higher-priority $C_i$ 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 $\xi_i$ by $\xi^d$ (i.e., the deadline) in Eq. (5) and then resolving for $b_i$. The resulting value is $\hat{b}_i$:

$$\hat{b}_i = \frac{\xi^d}{c_i} - \frac{\xi^T}{c_i} - \sum_{j=1}^{i-1} \left[ \frac{\xi^d}{r_j} \right] \xi^T_j.\quad (6)$$

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 $b_i$ from the lowest- to the highest-priority application, we can then calculate $t_i$ in the following manner:

$$t_i = \hat{b}_i - \max_{k=1}^{n} (\xi^T_k - \beta_k t_k),\quad (7)$$

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:

$$b'_i = \max_{k=1}^{n} (\xi^T_k - \beta_k t_k).\quad (8)$$

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.

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 $\xi^d_i$). 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). $C_i$ is schedulable on one slot if it can meet its timing requirement $\xi^d_i$ 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

---

**Algorithm 1** Computation of the number of slots

```
Require: Set of control applications $C_i$ with $\xi^d_i$ and $\xi^T_i$
Require: The minimum disturbance inter-arrival time $r_i$ for every $C_i$
1: number_slots=1
2: $C_i$ 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
```
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: \{C1, C3\}, \{C4, C2\}, \{C6\} and \{C5\}. 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: \{C1, C3, C2\}, \{C4, C6\} and \{C5\}. 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 $\xi_i(b_i')$, i.e., $\xi_i$ 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 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