To appear in 41st IEEE Real-Time Systems Symposium (RTSS), 2020

GoodSpread: Criticality-Aware Static Scheduling of CPS with Multi-QoS Resources

Debayan Roy*, Sumana Ghosh*, Qi Zhu†, Marco Caccamo*, Samarjit Chakraborty‡
*Technical University of Munich, Germany, †Northwestern University, USA, ‡UNC Chapel Hill, USA
{debayan.roy, sumana.ghosh, mcaccamo}@tum.de, qzhu@northwestern.edu, samarjit@cs.unc.edu

***Abstract*—In practice, safety-critical cyber-physical systems (CPS) are often implemented using high quality-of-service (QoS) resources to provide maximum performance in all scenarios. Such implementations are oblivious to the changing criticality levels of CPS based on their physical dynamics (e.g., steady or transient state). Considering that high-QoS resources are constrained for cost-sensitive CPS, such criticality-oblivious implementations are highly inefficient. Towards a tighter dimensioning of these resources, state-of-the-art approaches have considered multi-QoS resources and studied criticality-aware dynamic resource allocation along the lines of mixed-criticality systems. However, these approaches have high implementation overheads. Moreover, in safety-critical domains like automotive and avionics, certification of such dynamic policies is challenging and the implementation platforms typically do not support dynamic reconfiguration. To address these challenges, we present *GoodSpread* that uses a static scheduling strategy and offers the same performance guarantees while saving resources (more than $50\,\%$ in certain cases) compared to the existing dynamic schemes. The main idea here is to spread the high-QoS resources as uniformly as possible over time in order to accommodate the uncertainty of when the criticality level might change. Our proposed strategy studies the physical dynamics to determine the *spread factor*, i.e., how often the high-QoS resources need to be provisioned. We further propose an extensibility-driven optimization approach to obtain a static schedule that will accommodate future workloads on the remaining resources with maximum flexibility.**

I. Introduction

State of practice: In cyber-physical systems (CPS), a physical process is typically controlled by software running on an electrical and electronic (E/E) platform. For safety-critical processes, the control software needs to provide performance guarantees. Besides the control algorithm, the control performance also depends on the platform resources (e.g., computation, communication, and memory resources) on which the software is implemented [1], [2]. The software implementation influences the control timings (e.g., sampling period and closed-loop delay) that have a huge impact on the process dynamics. In practice, to offer maximum performance in all scenarios, the safety-critical control software often uses resources with high quality-of-service (QoS) (e.g., time-triggered resources) [3].

Criticality levels of a CPS: Typically, a controlled process exhibits multiple criticality levels based on its physical dynamics [4]. For example, when the plant is in steady state, the control software is in a low-criticality mode where delayed or infrequent control actions will not jeopardize the system's safety. Conversely, when there is a disturbance, the system must reject it within a specified time limit, and therefore, the software must run in the high criticality mode with hard timing guarantees. Note that with changing criticality levels, the resource requirements of the control software also change.

Multi-QoS resources: In several CPS domains, E/E platforms comprise resources with different QoS [5]. For example, a processor can run time- or event-triggered scheduling schemes where the former gives precise timing information while for the latter, the response time of a software task might vary. Similarly, communication buses have different bandwidths, e.g., 500 kbit/s for CAN [6] and 10 Mbit/s for FlexRay [7]. Hybrid communication buses are also prevalent, e.g., FlexRay in automotive and Profinet in industry automation. In the same vein, using dedicated scratchpad for predictable and faster execution of software code has become a common practice to mitigate large variation in memory access times [8].

Criticality-aware dynamic scheduling: Previous works have shown that criticality-aware implementation of control applications using multi-QoS resources enables prudent usage of high-QoS resources [9]–[11]. This is important in cost-sensitive safety-critical domains like automotive where limited high-QoS resources are available but in high demand. In a criticality-aware implementation, the controller mostly uses low-QoS resources and switches to high-QoS resources only for a certain time while experiencing disturbances. Compared to the conservative implementation with only high-QoS resources, the criticality-aware implementation uses only a fraction of high-QoS resources to meet the control requirements.

Despite these approaches showing promise in saving high-QoS resources, they are not applied in practice. One possible reason is that they require dynamic resource allocation. In safety-critical domains like automotive, the underlying E/E platforms typically do not support dynamic reconfiguration [12], [13]. Moreover, certification of dynamic schemes is challenging [14]. They also have significant implementation overhead as an arbiter needs to run for monitoring the physical states of a set of controlled plants to decide their criticalities and, accordingly, allocating the resources [15].

Proposed scheduling approach: Although several dynamic scheduling policies have been studied so far for multi-QoS resources, the potential of static scheduling has not yet been explored. *In this paper, we propose a framework GoodSpread that statically allocates multi-QoS resources to a set of control applications so that each of them meets its performance requirements in all scenarios while using the minimum amount of high-QoS resources.* To demonstrate our proposed idea, we study the implementation of distributed control applications on FlexRay-based ECU networks. Here, the main reason is

that the previous works [9]–[11] also use this setup to study dynamic policies, and therefore, there are results available for comparison. FlexRay supports two different scheduling policies, i.e., time-division multiple access (TDMA) and flexible TDMA (FTDMA) [7]. Using TDMA slots, the closed-loop delay can be minimized to a negligible value while for FTDMA communication, the worst-case delay can be significantly high. Hence, the control performance will strongly depend on how the control data is sent on the bus.

The main technique proposed in this paper can also be applied to many different scenarios. For example, we can implement a control task using dual priorities. For a higher priority, the worst-case response time will be shorter, thus, resulting in a shorter closed-loop delay. Conversely, a lower priority will give a longer delay. In the same vein, we can prefetch the code for a control task into the scratchpad for a shorter execution time, and hence, a shorter closed-loop delay. Running the code from the main memory via cache can result in a significantly longer worst-case execution time.

In our problem setting, each instance of the control data can be sent using two different scheduling policies. Thus, there are exponential number of scheduling options. It is computationally challenging to evaluate all possible options to determine the one that uses the least number of TDMA slots to meet the performance requirements. However, *experiments show that a control application, along with its control performance objectives and disturbance arrival patterns, can be characterized by a spread factor. This factor quantifies how the TDMA slots should be spread over time in order to accommodate the uncertainty in how the criticality level of the system might change.* For a spread factor given by $\{n_h, \widehat{n}_h\}$, GoodSpread constructs a periodic schedule where n_h slots are distributed as uniformly as possible over $\widehat{n}_h$ samples. A similar intuition is also used to distribute drops in weakly-hard scheduling of control applications [16], [17]. Considering the characteristics of the FlexRay protocol, GoodSpread uses a polynomial-time algorithm that studies the closed-loop dynamics to derive the spread factor for an application.

Given the spread factor for each control application, GoodSpread formulates an optimization problem to determine a concrete FlexRay schedule for the applications. *As the main goal here is to prudently use the high-QoS resources, GoodSpread maximize the extensibility of the schedule, i.e., the obtained schedule offers the maximum flexibility in accommodating future frames on the remaining slots.* Towards this, we propose an iterative approach that incrementally adds prospective data frames to the set of control data frames and tries to co-schedule them. In each iteration, we find the prospective data frame with maximum possible scheduling demand, and if it is not schedulable, we reduce the scheduling demand for the next iteration. We use a similar notion of extensibility as introduced in [18], [19]. Here, the premise is that if a frame with a periodicity of r cycles can be scheduled, then two frames with a periodicity of $2r$ are also schedulable, however, the converse is not always true for FlexRay.

Our experiments suggest that the static scheduling approach in GoodSpread *saves TDMA slots (more than* $50\,\%$ *in certain cases)* compared to existing dynamic schemes [11]. Such a result is surprising considering that a dynamic policy is supposed to be more flexible. Typically, in real-time scheduling, the amount of processor time required by a task is assumed to be a constant. However, for control applications implemented using multi-QoS resources, there is a tight coupling between the amount of high-QoS resources required and the order in which the application gets the high-QoS resources. For example, allocating a TDMA slot every 4 control samples might be sufficient to reject a disturbance in 16 samples, however, when the application does not get a slot in the first 8 samples, it might require a slot in each of the next 8 samples to reject the disturbance. Existing dynamic scheduling schemes have only considered the worst-case requirement, and hence, over-provisioned high-QoS resources. Conversely, we allocate TDMA slots statically in the order in which their usage is minimized, and thus, our proposed strategy performs better than the existing dynamic schemes. As a future work, it would be interesting to devise a dynamic scheme following the intuition derived in this paper. However, the main focus here is static scheduling because of its practical relevance.

Related works: Our work mainly follows the idea introduced in [9], i.e., implementing a control application using multi-QoS resources leads to savings in high-QoS resources. Following this work, several dynamic scheduling strategies, e.g., [10], [11], have been proposed along the lines of mixed-criticality systems [20]. That is, when the controlled plant is in steady state (low-criticality mode), the controller uses the low-QoS resources, while during a disturbance (high-criticality mode), it tries to get high-QoS resources to reject the disturbance faster. In addition to having limited scope in practical safety-critical systems, these dynamic schemes have largely been conservative to accommodate for the worst-case switching sequence that may arise from arbitration. Conversely, a static scheduling strategy, which is the main focus of this work, is easier to implement and can tune the switching between resources offline based on the system dynamics.

Performance-aware static scheduling of control applications is extensively studied for both time- [21], [22] and event-triggered architectures [23], [24]. However, such a scheduling problem is not well-known for multi-QoS resources.

Furthermore, in control theory, there have been works on optimal control of switched systems [25], [26]. These works mostly consider theoretical performance metrics like quadratic cost [26] and H_∞ [25]. It is challenging to formulate a closed-form mathematical problem to minimize the usage of high-QoS resources while meeting the performance requirements.

In the context of multi-modal scheduling of control applications, several works have considered to switch between sampling periods online [27], [28] and offline [29]. Here, [29] uses genetic algorithm to determine feasible dual-mode schedules for a set of control applications. Although the motivation is similar, we consider a concrete multi-QoS platform, for which the approach used in [29] cannot be applied.

Another related research direction for this work is extensibility-driven scheduling. Different notions of extensibility exist in the literature, e.g., ability (or cost) to accommodate

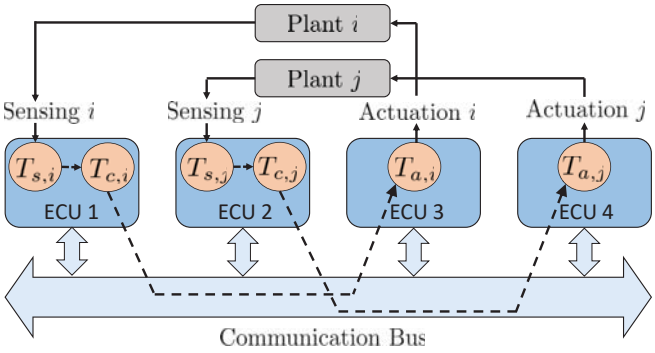

Fig. 1. Distributed control applications share a communication bus.

future workloads [18], [19], [30] or extension in existing workloads [31], [32]. We follow a similar notion of extensibility as in [18], [19]. While [18] uses bin packing heuristics to maximize the extensibility of the FlexRay schedules, [19] uses simulated annealing. In contrast to these heurtistic approaches, the proposed extensibility-driven schedule optimization technique guarantees optimality.

Contributions: This paper has the following contributions:

- We show that for control applications implemented using multi-QoS resources, established dynamic policies might be significantly inefficient compared to a static schedule, unlike popular beliefs. This is the main novelty of this work.
- We propose a multi-stage optimization framework *GoodSpread* to statically allocate high-QoS resources to control applications. In the first stage, we compute the spread factor for each application, i.e., how often the high-QoS resources must be provided for an application to meet the performance requirements in all scenarios. In the second stage, given the spread factors, we determine the most extensible schedule configuration for the applications.
- For spread factor computation, we propose a polynomial-time algorithm based on the intuition that it is resource-efficient to spread the high-QoS resources uniformly over time. This intuition is based on empirical evidence.
- For schedule synthesis, we propose an iterative method for optimization that guarantees maximum extensibility.

Paper organization: The rest of the paper is organized as follows. Sec. II describes the problem setting and derives the system model. In Sec. III, using a motivational example, we compare different scheduling schemes. Sec. IV outlines the proposed algorithm to derive the spread factor for a control application. The proposed extensibility-driven schedule optimization approach is explained in Sec. V. Experimental results are provided in Sec. VI, and we conclude in Sec. VII.

II. Problem Setting

We study a problem setting where a number of electronic control units (ECUs) communicate over a shared communication bus, as shown in Fig. 1. Multiple control applications, denoted by $\mathbb{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_n\}$, are implemented on this distributed E/E platform. Each application $\mathcal{C}_i$ comprises three tasks, i.e., $T_{s,i}$, $T_{c,i}$, and $T_{a,i}$. $T_{s,i}$ is the sensor task, $T_{c,i}$ is the controller task, and $T_{a,i}$ is the actuator task. $T_{s,i}$ and $T_{c,i}$ are mapped on the same ECU while $T_{a,i}$ is mapped on a different ECU. The control input is calculated by $T_{c,i}$ based on the sensor values read by $T_{s,i}$. The control data is then communicated over the shared bus to $T_{a,i}$ that will eventually actuate the plant. This is a common scenario in a CPS setting where sensors and actuators are spatially distributed. Note that our proposed technique can also be applied for different task partitioning and mapping schemes.

A. Heterogeneous Resources

We consider that a controller can be implemented using a mix of high- and low-QoS resources. Towards this, in our problem setting, the communication takes place over the hybrid FlexRay bus. FlexRay is commonly used in modern cars for data transmission by safety-critical functions like steering and brake control in the chassis domain [33]. According to the FlexRay protocol [7], each communication cycle comprises a static segment and a dynamic segment.

In the static segment, messages are transmitted based on the TDMA protocol. That is, the segment is partitioned into fixed-length time-slots, as shown in Fig. 2. We denote the length of a static slot as Δ. The time when a message frame is sent on a TDMA slot can be precisely computed. These slots are in high demand because they are required to send safety-critical data with strict timing requirements. However, note that an empty frame is sent when the data is not ready, and therefore, the slot cannot be used by other message frames. Such inflexibility leads to a lower utilization of static slots. Our proposed scheduling scheme allocates these slots prudently so that a higher number of time-critical messages can use them.

The dynamic segment, as shown in Fig. 2, is partitioned into mini-slots of length δ, where $\delta \ll \Delta$ (e.g., $\delta \leq 0.1\Delta$). A FlexRay frame can take more than one mini-slots on the dynamic segment. Unlike the static segment where the slot counter is updated at the end of a slot, in the dynamic segment, the slot counter is updated when a frame is completely sent or at the end of an empty mini-slot. Thus, the time when a frame is sent on the dynamic segment depends on the size of the preceding frames. And therefore, a frame can experience significant timing jitters, while the worst-case response time can still be bounded [34]. However, when the data is not ready, only a mini-slot will go unused. Such a flexible TDMA policy results in a higher resource utilization. Hence, our proposed scheduling strategy tries to maximize the usage of the dynamic segment for sending the control data.

It is important to note here that in a general multi-QoS resource, the time division between high-QoS and low-QoS segments is dynamic, which is not the case with FlexRay. However, in our problem setting, each instance of a control data can be sent using either high-QoS (in a static slot) or low-QoS (in the dynamic segment). Considering the time granularity of one control period, FlexRay offers high-QoS and low-QoS communication options. Hence, we term FlexRay as a multi-QoS resource.

FlexRay communication is organized as an infinite repetition of $N_{com} = 2^N$ bus cycles, where N_{com} is configurable for FlexRay 3.0.1. Each bus cycle is of length T_{bus} time units. For $N = 6$, communication in the first 64 bus cycles can be configured, which will repeat in the next 64 cycles and so on.

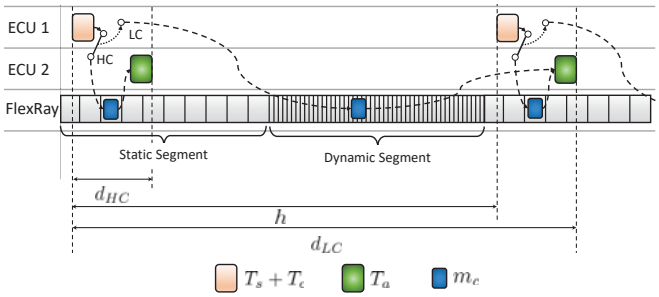

Fig. 2. Closed-loop timings for the two scheduling modes.

The schedule of a control data frame $m_{c,i}$ corresponding to the application $\mathcal{C}_i$ is represented using a tuple $\{s_{c,i}, b_{c,i}, r_{c,i}\}$. Here (i) $s_{c,i}$ is the assigned slot id, (ii) $b_{c,i}$ is the base cycle, i.e., the first cycle where the message is allocated the slot s_i, and (ii) $r_{c,i} \in \{2^k | 0 \leq k \leq \log_2 N_{com}\}$ is the cycle repetition rate that denotes the number of cycles between two consecutive slot allocations.

Slot multiplexing: FlexRay schedules are typically static and configured offline. While configuring the FlexRay schedules, we consider that no two frames are assigned the same slot in a FlexRay cycle. However, one slot id can be provisioned for different frames in different cycles. For example, two frames with schedules $\{5, 0, 2\}$ and $\{5, 1, 2\}$ are sent on slot 5 in even and odd cycles respectively. We exploit this opportunity of slot multiplexing as offered by FlexRay 3.0.1.

Despite the fact that FlexRay schedules cannot be reconfigured online, previous works, e.g., [10], [11], have proposed strategies to dynamically allocate static slots. These works exploit the middleware proposed in [15]. It requires a software to run on the same ECU as the controller tasks. This software selects and sends only one of the control data coupled with an identification tag on a slot in a bus cycle. An actuator task receives data in each cycle and evaluates the identifier to determine if the data is destined for the task. Thus, the controller tasks that are mapped on the same ECU can share a slot, which is a limitation. A dynamic policy would also require a scheduler to run on the ECU for each shared slot. In contrast, our proposed static scheduling strategy does not have any implementation overheads.

B. Switched Control Systems

In this work, we study linear and time-invariant (LTI) physical systems as they are common in practice [35], [36]. The continuous-time mathematical model for such a system is given as follows:

$$\dot{x}(t) = Ax(t) + Bu(t), \quad y(t) = Cx(t), \tag{1}$$

where $x(t)$, $u(t)$, and $y(t)$ represent the state, the control input, and the output, respectively. A, B, and C are the constant system matrices.

We study a distributed implementation of full state-feedback discrete-time controllers where the control data can be sent over the static or the dynamic segment of a FlexRay communication cycle [9], as depicted in Fig. 2. The two scheduling modes are denoted as HC (i.e., high-cost) and LC (i.e., low-cost) respectively, implying that the static slots are scarce, and hence, considered expensive to use compared to the communication over the dynamic segment. We denote the sensing-to-actuation delay in the HC and the LC modes as d_{HC} and d_{LC} respectively. In the HC mode, the communication delay is known precisely and is equal to the length of a static slot. Thus, the schedule of the tasks $T_{s,i}$, $T_{c,i}$, and $T_{a,i}$ can be optimized to obtain a very small delay. However, in the LC mode, the worst-case communication latency can be significantly high and the tasks have to be scheduled accommodating for the worst-case. Hence, we get $d_{HC} \ll d_{LC}$, as depicted in Fig. 2. Note that this assumption is also true for several other multi-modal implementation scenarios, e.g., (i) the control task switches between a high-priority (i.e., the HC mode) and a low priority (i.e., the LC mode); and (ii) the control task is executed from the scratchpad (i.e., the HC mode) or from the main memory (i.e., the LC mode).

For a delay d_M and a sampling period h, the equivalent discrete-time model of the plant in mode $M \in \{HC, LC\}$ can be written, similar to [37], as follows:

$$\begin{aligned} x[k+1] &= \Phi x[k] + \Gamma_M^0 u\left[k - \left\lfloor \frac{d_M}{h} \right\rfloor\right] + \Gamma_M^1 u\left[k - \left\lceil \frac{d_M}{h} \right\rceil\right], \\ y[k] &= Cx[k]. \end{aligned} \tag{2}$$

Here, $x[k]$ denotes the system states at the sampling instant t_k where $t_{k+1} - t_k = h$, while $u[k]$ is the control input calculated based on $x[k]$. Thus, for a zero-order hold implementation, $u(t) = u[k], \ \forall\, t_k + d_M \leq t < t_{k+1} + d_M$. In Eq. (2), Φ, Γ_M^0, and Γ_M^1 are calculated as follows:

$$\Phi = e^{Ah}; \ \Gamma_M^0 = \int_0^{h-d'_M} e^{At} B dt; \ \Gamma_M^1 = \int_{h-d'_M}^{h} e^{At} B dt; \tag{3}$$

where $d'_M = d_M - \left\lfloor \frac{d_M}{h} \right\rfloor h$.

Let us consider an augmented state vector $z[k] = \left[x[k] \quad u[k-1] \quad \cdots u\left[k - \left\lceil \frac{d_M}{h} \right\rceil\right]\right]^T$. For $0 < d_M \leq h$, Eq. (2) can be rewritten as follows:

$$\begin{aligned} z[k+1] &= \begin{bmatrix} \Phi & \Gamma_M^1 \\ \mathbf{0} & \mathbf{0} \end{bmatrix} z[k] + \begin{bmatrix} \Gamma_M^0 \\ \mathbf{I} \end{bmatrix} u[k] \\ &= \Phi_M^a z[k] + \Gamma_M^a u[k]; \\ y[k] &= [C \quad \mathbf{0}]\, z[k] = C_M^a z[k]. \end{aligned} \tag{4}$$

For $d_M > h$, Eq. (2) becomes:

$$\begin{aligned} z[k+1] &= \begin{bmatrix} \Phi & \mathbf{0} & \cdots & \Gamma_M^0 & \Gamma_M^1 \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} & \cdots & \mathbf{0} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \mathbf{0} & \cdots & \mathbf{0} & \mathbf{I} & \mathbf{0} \end{bmatrix} z[k] + \begin{bmatrix} \mathbf{0} \\ \mathbf{I} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{bmatrix} u[k] \\ &= \Phi_M^a z[k] + \Gamma_M^a u[k]; \\ y[k] &= [C \quad \mathbf{0} \quad \cdots \quad \mathbf{0}]\, z[k] = C_M^a z[k]. \end{aligned} \tag{5}$$

Here, $\mathbf{0}$ and $\mathbf{I}$ are zero and identity matrices of appropriate dimensions.

The feedback control law is given by:

$$u[k] = -K_M \cdot z[k], \tag{6}$$

where K_M is the feedback gain. The closed-loop model in mode M, therefore, is given by:

$$z[k+1] = (\Phi_M^a - \Gamma_M^a K_M) z[k] = \Phi_M^{cl} \cdot z[k], \tag{7}$$

where Φ_M^{cl} is the closed-loop state-transition matrix.

Control performance: Typically, the control performance J improves with a smaller sensing-to-actuation delay [38]. This is because a smaller delay enables a faster reaction from the controller to any physical disturbance (in case of stabilization control) or change in reference input (during reference tracking). We, therefore, assume that the control performance J_{HC} that is obtained when the controller runs only in the HC mode is better than the control performance J_{LC} obtained using the LC mode exclusively. In several real-world stabilization control problems like car suspension system and motor position control, it is critical to reject a physical disturbance within a specific time. Hence, we study settling time as the control performance measure that is defined as the amount of time the controller takes to reject the disturbance and get the plant back to the steady state. The lower the settling time is, the better the control performance is. It is important to note that the techniques proposed in this work can also be applied to reference tracking or even other performance metrics, e.g., quadratic cost. If J_r is the minimum settling time requirement and $J_{HC} < J_r < J_{LC}$, then operating the controller only in the LC mode does not meet the requirement. Conversely, exclusive use of the HC mode might be too conservative. Thus, we consider to switch between the modes to minimize the usage of static slots while simultaneously ensuring that the requirements are met.

Switching stability: In this work, we eventually compute a periodic static sequence of scheduling modes (HC and LC) for a controller such that the settling time requirement is met while the usage of high-QoS resources is minimized. Given a periodic sequence $\mathcal{M} = \{M_1, M_2, \cdots, M_m\}$, we can derive an equivalent closed-loop system as follows:

$$z[k+m] = \Phi^{cl}_{M_m} \times \Phi^{cl}_{M_{m-1}} \times \cdots \Phi^{cl}_{M_1} z[k] = \Phi^{cl}_{\mathcal{M}} z[k]. \quad (8)$$

If the eigenvalues of $\Phi^{cl}_{\mathcal{M}}$ lie inside a unit circle then the switched system is asymptotically stable. This means that the controller can reject any disturbance that might arrive. For the problem under study, if $\lceil \frac{d_{HC}}{h} \rceil \neq \lceil \frac{d_{LC}}{h} \rceil$, then Φ^{cl}_{HC} and Φ^{cl}_{LC} have different dimensions. However, we can compute an equivalent Φ^{cl}_{HC} of a higher dimension as Φ^{cl}_{LC} by starting with the same augmented state vector $z[k]$ as for the LC mode. For example, if $0 < d_{HC} \leq h$ and $h < d_{HC} \leq 2h$, we consider $z[k] = \begin{bmatrix} x[k] & u[k-1] & u[k-2] \end{bmatrix}^T$. Thus, Eq. (4) becomes:

$$z[k+1] = \begin{bmatrix} \Phi & \Gamma_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} \end{bmatrix} z[k] + \begin{bmatrix} \Gamma_0 \\ \mathbf{I} \\ \mathbf{0} \end{bmatrix} u[k],$$

and Eq. (6) becomes:

$$u[k] = - \begin{bmatrix} K_{HC} & \mathbf{0} \end{bmatrix} z[k].$$

We can compute Φ^{cl}_{HC} accordingly.

C. The Scheduling Problem

The implementation of a set of distributed control applications in the setting under consideration comprises several stages: (i) determining a periodic sequence of HC and LC modes for each application such that the usage of FlexRay static slots is minimized while the settling time requirement is met; (ii) allocating FlexRay static slots for the applications; (iii) scheduling the control data frames in the dynamic segment; and (iv) scheduling the application tasks on the ECUs. In this paper, we propose novel algorithms for the first two stages in Sec. IV and Sec. V respectively. Once the sequence of scheduling modes are fixed and the static slots are allocated for the HC modes, in stage (iii), we can check the schedulability of control data frames in the dynamic segment corresponding to the LC modes using a standard technique [34]. Here, we assume that a frame mapped on the dynamic segment will not be lost, i.e., it has a bounded worst-case latency. Further, in stage (iv), we can determine the task schedules on the processors by solving a constraint programming problem [21], [39]. Note that FlexRay-based ECU networks are time-synchronized and safety-critical control applications are typically implemented using time-triggered tasks [21], [39]. Details of stages (iii) and (iv) are omitted from this paper due to the limited space.

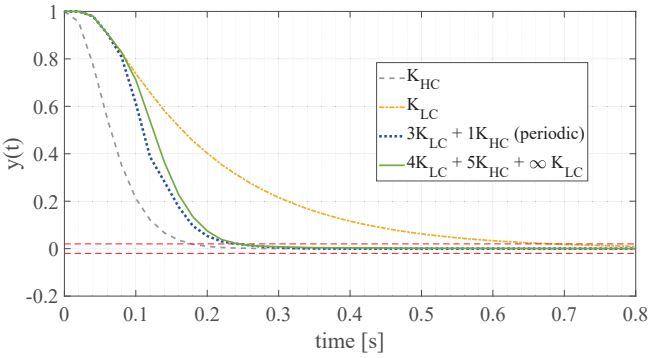

Fig. 3. Control responses for different static schedules.

III. A Motivational Example

We study a DC motor position control system [40] for which the continuous-time plant model is given by:

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -0.0227 & 54.5455 \\ 0 & -34.2857 & -70 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 0 \\ 28.1754 \end{bmatrix}, C^T = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}. \quad (9)$$

Here, the control goal is to stabilize the motor after a disturbance at the position $y = 0$ within a setting time of $J = 0.24\,\text{s}$. Towards modeling disturbances, we follow [41] that says "the effect of a disturbance on a linear system can be analyzed as an initial-value problem" in Page-370. Let us assume that the disturbance brings the motor to a state $x = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^T$ and the stability threshold is given by $||y[k]|| \leq 0.02, \forall k \geq J$.

For a sampling period of $h = 0.02\,\text{s}$, we consider control gains K_{HC} and K_{LC} for the HC and the LC modes respectively. They are given as follows:

$$K_{HC} = [30 \quad 1.2626 \quad 1.1071], \quad (10)$$

$$K_{LC} = [13.8921 \quad 0.5773 \quad 0.8672 \quad 1.0866]. \quad (11)$$

When the controller uses K_{HC} in the HC mode and K_{LC} in the LC mode, switching between the modes is stable, i.e., there exists a CQLF.

In Fig. 3, we show the control responses corresponding to different schedules. *Case 1:* When the controller only operates in the HC mode, the settling time is $0.18\,\text{s}$ (gray dashed line). *Case 2:* When the controller operates only in the LC mode, the settling time becomes $0.7\,\text{s}$ (orange dash-dotted line). *Case 3:* When the controller stays in the LC mode for 3 control instances followed by 1 instance in the HC

mode, and then repeats the pattern, the settling time is $0.24\,\mathrm{s}$ (blue dotted line). *Case 4:* This one replicates the dynamic scheduling policy proposed in [11] for which the settling time is also $0.24\,\mathrm{s}$ (green solid line). In this case, after the disturbance, the controller waits in the LC mode for 4 control instances before switching to the HC mode where it stays until the performance requirement is guaranteed to be met (i.e., 5 control instances in this case). And thereafter, it switches back to the LC mode until the next disturbance.

There are two important observations here: (i) In *Case 3*, by switching to the HC mode for $25\,\%$ of the instances, we can improve the settling time by almost 65%. Thus, it is possible to save static slots without compromising much on the control performance. (ii) For the state-of-the-art dynamic scheduling policy in *Case 4*, 5 static slots are required to reject the disturbance within $0.24\,\mathrm{s}$ compared to *Case 3* where only 3 slots were provided within $0.24\,\mathrm{s}$. As the disturbance can arrive at any control instance, the dynamic policy needs to provision 5 slots for any window of $0.24\,\mathrm{s}$ or 12 control instances. Thus, to account for the worst-case, $41.67\,\%$ of the slots needs to be provisioned in case of the dynamic policy. However, in *Case 3*, we use only $25\,\%$ of the slots. In this work, we try to derive a static schedule for the control applications similar to Case 3, thereby, enabling further resource savings.

Note that we consider static allocation of resources where the disturbance can arrive at any control instance. Thus, for a certain slot allocation, there can be multiple sequences of operating modes after a disturbance. For example, if $m_{c,i}$ is scheduled as $\{1, 0, 4\}$, i.e., it is sent on slot 1 every 4-th cycle starting from cycle 0. Now, if the disturbance arrives at cycle 0, then the sequence of operating modes will be $10001000\cdots$; while if the disturbance comes at cycle 1, then the sequence will be $00010001\cdots$. Here, 0 and 1 imply that the controller operates in the LC and the HC modes respectively. For a given slot allocation, we must evaluate all possible sequences that can be obtained by cyclic shift, and verify if the settling time requirement is met in all cases. In the same vein, the set of sequences for two slot allocations $m_{c,i} \sim \{1, 0, 4\}$ and $m_{c,i} \sim \{1, 3, 4\}$ are identical.

Now, for the switched controller, we obtain all possible slot allocations in which the controller can stay in the HC mode for 2 out of 8 consecutive instances. Overall, there are 28 such possibilities. For each slot allocation, we derive the set of all possible operating sequences. Note that there are only four unique sets of operating sequences. These sets are as follows: (i) $\mathbb{Q}_1 = \{11000000 \ll i | 0 \leq i < 8\}$; (ii) $\mathbb{Q}_2 = \{10100000 \ll i | 0 \leq i < 8\}$; (iii) (i) $\mathbb{Q}_3 = \{10010000 \ll i | 0 \leq i < 8\}$; (iv) $\mathbb{Q}_4 = \{10001000 \ll i | 0 \leq i < 4\}$. Here, $y \ll x$ implies a circular left shift of the literals in y by x positions, i.e., $10010000 \ll 2 = 01000010$. For each set, we determine the worst-case settling time $\widehat{J}$ that are tabulated in TABLE I. Note that the lowest settling time is obtained when the static slots are spread uniformly over control instances, while the maximum settling time is obtained when the two slots are allocated consecutively.

Note that in weakly-hard scheduling of control applications, uniform distribution of packet drops (or deadline misses in tasks) improves the control performance [16], [17]. Our experiments show a similar trend, i.e., when the static slots are spaced uniformly, a higher control performance is obtained. We use this intuition while determining the spread factor (i.e., how often the slots must be provisioned to meet the performance requirements in all scenarios) for the control applications in Sec. IV.

TABLE I
WORST-CASE SETTLING TIME ($\widehat{J}$) FOR DIFFERENT SLOT ALLOCATIONS

Set of operating sequence $\mathbb{Q}_i$	$\mathbb{Q}_1$	$\mathbb{Q}_2$	$\mathbb{Q}_3$	$\mathbb{Q}_4$
Worst-case settling time $\widehat{J}$ [s]	0.36	0.25	0.32	0.24

IV. SPREAD FACTOR COMPUTATION

When a control application $\mathcal{C}_i$ uses only the static slots for communication, one slot is assigned periodically to the application based on the period with which the controlled plant is sampled. For a sampling period h_i and a bus cycle time T_{bus}, the control data frame $m_{c,i}$ will be scheduled with a repetition rate $r_{c,i} = \frac{h_i}{T_{bus}}$. In N_{com} FlexRay cycles, the total number of slots used by the frame is equal to the number of control instances, i.e., $\widehat{n}_{h,i} = \frac{N_{com}}{r_{c,i}}$. For the switched controller under study, it might not be necessary to provide a slot for every instance as the control data can alternatively be sent on the dynamic segment. Here, our proposed framework GoodSpread uses a scalable technique to derive the number of slots $n_{h,i}$ that must be assigned to an application for sending the control data in $\widehat{n}_{h,i}$ control instances so that the settling time requirement is met. Note that according to the FlexRay protocol, the slot assignments repeat every N_{com} cycles, i.e., $\widehat{n}_{h,i}$ control instances.

We denote $J_{r,i}$ as the settling time requirement for $\mathcal{C}_i$ and $\widehat{J}_i$ as the worst-case settling time obtained after scheduling. Thus, $m_{c,i}$ must be scheduled to guarantee $\widehat{J}_i \leq J_{r,i}$. The settling time will depend on the closed-loop system properties in the HC and the LC modes respectively and the exact sequence of the modes in which the controller operates after a disturbance. For a particular slot assignment, there can be several operating sequences of the switched controller to reject disturbances. As explained in Sec. III, this is because a disturbance can arrive at any control instance. We are interested in the worst-case settling time considering all possible switching sequences corresponding to a slot assignment. It is challenging to deduce a mathematical expression for the settling time that can be used as a constraint during slot allocation. Thus, we have to search for a slot assignment that meets the settling time requirement using minimum number of static slots.

For example, when $h_i = T_{bus}$ and $N_{com} = 64$ cycles, there are 2^{64} ways in which slots can be assigned to $m_{c,i}$. To ensure that the settling time requirement will be met for a slot assignment, it is required to perform closed-loop simulations for all possible disturbance arrivals. Thus, it is computationally challenging to obtain a slot assignment for which we can guarantee that the application uses the minimum number of static slots to satisfy the settling time requirement.

Towards a more scalable search of the design space, GoodSpread uses the intuition gained from the example in Sec. III. That is, the settling time is the maximum when the slot

Algorithm 1: Compute spread factor

```
   Input  : Φ_i = (A_i, B_i, C_i), h_i, K_{HC,i}, K_{LC,i}, J_{r,i}, T_bus,
            N_com
   Output : {n_{h,i}, n̂_{h,i}}, S_i
1  n̂_{h,i} = (N_com · T_bus) / h_i ;
2  for n_{h,i} ← 0 to n̂_{h,i} do
3      S_i = zeros(1, n̂_{h,i});
4      if n_{h,i} > 0 then
5          S_i(1) = 1;
6          pos = 1;
7          for j ← 1 to n_{h,i} − 1 do
8              dpos = round( (n̂_{h,i} + 1 − pos) / (n_{h,i} + 1 − j) );
9              pos = pos + dpos;
10             S_i(pos) = 1;
11         end
12     end
13     isFeasible = true;
14     for k ← 0 to n̂_{h,i} − 1 do
15         Q = S_i ≪ k ;                    // cyclic left shift
16         J_i^k = Simulate (Φ_i, h_i, K_{HC,i}, K_{LC,i}, Q);
17         if J_i^k > J_{r,i} then
18             isFeasible = false;
19             break;
20         end
21     end
22     if isFeasible == true then
23         return {n_{h,i}, n̂_{h,i}}, S_i;
24     end
25 end
26 return {}, [];
```

allocation is contiguous while the settling is the minimum when the slots are spread uniformly over time. For the example in Sec. III, let us consider a settling time requirement $J_{r,i} = 0.24\,\text{s}$. We know from the results in TABLE I that 2 out of 8 slots are sufficient to meet the requirement when the 2 static slots are separated by 4 samples. However, if we assign static slots in consecutive samples, then we need 5 of them in 8 control instances to meet the settling time requirement. Considering that the main goal of this paper is to save static slots, we only evaluate the case where the slots are spread uniformly over time.

Proposed algorithm: GoodSpread uses Algorithm 1 to compute the *spread factor* that quantifies how to efficiently spread the static slots over the control instances for an application $\mathcal{C}_i$ to meet the settling time requirement $J_{r,i}$ in all possible scenarios. Based on our intuition, we denote the spread factor as $\{n_{h,i}, \widehat{n}_{h,i}\}$, i.e., $n_{h,i}$ static slots are spread as uniformly as possible over $\widehat{n}_{h,i}$ control instances. Algorithm 1 takes as input the continuous-time plant model $\Phi_i = (A_i, B_i, \mathcal{C}_i)$, the sampling period h_i, the control gains, $K_{HC,i}$ and $K_{LC,i}$, for the HC and the LC modes respectively, the settling time requirement $J_{r,i}$, the bus cycle time T_{bus} and the number of configurable FlexRay cycles N_{com}. It returns the spread factor $\{n_{h,i}, \widehat{n}_{h,i}\}$ and a corresponding slot assignment S_i.

In line 1, the algorithm calculates the total number of control instances $\widehat{n}_{h,i}$ in N_{com} consecutive cycles. In lines 2-25, it starts with zero slot and increments by one in each iteration until $\widehat{n}_{h,i}$, while simultaneously determining the minimum number of slots that meets the settling time requirement.

In lines 3-12, the algorithm generates a slot assignment S_i for $\widehat{n}_{h,i}$ control instances with the given number of slots in the current iteration, i.e., $n_{h,i}$ slots. Based on our intuition, we distribute the slots as uniformly as possible. Note that the slot assignment will repeat infinitely. For example, if we assign a static slot for the third control instance and $\widehat{n}_{h,i} = 16$, then a slot will also be provided for the 19-th control instance. Now, in line 3, we initialize S_i with zero slots. In lines 4-12, we add slots to S_i only if $n_{h,i} > 0$. In line 5, the first slot is placed in the first control instance and its position pos is noted in line 6. In lines 7-11, the algorithm places the remaining $n_{h,i} - 1$ slots one by one. Towards this, in line 8, it calculates the distance $dpos$ to the next slot from the current slot. Note that the slot corresponding to $S_i(1)$ will repeat at $\widehat{n}_{h,i} + 1$. Until that slot, there are $\widehat{n}_{h,i} + 1 - pos$ control instances and $n_{h,i} + 1 - j$ slots. Thus, the algorithm calculates the average distance and round it up to get $dpos$. The current position pos is updated based on the value of $dpos$ in line 9. A slot is assigned in the updated position in line 10.

Let us consider an example where we want to assign 3 slots in 8 control instances. We start with $S_i(1) = 1$ and then we have 2 remaining slots to be assigned. In the first iteration, $dpos = \textbf{round}(\frac{8}{3}) = 3$, and therefore, we place the next slot in the fourth control instance, i.e., $S_i(4) = 1$. In the next iteration, we get $dpos = \textbf{round}(\frac{5}{2}) = 3$, and therefore, the next slot is provided in the seventh control instance, i.e., $S_i(7) = 1$. Finally, we obtain $S_i = [1, 0, 0, 1, 0, 0, 1, 0]$.

In lines 13-24, for the obtained slot assignment S_i, all possible sequences are evaluated in which the controller can operate after a disturbance. When the disturbance arrives in the first control instance and $S_i = [1, 0, 0, 1, 0, 0, 1, 0]$, the periodic sequence of operating modes is given by $Q = S_i$. Here, 0 and 1 imply that the controller operates in the LC and the HC modes respectively. However, for the same S_i, if the disturbance is experienced in the third control instance, then the periodic operating sequence is given by $Q = [0, 1, 0, 0, 1, 0, 1, 0]$, i.e., $S_i \ll 2$. Similarly, we can obtain all possible operating sequences for a particular slot assignment by cyclically shifting positions of the array elements towards left or right. The set of operating sequences, is therefore, given by:

$$\mathbb{Q} = \{S_i \ll k \mid 0 \leq k < \widehat{n}_{h,i}\}. \tag{12}$$

In line 13, $isFeasible$ is initialized as true. Using a for-loop (lines 14-21), all possible operating sequences are iterated by changing the value of k from 0 to $\widehat{n}_{h,i} - 1$. A periodic operating sequence Q is obtained by performing a left circular shift on the elements of S_i by k positions (line 15). For the obtained Q, the closed-loop system is simulated based on the plant model $\Phi_i = (A_i, B_i, \mathcal{C}_i)$, the sampling period h_i, and the control gains in the two modes ($K_{HC,i}$ and $K_{LC,i}$), as per Eq. (7). Here, the algorithm also evaluates if the system is stable for the given operating sequence as explained in Sec. II-B. From the simulated control response, the value of the settling time J_i^k is calculated (line 16). If $J_i^k > J_{r,i}$, it implies that there exists a sequence for the current slot assignment for which the settling time requirement is violated, and therefore, it is not necessary to evaluate further sequences for the current slot assignment (lines 17-20). Correspondingly, $isFeasible$ is set to false (line 18) and then the algorithm breaks out of the loop (line 19). On the other hand, if the settling time requirement

TABLE II
WORST-CASE SETTLING TIME VS NUMBER OF STATIC SLOTS

No. of static slots in 16 control samples	1	2	3	**4**
Worst-case settling time $\widehat{J}_i$ [s]	0.5	0.36	0.35	**0.24**

is met for all possible sequences then $isFeasible$ remains true. Thus, in lines 22-24, $isFeasible$ is evaluated, and if it is true then the spread factor is returned as $\{n_{h,i}, \widehat{n}_{h,i}\}$ together with a slot assignment S_i that satisfies the spread factor. Note that $n_{h,i}$ is the minimum number of slots that must be spread uniformly over $\widehat{n}_{h,i}$ control instances to meet the settling time requirement for any disturbance pattern. Furthermore, in case the requirement is not met even by operating the controller only in the HC mode, i.e., $J_{r,i} < J_{HC,i}$, the algorithm returns empty sets for the spread factor and the slot allocation respectively denoting that the problem is infeasible (line 26).

Algorithm 1 returns only one slot assignment S_i corresponding to the obtained spread factor. However, we can perform a circular shift on the elements of S_i to derive more scheduling options. For example, if $S_i^j = [1,0,0,1,0,0,1,0]$ is returned by the algorithm when the spread factor in $\{3,8\}$, then $S_i^{j'} = [0,0,1,0,0,1,0,1]$ also satisfies the spread factor $\{3,8\}$. Both slot assignments will result in an identical set of operating sequences. If $S_i^{j'}$ is obtained from S_i^j by performing a circular left shift, then an operating sequence for S_i^j is also a sequence for $S_i^{j'}$ and vice versa. This is written as follows:

$$S_i^j \ll k' = S_i^{j'} \implies S_i^j \ll k = S_i^{j'} \ll (k - k'). \quad (13)$$

As we have evaluated the settling time for each possible sequence of operation modes corresponding to S_i^j, it is equivalent to verifying the settling time for each sequence obtained for $S_i^{j'}$. Therefore, if S_i^j is a feasible slot assignment then $S_i^{j'}$ is also feasible.

Example: We apply Algorithm 1 to determine the spread factor for the DC motor position control application ($\mathcal{C}_i$) given in Sec. III. We assume $T_{bus} = h_i = 0.2\,\text{s}$, $N_{com} = 16$, and $J_{r,i} = 0.24\,\text{s}$. It can be seen in TABLE II that using 4 slots in 16 control samples, the worst-case settling time $\widehat{J}_i$ is $0.24\,\text{s}$. Using a lower number of slots, the requirement is not met. Thus, the spread factor in this case is $\{4, 16\}$.

Complexity analysis of Algorithm 1: In the algorithm, we have two for-loops, i.e., lines 7-11 (IL_1) and lines 14-24 (IL_2) respectively, inside an outer loop OL (lines 2-25). Here, for each value of $n_{h,i}$ within the range $[1, \hat{n}_{h,i}]$ in OL, IL_1 iterates $(n_{h,i}-1)$ times. Therefore, the total iterations for IL_1 is upper bounded by $\frac{\hat{n}_{h,i}\cdot(\hat{n}_{h,i}-1)}{2}$. On the other hand, for a value of $n_{h,i}$, IL_2 can maximally run $\hat{n}_{h,i}$ times. Thus, the total number of iterations for IL_2 is upper-bounded by $\hat{n}_{h,i}\cdot(\hat{n}_{h,i}+1)$. Here, $\widehat{n}_{h,i}$ can attain a maximum value of N_{com}. Further, in IL_2, we simulate the closed-loop system. Considering that in the worst-case, the settling time for an operating sequence will be $J_{LC,i}$, i.e., the settling time obtained when the controller operates only in the LC mode. Thus, we can simulate the system up to $n_{LC,i} = \frac{J_{LC,i}}{h_i}$ samples. Note that we design the control gains such that the closed-loop system is stable, thus, $n_{LC,i}$ is not exponential. In fact, we study fast control loops with lower settling times that are common in domains like automotive, avionics and industry automation. The maximum number of times we compute the system states and the control input (as per Eq. (7) and Eq. (6) respectively) is given by $\widehat{n}_{h,i} \cdot (\widehat{n}_{h,i} + 1) \cdot n_{LC,i}$. Therefore, the asymptotic time complexity of the proposed algorithm is $\mathcal{O}(\widehat{n}_{h,i}^2 \cdot n_{LC,i})$. This implies that Algorithm 3 has polynomial-time complexity.

V. EXTENSIBILITY-DRIVEN SCHEDULE OPTIMIZATION

Based on the control requirements, GoodSpread determines the spread factor using Algorithm 1 corresponding to which there is a set of scheduling options for an application. Next, GoodSpread selects a feasible static allocation of slots to applications taking into account the available options. In the process, it optimizes the extensibility of the schedule. Here, the scheduling problem is formulated with a Satisfiability Modulo Theories (SMT) model.

Scheduling constraint for an application: Let us consider a set of binary variables $\{\gamma_{i,j} \in \{0,1\} \mid 1 \leq j \leq N_{com}\}$, where $\gamma_{i,j} = 1$ denotes that $\mathcal{C}_i$ will get a slot in the communication cycle $j-1$. Here, the cycles are numbered from 0 to $N_{com}-1$. For a spread factor $\{n_{h,i}, \widehat{n}_{h,i}\}$, we can formulate a constraint using the binary variables as follows:

$$\sum_{j=1}^{N_{com}} \gamma_{i,j} = n_{h,i}. \quad \text{(A1)}$$

This constraint states that we should allocate exactly $n_{h,i}$ slots within N_{com} cycles.

Note that it is not sufficient to ensure that we assign $n_{h,i}$ slots to $\mathcal{C}_i$ in N_{com} cycles. We must also consider the order in which the slots are assigned. Towards this, we can first determine the set of control instances $\mathcal{K}_i$ in which the slots are assigned in S_i as follows:

$$\mathcal{K}_i = \{k \in \mathbb{N} \mid S_i(k) = 1\}. \quad (14)$$

For a sampling period $h_i = 2^m \cdot T_{bus}$, a control instance $k \in \mathcal{K}_i$ does not always correspond to the cycle number $k-1$. This is because when sampling period is higher than the bus cycle time, the control data is not sent every cycle. For example, if $S_i = [1,0,1,0]$, $h_i = 4 \cdot T_{bus}$, and the first control instance is scheduled in cycle 0, then $\mathcal{K}_i = \{1,3\}$ will imply that $\mathcal{C}_i$ will get a slot in cycles 0 and 8. Thus, $\gamma_{i,1}$ and $\gamma_{i,9}$ will be 1. Here, two slots are separated by 8 cycles, i.e., two samples, which is also suggested by S_i. Considering that the first control instance is scheduled in cycle 0, we can determine a set of cycle numbers corresponding to S_i using $\mathcal{K}_i$ as follows:

$$\mathcal{K}_i^* = \{(k-1) \cdot \frac{h_i}{T_{bus}} \mid k \in \mathcal{K}_i\}. \quad (15)$$

Now, if we want to assign slots exactly in the cycles contained in $\mathcal{K}_i^*$, we can formulate a constraint as follows:

$$\bigwedge_{j \in \mathcal{K}_i^*} \gamma_{i,j+1} = 1. \quad (16)$$

However, the first control instance can be scheduled anywhere between cycles 0 and cycle $\frac{h}{T_{bus}} - 1$. Moreover, as argued in Sec. IV, a slot assignment that is obtained by performing a circular shift on S_i also satisfies the spread factor

$\{n_{h,i}, \widehat{n}_{h,i}\}$. Considering these flexibilities, we formulate a constraint as follows:

$$\bigvee_{j'=0}^{N_{com}-1} \left(\bigwedge_{j \in \mathcal{K}_i^*} \gamma_{i,\text{mod}(j+j',N_{com})+1} \right) = 1. \quad \text{(A2)}$$

Here, $\text{mod}(x, y)$ gives the remainder when x is divided by y. For the case where $S_i = [1, 0, 1, 0]$ and $h_i = 4 \cdot T_{bus}$, the constraint is written as follows:

$$(\gamma_{i,1} \wedge \gamma_{i,9}) \vee (\gamma_{i,2} \wedge \gamma_{i,10}) \vee \cdots \vee (\gamma_{i,16} \wedge \gamma_{i,8}) = 1.$$

Note that there might be some redundant terms in the constraint, e.g., $\gamma_{i,8} \wedge \gamma_{i,16}$ and $\gamma_{i,16} \wedge \gamma_{i,8}$, that can be easily handled in the implementation.

Notion of extensibility: Schneider et al. in [18] have studied extensibility of FlexRay schedules. We follow a similar notion of extensibility as explained using the following example.

Three control applications $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$ require $n_{h,1} = 1$, $n_{h,2} = 1$, and $n_{h,3} = 3$ slots in $N_{com} = 8$ communication cycles. Based on the requirements, we can derive two sets of FlexRay schedules as follows: (i) Case 1: $m_{c,1} \sim \{1, 2, 8\}$, $m_{c,2} \sim \{1, 4, 8\}$, and $m_{c,3} \sim [\{1, 0, 8\}, \{1, 3, 8\}, \{1, 6, 8\}]$; (ii) Case 2: $m_{c,1} \sim \{1, 1, 8\}$, $m_{c,2} \sim \{1, 7, 8\}$, and $m_{c,3} \sim [\{1, 0, 8\}, \{1, 3, 8\}, \{1, 6, 8\}]$. In Case 1, slot id 1 is allocated in cycles 0, 2, 3, 4, and 6 while in Case 2, applications use slot id 1 in cycles 0, 1, 3, 6, and 7. Thus, in Case 1, we can still add schedules $\{1, 1, 4\}$ and $\{1, 7, 8\}$, while in Case 2, we can add $\{1, 2, 8\}$, $\{1, 4, 8\}$, and $\{1, 5, 8\}$. Now, if we want to add a periodic FlexRay frame with a repetition rate of 4, it is possible in slot id 1 for Case 1, however, it is not possible for Case 2 despite having the same number of unused slots. Note that in Case 1, slot id 1 can alternatively accommodate three different frames with a repetition rate of 8, as similar to Case 2. Thus, Case 1 is more extensible than Case 2 as it provides maximum flexibility for adding future messages.

Minimizing the use of different slot ids: Let us consider an example where we need to schedule two periodic FlexRay frames, each with a repetition rate of 2. We can schedule them using either two different slot ids, e.g., $\{4, 0, 2\}$ and $\{5, 0, 2\}$, or a single slot id, e.g., $\{4, 0, 2\}$ and $\{4, 1, 2\}$. Here, the latter schedule is more extensible than the former. This is because slot id 5 can be provided to a future message frame requiring a repetition rate of 1 in the latter schedule, which is not possible in the former case. Towards our goal of synthesizing extensible schedules, it is important to use as fewer slot ids as possible.

Note that, in this paper, we also consider aperiodic scheduling of control data frames in the FlexRay static segment. For example, if $S_i = [1, 0, 0, 1, 0, 0, 1, 0]$ and $h_i = T_{bus}$, then the control data can be sent in bus cycles 0, 3, and 6. That is, the first and the second frame instances are separated by 3 bus cycles, while 2 bus cycles elapse between the third and the fourth instances. Now, for such schedules, the minimum number of slot ids N_{id} that will be used cannot be calculated trivially as $\left\lceil \frac{\sum_{\mathcal{C}_i \in \mathbb{C}} n_{h,i}}{N_{com}} \right\rceil$. For example, if two control applications $\mathcal{C}_1$ and $\mathcal{C}_2$ need to be scheduled with $S_1 = [1, 0, 0, 1, 0, 0, 1, 0]$ and $S_2 = [1, 0, 1, 0, 1, 0, 1, 0]$ respectively, we cannot schedule them using one slot id despite $n_{h,1} + n_{h,2} = 3 + 4 = 7 \leq N_{com} = 8$. As there does not exist a trivial expression for N_{id}, GoodSpread formulates an optimization problem to determine N_{id}. Towards this, we add the following set of constraints:

$$\sum_{\mathcal{C}_i \in \mathbb{C}} \gamma_{i,j} \leq N_{id}, \quad \forall\ 1 \leq j \leq N_{com}. \quad \text{(A3)}$$

That is, in each cycle, we can allocate a maximum of N_{id} slots. The optimization problem can then be written as:

$$\textbf{Minimize}\ \ N_{id} \quad \textbf{s.t.}\ \text{(A1), (A2), and (A3).} \quad \text{(O1)}$$

Algorithm 2: Determine dummy requirements

```
   Input  : N_s, N_id, N_com
   Output : 𝔻
1  N_s^* = N_id · N_com − N_s;
2  𝔻 = [ ];
3  r = 2;
4  while N_s^* > 0 do
5      N_d = N_com / r;
6      if N_s^* ≥ N_d then
7          𝔻 = [𝔻  r];
8          N_s^* = N_s^* − N_d;
9      else
10         r = 2 · r;
11     end
12 end
13 return 𝔻;
```

Determining prospective FlexRay frames: Based on the value of N_{id} as obtained by solving (O1), GoodSpread determines a set of prospective frames $\mathbb{D}$ with maximum scheduling demands. Here, the lower the repetition rate is, the higher is the scheduling demand. If these frames in $\mathbb{D}$ can be co-scheduled with the control applications without increasing the slot ids, then such a schedule is guaranteed to have the maximum extensibility. Towards determining the set $\mathbb{D}$, GoodSpread first calculates the total number of slots N_s used by the control applications in N_{com} cycles as follows:

$$N_s = \sum_{\mathcal{C}_i \in \mathbb{C}} n_{h,i}. \quad (17)$$

Let us consider an example where $N_{com} = 64$, $N_s = 75$, and $N_{id} = 2$. There are 53 unused slots for the two slot ids. For such a case, the prospective frames in $\mathbb{D}$ will have repetition rates of 2, 4, 16, and 64, i.e., they will use 32, 16, 4, and 1 slots respectively. If $N_{id} \cdot N_{com} - N_s \geq N_{com}$, it means there are more than N_{com} free slots in total in N_{com} cycles. However, $\mathbb{D}$ will not have a frame with a repetition rate of 1. This is because if it was possible to add a frame with a repetition rate of 1, then the optimizer would have returned a lower value of N_{id}.

Using Algorithm 2, GoodSpread obtains $\mathbb{D}$ based on the values of N_s, N_{id}, and N_{com}. In line 1, the algorithm calculates the number of unused slots N_s^* in N_{com} cycles for N_{id} slot ids. It initializes $\mathbb{D}$ as an empty set in line 2. In line 3, r is initialized as 2, which implies that the algorithm will first try to add a frame with a repetition rate of 2. Note that 2 is the lowest possible repetition rate of a frame that can be mapped on a partially used slot id. Using the while loop in lines 4-12, the algorithm tries to find prospective frames until there are some unused slots, i.e., $N_s^* > 0$. For a repetition rate r, the number of slots N_d required in N_{com} cycles is calculated in line 5. Now, if the number of unused slots N_s^* is greater

than N_d, then a frame is added in $\mathbb{D}$ with a repetition rate of r and N_s^* is adapted accordingly (lines 6-8). Otherwise, the algorithm cannot add any more frames with a repetition rate r, and therefore, the value of r is incremented to the next possible value (lines 9-11). In line 13, it returns the set of prospective frames $\mathbb{D}$.

Maximizing extensibility: In this paper, we propose an algorithm to maximize the extensibility of the schedule for the control applications. The idea is to co-schedule prospective FlexRay frames with maximum possible scheduling demands together with the control data frames. This will ensure that slots are allocated to the control applications in an extensible way. Now, let us consider that we want to schedule a prospective frame D_t with a repetition rate r_t. We have to add scheduling constraints for such a frame. We take a set of binary variables $\{\beta_{t,j} | 1 \leq j \leq r_t\}$, where $\beta_{t,j} = 1$ if a slot is allocated to D_t in cycle $j-1$. We can formulate a constraint to ensure that we allocate a slot within r_t cycles as follows:

$$\sum_{j=1}^{r_t} \beta_{t,j} = 1. \tag{A4}$$

Let $\mathbb{D}^+$ represent the set of prospective frames that can be feasibly incorporated into the schedule of control applications and $D_{t'}$ represent a new frame that we want to co-schedule. We need to reformulate the constraint set (A3) considering these additional frames. That is, we cannot allocate more slots in one cycle than N_{id} that is obtained by solving (O1). This is formulated as follows:

$$\sum_{\mathcal{C}_i \in \mathbb{C}} \gamma_{i,j} + \sum_{D_t \in \mathbb{D}^+ \cup D_{t'}} \beta_{t,\text{mod}(j-1,r_t)+1} \leq N_{id}, \quad \forall\ 1 \leq j \leq N_{com}. \tag{A5}$$

Here, the first and the second terms on the left-hand side calculates the slots allocated to the control data frames and the prospective frames respectively in the $(j-1)$-th cycle.

GoodSpread uses Algorithm 3 to maximize the extensibility during schedule synthesis. It takes as inputs (i) the set of control applications $\mathbb{C}$; (ii) the sampling periods of the applications $\mathcal{H}$; (iii) the set of scheduling options $\mathcal{S}$ for the applications; (iv) the minimum number of slot ids N_{id} required to schedule the applications; (v) the ordered set of prospective frames $\mathbb{D}$; (vi) the number of configurable FlexRay cycles N_{com}; and (vii) the bus cycle time T_{bus}. The algorithm returns a valid schedule Ω with maximum possible extensibility.

In line 1, we initialize $\mathbb{D}^+$ as an empty set, where $\mathbb{D}^+$ denotes the set of prospective frames that can be feasibly co-scheduled with the control applications using N_{id} slot ids. In line 2, we declare Φ_1 as an SMT model. In lines 3 and 4, we add scheduling constraints (A1) and (A2) for the control applications. In lines 5-22, we identify the prospective frames with maximum scheduling demands that can be scheduled together with the control data frames.

In lines 6 and 7 respectively, we initialize an empty SMT model Φ_2 and copy the constraints from Φ_1 to Φ_2. We try to schedule the first frame $\mathbb{D}(1)$ in $\mathbb{D}$, and therefore, in line 8, we add constraint (A4) for $\mathbb{D}(1)$ in Φ_2. Note that this frame will have the maximum scheduling demand (or the lowest repetition rate) among all frames in $\mathbb{D}$. In line 9, we

Algorithm 3: Maximize extensibility

Input : $\mathbb{C}$, $\mathcal{H} = \{h_i | \mathcal{C}_i \in \mathbb{C}\}$, $\mathcal{S} = \{S_i | \mathcal{C}_i \in \mathbb{C}\}$, N_{id}, $\mathbb{D}$, N_{com}, T_{bus}
Output : Ω

```
1  𝔻⁺ = [ ];
2  Φ1 = initializeSMTmodel();
3  Φ1.addConstraint(A1, 𝒮, N_com);
4  Φ1.addConstraint(A2, 𝒮, ℋ, N_com, T_bus);
5  while 𝔻 ≠ [ ] do
6      Φ2 = initializeSMTmodel();
7      Φ2 = Φ1;
8      Φ2.addConstraint(A4, 𝔻(1));
9      Φ2.addConstraint(A5, ℂ, 𝔻⁺, 𝔻(1), N_id);
10     {Ω, isFeasible} = Φ2.solve();
11     if isFeasible == true then
12         Φ1.addConstraint(A4, 𝔻(1));
13         𝔻⁺ = [𝔻⁺  𝔻(1)];
14         𝔻 = 𝔻\𝔻(1);
15     else
16         r = 𝔻(1);
17         t = 1;
18         while 𝔻(t) == r do
19             𝔻 = [𝔻(1 : t − 1)  2r  2r  𝔻(t + 1 : end)];
20             t=t+2;
21         end
22     end
23 end
24 return Ω;
```

add constraints (A5) considering the control data frames, the frames in $\mathbb{D}^+$, and $\mathbb{D}(1)$. We solve Φ_2 in line 10. If there exists a feasible solution for Φ_2 then $\mathbb{D}(1)$ can be feasibly scheduled (lines 11-14). Correspondingly, we incorporate the constraint (A4) for $\mathbb{D}(1)$ in Φ_1 (line 12), add $\mathbb{D}(1)$ in $\mathbb{D}^+$ (line 13), and delete $\mathbb{D}(1)$ from $\mathbb{D}$ (line 14). However, if Φ_2 cannot be solved then $\mathbb{D}(1)$ cannot be scheduled (lines 15-22). This implies that the scheduling demand of $\mathbb{D}(1)$ cannot be met. In that case, we take $\mathbb{D}(1)$ and all other frames in $\mathbb{D}$ that have the same repetition rate and split each of them into two frames with double the repetition rate (lines 16-21).

For example, if $N_{com} = 8$ and a control application occupies slot 4 in the cycles 0, 3, and 6, then $\mathbb{D} = \begin{bmatrix} 2 & 8 \end{bmatrix}$ for slot 4. However, a frame with a repetition rate of 2 cannot be co-scheduled with the control application. Thus, after the first iteration $\mathbb{D}$ becomes $\begin{bmatrix} 4 & 4 & 8 \end{bmatrix}$. That is, when a certain repetition rate becomes infeasible we try the next possible value higher than that. This will ensure that we find a set of schedulable prospective frames with maximum possible scheduling demands. And this, in essence, maximizes the extensibility of the synthesized schedule.

When there is no further frames in $\mathbb{D}$, it means that the control data frames together with the frames in $\mathbb{D}^+$ completely occupy the slots corresponding to N_{id} slot ids in N_{com} communication cycles. The schedule Ω is returned by the algorithm in line 24. Note that in Ω, the slots allocated to the frames in $\mathbb{D}^+$ are actually free for future messages.

Complexity analysis of Algorithm 3: In the algorithm, we have a while loop (lines 5-23) which iterates until all the unused slots are allocated to the prospective frames. Let the number of unused slots be $\overline{N}_s^*$. Then, the maximum number of iterations is linear with respect to $\overline{N}_s^*$ and is given by: $\overline{N}_s^* + \min(\log_2 \frac{N_{com}}{2}, \left\lfloor \log_2 \overline{N}_s^* \right\rfloor)$. This is the case when only prospective frames with a repetition rate of N_{com} can

be accommodated. For example, if $\overline{N}_s^* = 27$ and $N_{com} = 64$, we first try prospective frames with repetition rates of 4, 8, 16, and 32 respectively. If these are not schedulable then we can add 27 frames with a repetition rate of 64. That is, in total we will have 31 iterations. In case, we can schedule a prospective frame with a lower repetition rate $r < N_{com}$, then we will have at least $(\frac{N_{com}}{r} - 1)$ less iterations.

In each iteration, the algorithm solves an SMT problem. For N_m number of frames to be scheduled (including the propsective ones), the maximum number of variables of the SMT problem is $N_m \cdot N_{com}$ and the maximum number of constraints is $2N_m + N_{com}$. For a given FlexRay configuration, N_{com} is a constant, and therefore, the numbers of variables and constraints are linear with respect to N_m. For example, when $N_m = 100$ and $N_{com} = 64$, the maximum number of variables and constraints are 6400 and 264 respectively. This is a reasonable-sized problem for the Z3 solver [42], [43] that we use in this work.

VI. Experimental Results

For our experiments, we consider 6 control applications denoted as $\mathcal{C}_1$, $\mathcal{C}_2$, $\mathcal{C}_3$, $\mathcal{C}_4$, $\mathcal{C}_5$, and $\mathcal{C}_6$ respectively. For these applications, the continuous-time plant models $\{A_i, B_i, C_i\}$ are given in TABLE III. We assume a sampling period of $h_i = 0.02$s for each application. The control gains, $K_{HC,i}$ and $K_{LC,i}$, that we use in the HC and the LC modes respectively, are provided in TABLE III. The control gains are obtained using pole placement considering the closed-loop model given in Eq. (7). For the HC mode, we assume zero delay, while for the LC mode, the delay is assumed to be equal to one sampling period. For each application, we assume that the steady state is $x(\infty) = \begin{bmatrix}0 & 0 & \cdots & 0\end{bmatrix}^T$ and the initial state after a disturbance is $x(0) = \begin{bmatrix}1 & 0 & \cdots & 0\end{bmatrix}^T$. The settling times, $J_{HC,i}$ and $J_{LC,i}$, that are obtained by operating the controller only in the HC mode or in the LC mode respectively, are shown in TABLE III. Note that for better comparison, we take the plant models and the control gains from [11]. For FlexRay, we assume that the bus cycle time is $T_{bus} = 0.02$ s and the number of configurable FlexRay cycles is $N_{com} = 16$.

Deriving permissible slot allocations: For each application, we take two different settling time requirements, noted as $J_{r,i}$ under *Case 1* and *Case 2* respectively in Table III. We use Algorithm 1 in GoodSpread to compute the spread factor $\{n_{h,i}, \widehat{n}_{h,i}\}$ for each application considering the two settling time requirements. The obtained spread factors are reported in TABLE III. For example, in case of $\mathcal{C}_1$, when $J_{r,1} = 0.36$ s, the spread factor is $\{2, 16\}$, while for $J_{r,1} = 0.24$s, the spread factor is $\{4, 16\}$. A spread factor of $\{2, 16\}$ implies that 2 slots are provided in 16 communication cycles where the slots are separated by 8 cycles, i.e., the slots are spread uniformly over the bus cycles. Corresponding to a spread factor of $\{2, 16\}$, we can provide a slot each in cycles 0 and 8, for example.

For this example where $\widehat{n}_{h,i} = 16$, there are 2^{16} possible slot allocations. We also perform an exhaustive search through all possible options as an alternative to Algorithm 1. For $\mathcal{C}_3$, we could obtain feasible allocations using 3 and 5 slots respectively for $J_{r,i} = 0.4$ s and $J_{r,i} = 0.3$ s, i.e., 1 slot could be saved in each case using exhaustive search. For all other cases, Algorithm 1 provides solutions with minimum number of slots. The results suggest that Algorithm 1 performs well in most cases, however, it does not guarantee optimality. Algorithm 1 is several order faster compared to the exhaustive search, and in cases, where $\widehat{n}_{h,i} = 2^n > 16$, exhaustive search is not feasible. Moreover, compared to existing dynamic schemes, Algorithm 1 enables to save significant number of static slots, as will be shown later in this section.

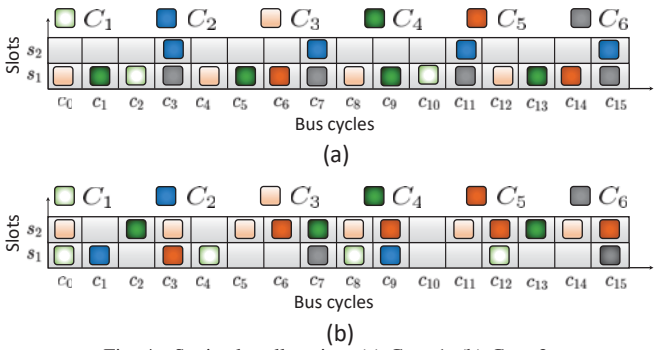

Fig. 4. Static slot allocation. (a) Case 1. (b) Case 2.

Maximizing extensibility during scheduling: We first consider the set of spread factors noted under *Case 1* in TABLE III and try to determine a FlexRay schedule for the applications based on them using GoodSpread. Here, we solve (O1) and obtain that 2 slot ids are required to schedule the applications. Now, we apply Algorithm 2 and Algorithm 3 to determine the most extensible schedule as shown in Fig. 4(a). Note that overall we require 20 slots, thus, 12 slots are unused for the two slot ids s_1 and s_2. Correspondingly, $\mathbb{D} = [2 \quad 4]$, i.e., if the unused slots can be allocated to two message frames with repetition rates of 2 and 4 respectively, there cannot be a schedule more extensible than that. For this case, we could find a schedule that can still accommodate two frames as $\{s_2, 0, 2\}$ and $\{s_2, 1, 4\}$. This shows that Algorithm 3 in GoodSpread can maximize the extensibility of the obtained schedule.

Now, for the set of spread factors given under *Case 2* in TABLE III, we solve (O1) and subsequently apply Algorithm 2 and Algorithm 3 to obtain the schedule shown in Fig. 4(b). In this case, 22 slots are occupied using 2 slot ids, and therefore, there are 10 unused slots. We get $\mathbb{D} = [2 \quad 8]$. However, in this case, there is no feasible schedule that can accommodate two additional frames with repetition rates of 2 and 8. The most extensible schedule that can be obtained in this case using GoodSpread, can accommodate frames with repetition rates 4, 8, and 16, i.e., $\{s_1, 2, 4\}$, $\{s_1, 5, 8\}$, $\{s_1, 11, 16\}$, $\{s_1, 11, 16\}$, $\{s_2, 1, 16\}$, $\{s_2, 4, 16\}$, and $\{s_2, 10, 16\}$.

Comparing with state-of-the-art techniques: We compare the results obtained using GoodSpread with the most efficient dynamic scheme [11] known so far. In this scheme, a slot id is fully reserved in all cycles for a set of applications. Now, when there is a disturbance, an application might have to wait for higher-priority applications to stop using the static slots. After waiting for $n_{w,i} = k$ samples, when it switches to the HC mode, it will use the slots for $n_{h,i}(k)$ consecutive instances, i.e., $n_{h,i}(\cdot)$ is a predetermined function of $n_{w,i}$. For $n_{w,i} = k$, we determine $\overline{n}_{h,i}(k) = \max_{0 \le j \le k} n_{h,i}(j)$, i.e., within

TABLE III
SPECIFICATION FOR THE CONTROL APPLICATIONS AND THEIR SPREAD FACTORS OBTAINED USING ALGORITHM 1

$\mathcal{C}_i$	$\{A_i, B_i, C_i\}$	$K_{HC,i}$	$K_{LC,i}$	$J_{HC,i}$ [s]	$J_{LC,i}$ [s]	*Case 1*		*Case 2*	
						$J_{r,i}$ [s]	$\{n_{h,i}, \widehat{n}_{h,i}\}$	$J_{r,i}$ [s]	$\{n_{h,i}, \widehat{n}_{h,i}\}$
C_1	Eq. (9)	Eq. (10)	Eq. (11)	0.18	0.7	0.36	$\{2, 16\}$	0.24	$\{4, 16\}$
C_2	$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -1.0865 & 8.4872 \cdot 10^3 \\ 0 & -9.9636 \cdot 10^3 & -1.4545 \cdot 10^6 \end{bmatrix}$, $B = \begin{bmatrix} 0 & 0 & 3.6364 \cdot 10^5 \end{bmatrix}^T$, $C = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0.1198 \\ -0.0130 \\ -2.9588 \end{bmatrix}^T$	$\begin{bmatrix} 0.0864 \\ -0.0128 \\ -1.6833 \\ 0.4059 \end{bmatrix}^T$	0.3	1	0.5	$\{4, 16\}$	0.7	$\{2, 16\}$
C_3	$A = \begin{bmatrix} -0.2 & 0.67 \\ -10 & -100 \end{bmatrix}$, $B = \begin{bmatrix} 0 \\ 37000 \end{bmatrix}$, $C = \begin{bmatrix} 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0.0500 \\ -0.0002 \end{bmatrix}^T$	$\begin{bmatrix} 0.0336 \\ 0.0004 \\ 0.4453 \end{bmatrix}^T$	0.2	0.62	0.4	$\{4, 16\}$	0.3	$\{6, 16\}$
C_4	$A = \begin{bmatrix} -10 & 1 \\ -0.02 & -2 \end{bmatrix}$, $B = \begin{bmatrix} 0 \\ 2 \end{bmatrix}$, $C = \begin{bmatrix} 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 100.0000 \\ 15.6226 \end{bmatrix}^T$	$\begin{bmatrix} -77.8275 \\ 24.3161 \\ 1.0265 \end{bmatrix}^T$	0.2	0.62	0.38	$\{4, 16\}$	0.4	$\{3, 16\}$
C_5	$A = \begin{bmatrix} -10 & 1 \\ -0.2 & 15 \end{bmatrix}$, $B = \begin{bmatrix} 0 \\ 20 \end{bmatrix}$, $C = \begin{bmatrix} 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 10.0000 \\ 1.0524 \end{bmatrix}^T$	$\begin{bmatrix} -2.4223 \\ 0.7014 \\ 0.2950 \end{bmatrix}^T$	0.2	0.5	0.36	$\{2, 16\}$	0.26	$\{5, 16\}$
C_6	$A = -0.05$, $B = 0.001$, $C = 1$	15000	$\begin{bmatrix} 8125.6 \\ 0.8659 \end{bmatrix}^T$	0.22	0.82	0.4	$\{4, 16\}$	0.5	$\{2, 16\}$

TABLE IV
RESOURCE PROVISIONING FOR DYNAMIC SCHEME IN [11]

$\mathcal{C}_i$	Case 1			Case 2		
	n^*_w	$n^*_{h,i}$	Slots [%]	n^*_w	$n^*_{h,i}$	Slots [%]
C_1	10	4	28.57	4	5	55.56
C_2	12	7	36.84	26	5	16.13
C_3	15	4	21.05	7	6	46.15
C_4	12	5	29.41	14	5	26.32
C_5	12	4	25	3	6	66.67
C_6	12	8	40	19	6	24

consecutive $k + \overline{n}_{h,i}(k)$ instances, at least $\overline{n}_{h,i}(k)$ slots must be provided to meet the requirement in all scenarios. Now, we determine $n^*_{w,i}$ and $n^*_{h,i}$ as follows:

$$\{n^*_{w,i}, n^*_{h,i}\} = \arg \min_{\forall n_{w,i}=k} \frac{\overline{n}_{h,i}(k)}{\overline{n}_{h,i}(k) + n_{w,i}}.$$

Here, $\frac{n^*_{h,i}}{n^*_{h,i}+n^*_{w,i}}$ gives the minimum fraction of instances for which slots must be provisioned for $\mathcal{C}_i$ using the dynamic policy. The results are given in TABLE IV for each application considering the two settling time requirements.

We can also calculate the fraction of instances for which slots are provided using GoodSpread from the spread factor as $\frac{n_{h,i}}{\widehat{n}_{h,i}}$. For *Case 1*, we allocate slots for $20.83\,\%$ of the instances per application compared to $30.15\,\%$ that is obtained for the dynamic policy, i.e., almost one-third of the slots can be saved per application. For *Case 2*, we can save even further, i.e., $40\,\%$ of the slots per application.

Note that for C_3, when $J_{r,i} = 0.4$, our technique provisions slots for $25\,\%$ of the instances compared to $21.05\,\%$ in case of the dynamic policy. This is mainly because Algorithm 1 returns a sub-optimal result for C_3. An exhaustive search gives the spread factor as $\{3, 16\}$. That is, $18.75\,\%$ of the instances use static slots which is less than that obtained using the dynamic policy. Although our proposed technique performs better than the existing dynamic schemes in most cases, it does not guarantee optimality. Devising provably optimal static and dynamic policies is a future work.

Furthermore, the dynamic policy in [11] will reserve 2 and 3 slot ids respectively for *Case 1* and *Case 2*. Considering that the slots corresponding to these slot ids cannot be used by any other application, in total 32 and 48 slots are reserved for *Case 1* and *Case 2* respectively. On the other hand, using our proposed technique, we use only 20 and 22 slots respectively for *Case 1* and *Case 2*. Thus, in *Case 1*, we save $37.5\,\%$ slots, while in *Case 2*, we save $54.17\,\%$. This shows that significant savings in high-QoS resources is possible using GoodSpread.

Scalability of Algorithm 3: Note that in Algorithm 3, we solve an SMT problem in each iteration. In this work, we use Z3 [44], [45] as the SMT solver. For *Case 1* and *Case 2*, Algorithm 3 takes $0.9\,\mathrm{s}$ and $8.3\,\mathrm{s}$ respectively. Here, *Case 2* takes a longer time because there are UNSAT problem instances (i.e., no feasible solution) and the solver takes time to prove that a problem is infeasible. To further evaluate the scalability of the algorithm, we construct a case combining *Case 1* and *Case 2*. That is, we have to schedule 12 control messages. For this case, we even consider $T_{bus} = 5\,\mathrm{ms}$ and $N_{com} = 64$. Thus, for each message, we need to consider 64 binary variables. This problem takes $70\,\mathrm{min}$ to run. Again, in this case the majority of the time is taken by UNSAT instances. The largest SAT instance (i.e., has a feasible solution) takes only $10\,\mathrm{s}$, which schedules 23 frames including 11 prospective ones. Here, the algorithm has encountered 11 SAT instances and 7 UNSAT instances. Now, when we put a time limit of $20\,\mathrm{s}$ in Z3 to solve a single instance, we get the same result in $193\,\mathrm{s}$, i.e. the computation time is reduced by $95\,\%$. Thus, based on the size of the problem, we can adjust the time limit in Z3 so that we do not have to wait a long time for UNSAT instances. This improves the scalability of Algorithm 3 significantly.

VII. CONCLUSION

This paper addresses the challenging problem of minimizing the provisioning of high-QoS resources in safety-critical cost-sensitive CPS. We follow a prevalent idea of substituting a fraction of high-QoS resources with lower-QoS resources without jeopardising the system's safety. While past works have considered dynamic allocation of resources, we demonstrate that, unlike the popular convention, a static allocation can save resource significantly in addition to its practical relevance. Here, we propose a simple idea of spreading the allocation of high-QoS resources as uniformly as possible over time so that whenever a disturbance arrives, it will be effectively rejected within the specified time. We further propose an optimization approach to prudently allocate high-QoS resources while maximizing the extensibility of the schedule.

ACKNOWLEDGEMENT

Marco Caccamo is supported by an Alexander von Humboldt Professorship endowed by the German Federal Ministry of Education and Research. Chakraborty's work is supported by NSF award #2038960.

REFERENCES

[1] D. Seto, J. P. Lehoczky, L. Sha, and K. G. Shin, "On task schedulability in real-time control systems," in *Real-Time Systems Symposium (RTSS)*, 1996.

[2] W. Chang, L. Zhang, D. Roy, and S. Chakraborty, *Control/architecture codesign for cyber-physical systems*, ser. Handbook of Hardware/Software Codesign. Springer Netherlands, 2017.

[3] F. Simonot-Lion, "In car embedded electronic architectures: How to ensure their safety," in *IFAC International Conference on Fieldbus Systems and their Applications (FET)*, 2003.

[4] P. Marti, J. M. Fuertes, G. Fohler, and K. Ramamritham, "Improving quality-of-control using flexible timing constraints: metric and scheduling," in *Real-Time Systems Symposium (RTSS)*, 2002.

[5] N. Navet and F. Simonot-Lion, *Automotive Embedded Systems Handbook*. CRC Press, 2009.

[6] "Road vehicles — Controller area network (CAN) — Part 2: High-speed medium access unit," International Organization for Standardization, ISO 11898-2:2016, 2016.

[7] FlexRay Consortium, "Flexray Communications System Protocol Specification Version 3.0.1," 2010.

[8] H. Faik and J. C. Kleinsorge, "Optimal Static WCET-aware Scratchpad Allocation of Program Code," in *Design Automation Conference (DAC)*, 2009.

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

[10] L. Maldonado, W. Chang, D. Roy, A. Annaswamy, D. Goswami, and S. Chakraborty, "Exploiting system dynamics for resource-efficient automotive cps design," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2019.

[11] D. Roy, W. Chang, S. K. Mitter, and S. Chakraborty, "Tighter dimensioning of heterogeneous multi-resource autonomous CPS with control performance guarantees," in *Design Automation Conference (DAC)*, 2019.

[12] "AUTOSAR Classic Platform Standards," https://www.autosar.org/standards/classic-platform/, accessed: 2020-01-01.

[13] P. Mundhenk, G. Tibba, L. Zhang, F. Reimann, D. Roy, and S. Chakraborty, "Dynamic platforms for uncertainty management in future automotive E/E architectures," in *Design Automation Conference (DAC)*, 2017.

[14] A. Wölfl, N. Siegmund, S. Apel, H. Kosch, J. Krautlager, and G. Weber-Urbina, "Generating qualifiable avionics software: An experience report (E)," in *International Conference on Automated Software Engineering (ASE)*, 2015.

[15] D. Majumdar, L. Zhang, P. Bhaduri, and S. Chakraborty, "Reconfigurable communication middleware for FlexRay-based distributed embedded systems," in *International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2015.

[16] J. Ning, S. Ye Qiong, and S. Françoise, "Graceful degradation of the quality of control through data drop policy," in *European Control Conference (ECC)*, 2007.

[17] S. Ghosh, S. Dutta, S. Dey, and P. Dasgupta, "A structured methodology for pattern based adaptive scheduling in embedded control," *Transactions on Embedded Computing Systems*, vol. 16, no. 5s, pp. 189:1–189:22, 2017.

[18] R. Schneider, D. Goswami, S. Chakraborty, U. Bordoloi, P. Eles, and Z. Peng, "Quantifying notions of extensibility in flexray schedule synthesis," *ACM Transactions on Design Automation of Electronic Systems*, vol. 19, no. 4, 2014.

[19] 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.

[20] A. Burns and R. I. Davis, "Mixed criticality systems - a review," 2015.

[21] D. Roy, L. Zhang, W. Chang, D. Goswami, and S. Chakraborty, "Multi-objective co-optimization of FlexRay-based distributed control systems," in *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2016.

[22] F. Mager, D. Baumann, R. Jacob, L. Thiele, S. Trimpe, and M. Zimmerling, "Feedback control goes wireless: Guaranteed stability over low-power multi-hop networks," in *International Conference on Cyber-Physical Systems (ICCPS)*, 2019.

[23] A. Aminifar, P. Eles, and Z. Peng, "Jfair: a scheduling algorithm to stabilize control applications," in *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2015.

[24] A. Aminifar, S. Samii, P. Eles, Z. Peng, and A. Cervin, "Designing high-quality embedded control systems with guaranteed stability," in *Real-Time Systems Symposium (RTSS)*, 2012.

[25] C. Yuan and F. Wu, "Hybrid control for switched linear systems With average dwell time," *Transactions on Automatic Control*, vol. 60, no. 1, pp. 240–245, 2015.

[26] W. Zhang, J. Hu, and A. Abate, "On the value functions of the discrete-time switched LQR problem," *Transactions on Automatic Control*, vol. 54, no. 11, pp. 2669–2674, 2009.

[27] R. Castane, P. Marti, M. Velasco, A. Cervin, and D. Henriksson, "Resource management for control tasks based on the transient dynamics of closed-loop systems," in *Euromicro Conference on Real-Time Systems (ECRTS)*, 2006.

[28] A. Cervin, J. Eker, B. Bernhardsson, and K.-E. Årzén, "Feedback–feedforward scheduling of control tasks," *Real-Time Systems*, vol. 23, no. 1/2, p. 25–53, 2002.

[29] X. Dai, W. Chang, S. Zhao, and A. Burns, "A dual-mode strategy for performance-maximisation and resource-efficient CPS design," *ACM Transactions on Embedded Computing Systems*, vol. 18, no. 5s, 2019.

[30] P. Pop, P. Eles, Zebo Peng, and T. Pop, "Scheduling and mapping in an incremental design methodology for distributed real-time embedded systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 8, pp. 793–811, 2004.

[31] W. Zheng, J. Chong, C. Pinello, S. Kanajan, and A. Sangiovanni-Vincentelli, "Extensible and scalable time triggered scheduling," in *International Conference on Application of Concurrency to System Design (ACSD)*, 2005.

[32] A. Ghosal, H. Zeng, M. Di Natale, and Y. Ben-Haim, "Computing robustness of FlexRay schedules to uncertainties in design parameters," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2010.

[33] W. Zimmermann and R. Schmidgall, *Bussysteme in der Fahrzeugtechnik*. Springer Fachmedien Wiesbaden, 2014.

[34] T. Pop, P. Pop, P. Eles, Z. Peng, and A. Andrei, "Timing analysis of the FlexRay communication protocol," *Real-Time Systems*, vol. 39, no. 1-3, pp. 205–235, 2008.

[35] K. Osman, M. Rahmat, and M. Ahmad, "Modelling and controller design for a cruise control system," in *International Colloquium on Signal Processing and Its Applications (CSPA)*, 2009.

[36] J. Fox, R. Roberts, C. Baier, L. Ho, L. Lacraru, and B. Gombert, "Modeling and control of a single motor electronic wedge brake," in *SAE Technical Paper 2007-01-0866*, 2009.

[37] H. Liang, Z. Wang, D. Roy, S. Dey, S. Chakraborty, and Q. Zhu, "Security-driven codesign with weakly-hard constraints for real-time embedded systems," in *International Conference on Computer Design (ICCD)*, 2019.

[38] S. Mohamed, A. U. Awan, D. Goswami, and T. Basten, "Designing image-based control systems considering workload variations," in *Conference on Decision and Control (CDC)*, 2019.

[39] R. Schneider, D. Goswami, S. Zafar, M. Lukasiewycz, and S. Chakraborty, "Constraint-driven synthesis and tool-support for FlexRay-Based automotive control systems," in *International conference on Hardware/software codesign and system synthesis (CODES+ISSS)*, 2011.

[40] N. Thomas and P. Poongodi, "Position control of dc motor using genetic algorithm based PID controller," in *World Congress on Engineering (WCE)*, 2009.

[41] K. J. Åström and B. Wittenmark, *Computer-Controlled Systems (3rd ed.)*, ser. Prentice Hall Information and System Sciences. Prentice-Hall Inc., 1997.

[42] W. Steiner, "An evaluation of SMT-based schedule synthesis for time-triggered multi-hop networks," in *Real-Time Systems Symposium (RTSS)*, 2010.

[43] S. Ghosh, S. Dey, and P. Dasgupta, "Pattern guided integrated scheduling and routing in multi-hop control networks," *Transactions on Embedded Computing Systems*, vol. 19, no. 2, 2020.

[44] L. De Moura and N. Bjørner, "Z3: An efficient SMT solver," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2008.

[45] N. Bjørner *et al.*, "νZ - An optimizing SMT solver," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2015.