Schedulability Analysis of Distributed Cyber-Physical Applications on Mixed Time-/Event-Triggered Bus Architectures with Retransmissions

Alejandro Masrur*, Dip Goswami*, Reinhard Schneider*, Harald Voit*,
Anuradha Annaswamy† and Samarjit Chakraborty*
*TU Munich, Germany
†Massachusetts Institute of Technology, USA

***Abstract*—In this paper we study the setup where multiple cyber-physical applications are partitioned and mapped onto spatially distributed electronic control units (ECUs). Further, applications communicate over a mixed time-/event-triggered bus like FlexRay. Such a setting commonly arises in automotive and other distributed cyber-physical systems. All control messages mapped onto the time-triggered or static segment of the bus result in *negligible/zero* communication delays (viz., the bus and the ECUs can be perfectly synchronized) and hence good control performance. At the other extreme, *all* messages scheduled in the priority-driven dynamic segment often result in poor control performance because of the intrinsic timing non-determinism of priority-based protocols. In this paper we are concerned with the intermediate case – where messages are dynamically moved between the time- and event-triggered segments in order to meet performance requirements in the presence of disturbances – and formally study the *schedulability analysis* problem on the bus. In particular, we propose a novel scheduling strategy that considerably reduces the number of static time-triggered slots required in such a switching scheme to meet specified performance requirements. The basic premise of our work is that time-triggered slots are *expensive* and, hence, they should be used sparingly. We further demonstrate the benefits of our proposed scheme through a number of illustrative examples.**

I. Introduction

Distributed cyber-physical architectures typically consist of multiple control applications mapped onto spatially distributed processors or electronic control units (ECUs). In this paper we are concerned with the case where these ECUs communicate over a mixed time-/event-triggered bus such as FlexRay [1]. When all control messages are mapped onto the static time-triggered segment of the bus, the time-triggered slots and the ECUs may be perfectly synchronized. This results in *zero* communication delays assuming that the actual *transmission times* of messages are negligible. Clearly, this leads to in a *semantic match* between the control models and their implementations and thereby also good control performance. However, it is widely believed that as application complexity and hence communication requirements continue to grow, the bandwidth of the time-triggered segment (in buses like FlexRay) will not suffice and a *purely* time-triggered implementation might be overly expensive. On the other hand, priority-driven event-triggered 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. As a result, a large *semantic gap* opens up between control models and their implementations over the event-triggered segment, which in the end translates into poor control performance.

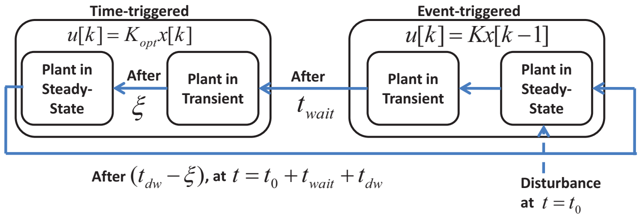

Fig. 1. Switching scheme

In this paper we investigate an intermediate possibility where the aim is to achieve control performance close to a purely time-triggered implementation, but using *fewer* time-triggered slots than what would be normally necessary. This is achieved by exploiting the following two observations: (i) the *settling time* of a controller (which is a widely used measure of control performance) is more susceptible to deterioration during its *transient phase*, i.e., when there are external disturbances, (ii) it is possible to find an upper bound on the frequency at which disturbances occur to the control applications. As a result, an event-triggered implementation might suffice when the applications are in *steady-state*. When they move to a transient phase because of an external disturbance, we propose to switch the relevant control messages to a time-triggered slot in order to minimize the response or settling time. This is illustrated in Fig. 1. The parameters in this figure are explained later.

In order to guarantee pre-specified control performance in the above scheme, a *schedulability analysis* is necessary, since the number of allocated time-triggered 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 time-triggered slot is occupied or not) before it may switch from an event-triggered to a time-triggered 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 [2] and the demand-bound criteria [3]. In this paper, we lift the classical response time analysis technique to a control-theoretic setting. Towards this we switch the control scheme (in particular, the controller gain values) when we move the associated control messages from the event- to the time-triggered scheme. This, along with the system model, determines the *dwell* *time* (t_{dw} in Fig. 1) for each application (considering all possible external disturbances and initial system states). The dwell time dictates the amount of time an application has to spend in the time-triggered mode in the event of an external disturbance. Given specified *settling times* (our measure of control performance), a mapping of messages to time-triggered slots and upper bounds on the arrival of disturbances for each control application, our analysis determines whether the control performance objectives of all applications may be satisfied. In addition, we reduce the number of time-triggered slots for a set of applications to be schedulable (in a control-theoretic sense).

There has been a considerable amount of work on schedulability analysis of both time-triggered [4] and a mix of time- and event-triggered systems [5]. But the questions addressed were typically: how to compute upper bounds on communication delays, and how to synthesize time-triggered schedules (see also [6] for time-triggered schedule synthesis for FlexRay). In the specific context of hybrid time- and event-triggered schedules, the focus has been on partitioning system functionality into time- and event-triggered activities. However, the schedulability problem arising in the context of dynamically switching messages between time- and event-triggered modes, and in particular the schedulability with control performance objectives has not been sufficiently addressed so far. Notable exceptions to this are [7] and [8]. The work presented in [7] studied how the performance of multiple control loops may be optimized while still ensuring schedulability in CAN networks. Similarly, the schedulability region that guarantees control performance has been computed in [8]. Our approach follows this line of work and specifically addresses the scheduling problem to minimize the required number of time-triggered slots while maintaining the desired response times for a set of control applications.

Organization: The rest of this paper is organized as follows. We first discuss the details of our system model along with some examples to motivate our setup (Section II). We formally formulate the problem in Section III. 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. Motivational Background

We consider a discrete-time control application of the form shown in (1) with constant sampling interval p. $x[k]$ is the $n \times 1$ vector of *state variables* and $u[k]$ is the *control input*. A is an $n \times n$ system matrix, B is an $n \times 1$ vector and we assume that (A, B) is a *controllable* pair. In this section, we intend to capture some motivational facts using an example of a second-order plant given by (2) with $p = 20ms$:

$$x[k+1] = Ax[k] + Bu[k], \quad (1)$$

$$A = \begin{bmatrix} 0 & 1.0 \\ -0.56 & -1.9 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 1.0 \end{bmatrix}. \quad (2)$$

The choice of $u[k]$ is what typically a control engineer is interested in to ensure a stable *regulation*:

$$x[k] \rightarrow \textit{reference} \text{ as } k \rightarrow \infty.$$

On top of ensuring a stable system regulation, a control design aims to bring $x[k]$ close to the reference within a finite amount of time (which is known as *settling time* ξ). At a high level, the settling time essentially indicates the response speed of the control application, i.e., response time[1].

In a feedback control system, $u[k]$ utilizes the system states $x[k]$ as *feedback*. In this work, we consider a *state-feedback controller*:

$$u[k] = Kx[k - \Delta], \quad (3)$$

where K is the *state-feedback gain* [9] and Δ is the feedback delay measured in number of samples. For example, considering a sampling period $p = 20ms$, $\Delta = 2$ implies a delay of $40ms$. Designing a state-feedback controller $u[k]$ essentially boils down to a problem of finding suitable controller gains K such that $x[k]$ reaches a specified proximity of the reference (e.g., 1%) within a given time, i.e., a desired settling/response time.

Case I - Control with zero feedback delay ($\Delta = 0$)**:** In this case, $u[k] = Kx[k]$, $x_1[0] = 20$, $x_2[0] = 15$. Without loss of generality, let us assume that the reference is *zero*. Given such a system where all the states $x[k]$ are *measurable*, it is possible to adapt well-known optimal control approaches such as the Linear Quadratic Regulator (LQR) [10] to derive the optimal feedback gain K_{opt}. Fig. 2 shows how the state $x_1[k]$ evolves with $u[k] = K_{opt}x[k]$. We can see that $x_1[k]$ (and similarly, $x_2[k]$) reaches very close to the reference after 4 samples, i.e., $4 \times$ 20ms= $80ms$. Hence, the settling time $\xi = 80ms$, i.e., the system response time is $80ms$ for the given initial conditions ($x_1[0] = 20$ and $x_2[0] = 15$).

Case II - Control with feedback delay ($\Delta = 1$)**:** Here, $u[k] = Kx[k-1]$ and we consider identical initial conditions as in Case I. Unlike the previous case, the feedback control $u[k]$ does not have the values of the current state $x[k]$. $u[k]$ has rather an older state $x[k-1]$ as feedback. Because of such restriction in the feedback, the choice of K cannot be optimized using standard design techniques such as LQR. We choose $K = \begin{bmatrix} -0.2540 & -0.6433 \end{bmatrix}$ by *pole placement* technique [10]. Fig. 3 shows how $x_1[k]$ evolves with $u[k] = Kx[k-1]$. We can see that the settling time is $\xi = 700ms$ or 35 samples, i.e., the system response becomes slower with delayed feedback.

The main motivation of this paper from the control theoretic perspective is that it is possible to regulate the response time ξ of the control application by appropriately switching the controllers $u[k] = K_{opt}x[k]$ and $u[k] = Kx[k-1]$. We support the above observations using the following example.

[1]The settling time (in the control theory parlance) and the response time (in the real-time systems parlance) are used interchangeably.

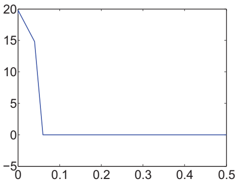

Fig. 2. Case I

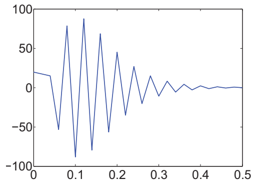

Fig. 3. Case II

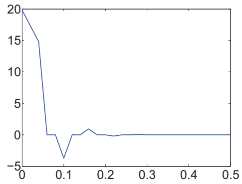

Fig. 4. Case III

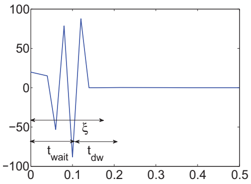

Fig. 5. Case IV

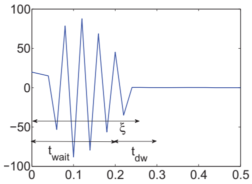

Fig. 6. Case V

Case III: In this case, we first use $u[k] = K_{opt}x[k]$ for the first two samples and then switch to $u[k] = Kx[k-1]$ from the third sample onwards. We can see that the settling time is now $\xi = 300ms$ or 15 samples (Fig. 4). Therefore, the system response can be adjusted by appropriately switching between the cases where $\Delta = 0$ and $\Delta = 1$, i.e., between the time- and the event-triggered message communication protocol. This switching between communication protocols is also associated with a corresponding switching of controllers.

A. *Concepts and Definitions*

Let us define the following terminologies to facilitate the problem statement.

Definition 2.1: The system (1) is said to be in a transient *phase if the system states are such that $x[k]^T x[k] > E_{th}$ where E_{th} indicates certain specified energy threshold of the system. Similarly, the system (1) is said to be in* steady-state *if the systems states are such that $x[k]^T x[k] \leq E_{th}$.*

The state of a system is governed by values of the *state variable* vector $x[k]$. In steady-state, the values of every element of the vector $x[k]$ should be small (or close enough to the reference). Therefore, $x[k]^T x[k]$ often acts as a measure of the system state or energy level of the system. The value of energy threshold indicates how much deviation from the reference is tolerated by the designer in steady-state. If the deviation is more than the energy threshold, the system is considered to be in transient phase. The occurrence of a transient phase can either be initiated by an external *disturbance* (e.g., the norm of a certain state $\|x[k]\|$ suddenly becomes too large) or by initial conditions. In this work, anything that causes a *transient* phase is referred to as a *disturbance*. In model (1), the disturbance is not shown explicitly for the sake of simplicity. However, the model can be modified to take *disturbances* into account at any sampling instant:

$$x[k+1] = Ax[k] + Bu[k] + D[k], \tag{4}$$

where $D[k]$ denotes external disturbances. $D[k] = 0$ for most of the samples k. $D[k]$ becomes non-zero at the occurrence of a disturbance, e.g., $D[12] = \begin{bmatrix} 0 \\ 10.0 \end{bmatrix}$ indicates that a disturbance of magnitude 10 occurs in the state $x_2[k]$ of our system at the twelfth sample (i.e., $k = 12$).

Definition 2.2: The maximum time required by the optimal controller $u[k] = K_{opt}x[k]$ to bring the system (1) to the steady-state from any possible transient phase (or initial states x[0]) is called dwell time *and will be denoted by t_{dw} in this paper.*

In a discrete-time representation, the time is computed in terms of samples, i.e., the absolute time is always a multiple of samples. Therefore, t_{dw} is always a multiple of the sampling interval p, e.g., if $u[k] = K_{opt}x[k]$ takes 5 samples to reject a disturbance then $t_{dw} = 5 \times p$. The set of all possible initial states $x[0]$ is usually known in real-life applications. Note that a disturbance changes the system states and it effectively starts from a new initial state after the occurrence of disturbance. Knowledge of all possible initial states essentially covers the initial state at the worst-case disturbance. The worst-case disturbance is usually known to the designer, e.g., if the current/voltage is one state then worst-case disturbance is the maximum/minimum voltage/current available from the source. Based on this assumption, we compute t_{dw} taking all these initial states into account. Hence, the optimal controller $u[k] = K_{opt}x[k]$ brings the system back to steady-state from *any* such initial condition in t_{dw} time. The application of $u[k] = K_{opt}x[k]$ for a time interval $t < t_{dw}$ does not necessarily guarantee a system recovery (i.e., a transition from transient to steady-state) within a *desired response time* ξ^d. The use of $u[k] = K_{opt}x[k]$ can wait $t_{wait} = (\xi^d - t_{dw})$ time units after the initiation of the transient phase without violating the response time requirement. For example, in model (1), let us assume that the system starts with a transient phase, $t_{dw} = 100ms$ (5 samples) and $\xi^d = 200ms$. Intuitively, the system response is identical to Fig. 2 when $t_{wait} = 0$, i.e., $u[k] = K_{opt}x[k]$ for the first 5 samples and $u[k] = Kx[k-1]$ from the sixth sample onwards.

Case IV: Let us assume that the waiting time is $t_{wait} = 100ms$, i.e., the system runs with $u[k] = Kx[k-1]$ for the first $100ms$ or 5 samples and the control law changes to $u[k] = K_{opt}x[k]$ from the sixth sample. In this case, it can be seen from Fig. 5 that $\xi = 180ms < \xi^d$.

Case V: Let us assume that the waiting time is $t_{wait} = 200ms$. Here it can be seen from Fig. 6 that $\xi = 260ms > \xi^d$.

From the above two cases, we observe that the response time ξ of an application changes based on t_{wait}. Whether an application is going to meet its response time requirement

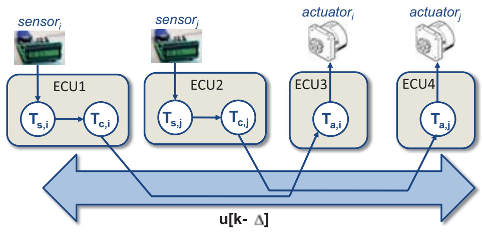

Fig. 7. The distributed cyber-physical architecture in this paper

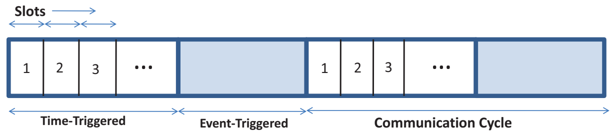

Fig. 8. Hybrid communication protocol: *time-triggered* and *event-triggered*

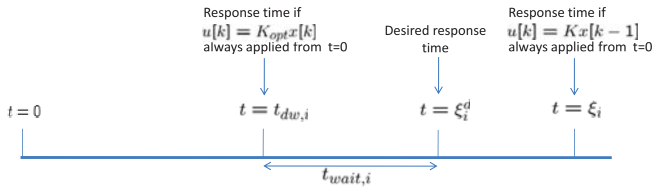

Fig. 9. Relation between $t_{dw,i}$, ξ_i^d and $t_{wait,i}$

or not depends on t_{wait}. In Case IV, t_{wait} is less than the maximum allowed waiting time before $u[k] = K_{opt}x[k]$ is applied to meet the response time requirement ξ^d. In Case V, t_{wait} is more than the maximum allowed waiting time and hence $\xi > \xi^d$. Based on this observation, we formulate the problem addressed in this paper.

III. Problem Formulation

We consider multiple control applications of the form of (1). Such applications are denoted by C_i with sampling period p_i ($i \in \{1, 2 \ldots n\}$) and run on a distributed architecture of the form shown Fig. 7. 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). Such tasks are then mapped onto spatially distributed ECUs which are connected via a shared communication bus.

As an underlying communication medium, we consider a hybrid communication protocol (e.g., FlexRay) as shown in Fig. 8 where each communication cycle is divided into time-triggered (or static) and event-triggered (or dynamic) segments. On the time-triggered 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 *event-triggered* segment. 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 trivially extends 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 equal period (which is the sampling time p_i of the application C_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 μs) are negligible compared to the sampling period p_i (in the order 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. The maximum delay experienced by the control related messages can be computed by the traditional worst-case response time analysis [11], [12]. We assume that the priority assigned to every $T_{c,i}$ sending over the dynamic segment guarantees that $0 < \tau_i \leq p_i$ holds for the corresponding C_i. From the above properties, it is clear that the controller design in Case II is essentially based on the worst-case delay. The performance of such controller design is often pessimistic as we can see in Case II. By switching between the designs with zero-delay and worst-case delay, we can avoid the pessimism coming from the design based on the worst-case delay (discussed in the following paragraphs).

Now, we try to match the above distributed setup with the control scenario described in the previous section (Sec. II). More specifically, if a task $T_{c,i}$ sends a message over the static segment, the communication delay is zero and we will be able to implement the optimal control strategy $u[k] = K_{opt}x[k]$ (Case I). In this work, we neglect the effect of transmission time for messages and consider $\Delta = 0$ under a *time-triggered* communication scheme. On the other hand, if a $T_{c,i}$ transmits over the dynamic segment, we can implement the controller of the form $u[k] = Kx[k-1]$ since the communication delay here can be as much as a sampling period p_i (Case II).

Based on the above discussion, we consider the following sequence of events (illustrated in Fig. 1 and 9):

- A control application C_i can either apply $u[k] = K_{opt}x[k]$ or $u[k] = Kx[k-1]$. In both cases, the asymptotic stability is guaranteed, i.e., $x[k] \rightarrow$ *reference* as $k \rightarrow \infty$.
- The implementation of $u[k] = K_{opt}x[k]$ needs zero-delay feedback, which can only be achieved using the static segment for sending the control-related messages (from $T_{c,i}$). On the other hand, $u[k] = Kx[k-1]$ can be implemented using the dynamic segment.

- The system response time ξ is lower in the case of using $u[k] = K_{opt}x[k]$ and higher with $u[k] = Kx[k-1]$.
- Every control application is associated with a desired response time ξ_i^d within which it must get back to steady-state after the occurrence of any disturbance.
- 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 $\frac{t_{dw,i}}{p_i}$ consecutive messages with zero delay. The application of only $u[k] = Kx[k-1]$ causes a violation of the response time requirement ξ_i^d.
- For a given C_i, the transmission of $\frac{t_{dw,i}}{p_i}$ consecutive zero-delay messages can wait at most for $t_{wait,i} = (\xi_i^d - t_{dw,i})$.

Switched control/communication scheme (Fig. 1): In the context of the above setup, the proposed switching scheme behaves as follows:

- **(i)** A control application is running in steady-state and utilizes $u[k] = Kx[k-1]$ implemented over the dynamic segment.
- **(ii)** A disturbance occurs at $t = t_0$ which changes the system state to transient phase.
- **(iii)** The control application may continue applying $u[k] = Kx[k-1]$ for at most $t_{wait,i}$ time.
- **(iv)** At the latest at $t = t_0 + t_{wait,i}$, the controller and the communication are *switched* to $u[k] = K_{opt}x[k]$ and the static segment respectively.
- **(v)** The control application keeps on applying $u[k] = K_{opt}x[k]$ for another $t_{dw,i}$ time.
- **(vi)** At $t = t_0 + t_{wait,i} + t_{dw,i}$ (the latest), the controller and the communication are *switched* back to $u[k] = Kx[k-1]$ and the dynamic segment respectively. The control application C_i remains in this configuration until another disturbance occurs.

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. Hence, we propose allocating multiple applications to the same time-triggered 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 $t_{dw,i}$ and ξ_i^d ($i \in \{1, 2 \ldots n\}$). 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.

IV. Number of Static Segment Slots

The computation of the number of slots on the static or time-triggered segment consists of two interlocked steps:

- The schedulability analysis on one static segment slot shared by multiple control applications.

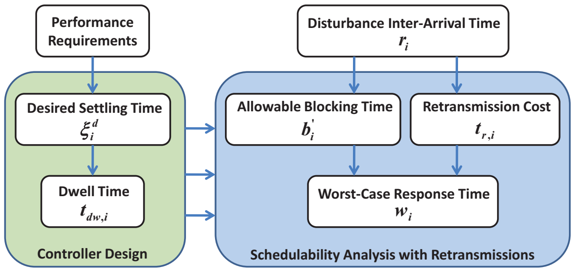

Fig. 10. Control/scheduling co-design

- The allocation of applications to one or more static segment slots, which is based on the schedulability analysis for one slot.

A. Schedulability Analysis

In this section, we first analyze the schedulability of control messages on one shared time-triggered slot according the switched control/communication scheme described in the previous section. As shown in Fig. 10, the schedulability analysis requires two inputs: **(i)** the performance-related requirements derived from the control design, **(ii)** the disturbance arrival pattern.

In principle, a time-triggered 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 time-triggered slot or processor. At the occurrence of a disturbance, a control application requests access to zero-delay transmission for a given amount of time $t_{dw,i}$. $t_{dw,i}$ here behaves as the *execution time* of C_i. A time-triggered slot or processor must provide $t_{dw,i}$ amount of *uninterrupted service* to a task C_i within ξ_i^d, which acts as a *deadline* for C_i.

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 $\frac{t_{dw,i}}{p_i}$ consecutive zero-delay messages (i.e., it requires uninterrupted access to the time-triggered slot for at least $t_{dw,i}$ time units).

In order to schedule a number of control applications C_i on the same static segment/time-triggered slot, we propose 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 [13], 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.

Considering retransmissions: Although a communication bus is intrinsically non-preemptive, the setting studied in this paper allows aborting the transmission of a sequence of lower-priority messages and hence reducing the blocking time suffered by higher-priority applications. This is possible because, in every communication cycle, we can decide again which of the applications sharing a slot may have access to it and start transmitting next.

On the other hand, as discussed previously, an application C_i needs to transmit at least $\frac{t_{dw,i}}{p_i}$ consecutive zero-delay messages to guarantee control performance requirements. Thus, if an ongoing transmission sequence is canceled, we will need to retransmit the whole sequence of C_i's messages (irrespective of how many C_i's messages could have been transmitted previously). This results in retransmission cost which needs to be considered in the schedulability analysis.

To find a balance between blocking time and retransmission cost, a higher-priority C_j is only allowed to interrupt a lower-priority C_i after a configurable blocking time b'_j. This is similar to implementing a limited-preemption scheme [14]. However, our analysis differs from the known techniques in a non-trivial manner, since we also consider the effect of retransmitting messages.

In what follows, we denote by w_i *the worst-case response time* of a control application C_i. That is, the maximum time that it takes C_i to finish transmitting $\frac{t_{dw,i}}{p_i}$ consecutive messages over the shared slot. If w_i is less than or equal to the desired system response ξ_i^d, C_i is going to be schedulable on the given shared slot.

Since a C_i can be blocked by a lower-priority application, computing w_i here has some similarities with computing the worst-case response time in a fixed-priority non-preemptive scheduling like the one of CAN [11], [12]. To find the worst-case response time of a task under a fixed-priority non-preemptive scheduling, we need to compute the response times of all *jobs* of that task within its *maximum busy period* [12].

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 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. C_i's maximum busy period denoted by $t_{max,i}$ results when all higher-priority control applications (that share the same slot) require sending their message sequences at the same time and can be computed as follows:

$$t_{max,i} = b_i + \left\lceil \frac{t_{max,i}}{r_i} \right\rceil t_{dw,i} + \sum_{C_j} \left\lceil \frac{t_{max,i}}{r_j} \right\rceil t_{dw,j}, \qquad (5)$$

where $C_j \in HP(i)$ and $HP(i)$ denotes the subset of control applications with higher priority than C_i (i.e., for every C_j in $HP(i)$, $\xi_j^d \leq \xi_i^d$ must hold under the DM policy). b_i is the maximum blocking time suffered by C_i due to lower priority applications. (5) can also be solved in an iterative manner starting from $t_{max,i}^{(1)} = t_{dw,i} + b_i$ and proceeding until $t_{max,i}^{(\kappa+1)} = t_{max,i}^{(\kappa)}$ holds where κ is an integer number indicating the iteration step. The resulting value is C_i's maximum busy period.

In this paper, we consider that $t_{max,i} \leq r_i$ holds, i.e., there is only one transmission of $\frac{t_{dw,i}}{p_i}$ messages of C_i within its busy period $t_{max,i}$. So we can compute w_i in the following manner:

$$w_i = b_i + t_{dw,i} + \sum_{C_j} \left\lceil \frac{w_i}{r_j} \right\rceil t_{dw,j}, \qquad (6)$$

where $C_j \in HP(i)$ and again $HP(i)$ is the set of all higher-priority control applications. (6) can be solved iteratively starting from $w_i^{(1)} = t_{dw,i} + b_i$ and proceeding as above.

Now, we first compute the maximum admissible blocking time $\hat{b}_i$ of a control application C_i. $\hat{b}_i$ is the blocking time for which the worst-case response time of a C_i is equal to its deadline, i.e., $w_i = \xi_i^d$ holds. Using (6), we can obtain a $\hat{b}_i$:

$$\hat{b}_i = \xi_i^d - t_{dw,i} - \sum_{C_j} \left\lceil \frac{\xi_i^d}{r_j} \right\rceil t_{dw,j}. \qquad (7)$$

Now, for every control application C_i, we *configure* a blocking time b'_i such that $b'_i \leq \hat{b}_i$ holds. This means that C_i can be blocked by a lower-priority application for at most b'_i time units after which it cancels the transmission of any lower-priority application.

In the same way, a higher-priority C_j can cancel the transmission of C_i after b'_j time units. Hence, C_i may have to retransmit a certain number of messages and incurs in a retransmission cost that is given by b'_j (i.e., the *configured* blocking time of the higher-priority C_j). Notice that if $b'_j \geq t_{dw,i}$ holds, C_i has enough time to finish sending its message sequence before b'_j expires (and it will not incur in retransmission). Further, to obtain the maximum retransmission cost $t_{r,i}$ for a C_i, we need to consider the fact that C_i's transmission can be canceled by any higher-priority C_j:

$$t_{r,i} = \max_{C_j, b'_j < t_{dw,i}} \left(b'_j \right), \qquad (8)$$

where $C_j \in HP(i)$ and $HP(i)$ is the set of all applications with higher-priority than C_i. Clearly, if $b'_j < t_{dw,i}$ does not hold for at least one C_j, $t_{r,i}$ will be zero.

Now, for a lower-priority C_i, if the transmission of any of its higher-priority C_j is interrupted by another higher-priority application, the retransmission cost of C_j needs to be considered as *blocking time* for C_i. In worst case, all higher-priority tasks of C_i may need retransmission. As a result, we can configure any positive blocking time for C_i that is at most equal to:

$$b'_i = \hat{b}_i - t_{r,i} - \sum_{C_j} t_{r,j}, \qquad (9)$$

where $C_j \in HP(i)$ and $HP(i)$ is the set of all control applications with higher-priority than C_i. Clearly, if no positive

Algorithm 1 Computation of the number of slots

Require: Set of control applications C_i with ξ_i^d and $t_{dw,i}$
Require: The minimum disturbance inter-arrival time r_i for every C_i

```
1:  slot_number=1
2:  Sort C_i according to ξ_i^d
3:  for i = 1 to n do
4:      for s = 1 to slot_number do
5:          if Schedulable(C_i,slot(s)) then
6:              Allocate C_i to slot(s)
7:          else if s==slot_number then
8:              slot_number = slot_number + 1
9:              Allocate C_i to slot(slot_number)
10:         end if
11:     end for
12: end for
13: Return slot_number
```

b_i' can be found, C_i cannot be scheduled. Furthermore, the schedulability on one slot can be guaranteed if a positive b_i' can be found for every C_i running on the slot. To this end, we need to compute b_i' in order of decreasing priorities from the highest to the lowest priority.

In the proposed scheme, the blocking time of a higher-priority C_j is the maximum time that C_j waits for a lower-priority C_i to free the shared slot. If more than one C_j are waiting for a C_i, C_i will be canceled after the minimum b_j' greater than zero among all *waiting* C_j. However, once this minimum b_j' elapses and C_i gets interrupted, the C_j with the highest priority prevails over the other waiting applications (independently of whether this has the minimum b_j' or not).

B. 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.

In this paper, we propose an algorithm based on the well-known First Fit (FF) heuristic, since it 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 urgency, i.e., 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 tries to allocate all C_i to one or more slots in the list of existing slots (line 4 to 11). It then adds a slot to the list (line 8), only if a C_i could not be scheduled on any of the exiting slots. The algorithm finishes when all C_i have been allocated and returns the number of slots that were necessary for accommodating all C_i (line 13).

TABLE I
CONTROL APPLICATIONS

C_i	$r_i(ms)$	$\xi_i^d(ms)$	$t_{dw,i}(ms)$
C_1	2000	300	100
C_2	2000	400	120
C_3	1500	450	150
C_4	2000	1000	300
C_5	5000	3000	800
C_6	500	500	50

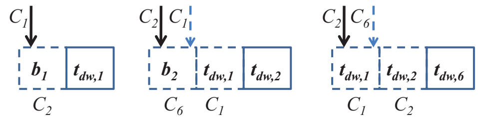

Fig. 11. Response times w_i under the non-preemptive scheme (for the applications allocated to the first slot) - $C_1 : w_1 = b_1 + t_{dw,1} = 220ms$, $C_2 : w_2 = b_2 + t_{dw,1} + t_{dw,2} = 270ms$ and $C_6 : w_6 = t_{dw,6} + t_{dw,1} + t_{dw,2} = 270ms$.

V. RESULTS AND EVALUATION

In this section, we evaluate the proposed switching scheme through an illustrative example. We consider six control applications with the parameters shown in Table I. All these applications are distributed as shown in Fig. 7. The communication protocol is assumed to be FlexRay with a cycle length of $5ms$. The static segment has $2ms$ length and it is divided into 10 slots. The rest of the cycle is assigned to the dynamic segment.

Non-preemptive scheduling: For the sake of comparison, we first utilize fixed-priority non-preemptive scheduling such as that of CAN for arbitrating the access to the shared time-triggered slots. We use the known schedulability analysis for non-preemptive scheduling [11], [12] in combination with Alg. 1 to determine the minimum number of slots that guarantee all response time requirements. For the considered example, we obtained three slots under the non-preemptive scheme: C_1, C_2 and C_6 are *schedulable* in one slot; C_3 and C_4 are schedulable in a second slot; and finally, C_5 needs a separate third slot.

Fig. 11 shows a graphical representation of the response times of C_1, C_2 and C_6 (i.e., the control applications that are allocated to the first slot) in the worst case. This results from considering that for every application the maximum blocking time may occur (e.g., C_1's worst-case response time occurs when C_1 is blocked by C_2 as shown in Fig. 11).

Our proposed slot sharing with retransmissions: The second option is the one presented in this paper, which consists of implementing a limited preemption on the shared slots and retransmissions as explained in Section IV. In this case, Alg. 1 leads to two slots (one slot less than in the non-preemptive case, but four slots less than in the case of a purely time-triggered scheme): C_1, C_2, C_3, C_4 and C_6 are now schedulable in one slot, whereas C_5 still needs a slot on its own. In the case of limited preemption with retransmissions, Fig. 12 illustrates the response times of the control applications assigned to the first slot (i.e., C_1, C_2, C_3, C_4 and C_6). This results from considering the

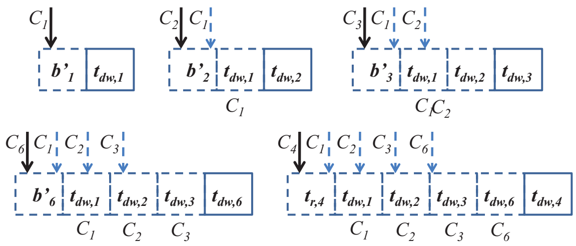

Fig. 12. Response times w_i under the limited preemptive scheme with retransmissions (for the applications allocated to the first slot) - $C_1 : w_1 = b'_1 + t_{dw,1} = 300ms$, $C_2 : w_2 = b'_2 + t_{dw,1} + t_{dw,2} = 400ms$, $C_3 : w_3 = b'_3 + t_{dw,1} + t_{dw,2} + t_{dw,3} = 450ms$, $C_6 : w_4 = b'_6 + t_{dw,1} + t_{dw,2} + t_{dw,3} + t_{dw,6} = 500ms$ and $C_4 : w_4 = t_{r,4} + t_{dw,1} + t_{dw,2} + t_{dw,3} + t_{dw,4} + t_{dw,6} = 920ms$.

blocking time obtained with (9) for every application. The maximum admissible blocking time of the highest-priority C_1 is $b'_1 = 200$. Since C_2, C_3 and C_6 can finish transmitting their sequences of messages within b'_1, they do not suffer from any retransmission cost as shown in Fig. 12. On the other hand, C_4 can start transmitting its messages when C_1 requires access to the slot. C_1 then decides to wait up to $200ms$ and preempts C_4, which has to retransmit its whole sequence of messages (since it cannot finish within b'_1). Thus, C_4 suffers from a *retransmission cost* of $200ms$.

Finally, Fig. 13 shows the *schedulability region* for the considered set of applications. For C_4, C_5 and C_6 given as in Table I, we vary the disturbance arrival rates of C_1, C_2 and C_3. As it can be noticed, for $r_1 = r_2 = 400ms$, the applications of Table I are only schedulable for an $r_3 = 650ms$. Further, an $r_3 = 400ms$ is then possible if we increase r_1 and r_2 to $600ms$ or more.

Discussion: We observe the following from the above results: **(i)** Under the limited-preemption scheme with retransmissions, the blocking time of higher-priority applications is reduced compared to the traditional non-preemptive scheme. On the other hand, the lower-priority applications incur into *retransmission delay* due to interruptions by higher-priority tasks. Since, with the DM scheduling, the lower-priority applications have longer response time requirements (i.e., deadlines), they normally tolerate this additional delay. In general, the limited-preemption scheme leads to a smaller number of slots and, hence, to a more efficient use of the bandwidth in the communication bus. **(ii)** As a consequence of (i), the limited-preemption scheme is more suitable when some of the control applications have lower $\frac{t_{dw,i}}{\xi_i^d}$ ratios, e.g., C_6 in the above example. The ratio $\frac{t_{dw,i}}{\xi_i^d}$ is a measure for the "communication demand" of a C_i (similar to the concept of *density* in the schedulability theory). We expect the non-preemptive and the limited-preemption schemes to have identical behavior for higher $\frac{t_{dw,i}}{\xi_i^d}$.

VI. Concluding Remarks

In this paper we proposed a scheduling strategy for distributed control applications, where control messages are dynamically switched between event- and time-triggered communication slots in response to external disturbances. In order to trade off between the response times of higher- and lower-priority applications, we proposed a limited-preemption scheme with retransmissions which reduces the number of time-triggered slots that are necessary. The novelty of our work stems from formulating the schedulability analysis problem in the context of a control-theoretic setting and also from our control-performance driven scheduling technique. As a part of future work, we plan to distinguish between different kinds of disturbances (rather than always assuming the *worst-case disturbance*) and extend our analysis to handle them in a conservative fashion.

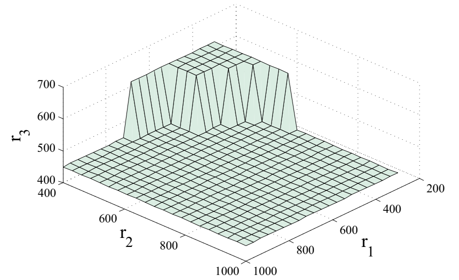

Fig. 13. Schedulability region under limited preemption and retransmissions

References

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

[2] 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.

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

[4] 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.

[5] T. Pop, P. Eles, and Z. Peng, "Design optimization of mixed time/event-triggered distributed embedded systems," in *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2003, pp. 83–89.

[6] M. Lukasiewycz, M. Glaß, J. Teich, and P. Milbredt, "Flexray schedule optimization of the static segment," in *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2009, pp. 363–372.

[7] A. Martinez and P. Tabuada, "On the benefits of relaxing the periodicity assumption for networked control systems over CAN," in *Proceedings of the Real-Time Systems Symposium (RTSS)*, 2009, pp. 3–12.

[8] F. Zhang, K. Szwaykowska, W. Wolf, and V. J. M. III, "Task scheduling for control oriented requirements for cyber-physical systems," in *Proceedings of the Real-Time Systems Symposium (RTSS)*, 2008, pp. 47–56.

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

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

[11] K. Tindell, H. Hansson, and A. Wellings, "Analysing real-time communications: Controller area network (CAN)," in *Proceedings of the Real-Time Systems Symposium (RTSS)*, December 1994, pp. 259–263.

[12] 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.

[13] J. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," *Performance Evaluation*, vol. 2, no. 4, pp. 237–250, 1982.

[14] S. Baruah, "The limited-preemption uniprocessor scheduling of sporadic task systems," in *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, 2005, pp. 137–144.