Tighter Dimensioning of Heterogeneous Multi-Resource Autonomous CPS with Control Performance Guarantees

Debayan Roy
Technical University of Munich
debayan.roy@tum.de

Wanli Chang
University of York
wanli.chang@york.ac.uk

Sanjoy K. Mitter
Massachusetts Institute of Technology
mitter@mit.edu

Samarjit Chakraborty
Technical University of Munich
samarjit@tum.de

ABSTRACT

In modern autonomous systems, there is typically a large number of connected components realizing complex functionalities. For example, in autonomous vehicles (AVs), there are tens of millions of lines of code implemented on hundreds of sensors, controllers, and actuators. AVs have been deployed, mostly in trials and restricted environments, showing that substantial progress has been made in functionality development. However, they are still faced with two major challenges: (i) performance guarantee of safety-critical functions under *all* possible scenarios; (ii) functionality implementation with limited resources. These two challenges are conflicting because safety guarantees necessitate a worst-case analysis that is often very pessimistic for complex hardware/software systems, and thus require more resources. To address this, we study an abstraction of a heterogeneous cyber-physical systems architecture consisting of a mix of high- and low-quality resources, such as time- and event-triggered resources, or wired and wireless resources. We show that by properly managing such a mix of resources and formulating a formal verification (model checking) problem, it is possible to tightly dimension the high-quality resource to the minimum (50% in certain cases) while providing control performance guarantees.

KEYWORDS

Cyber-physical systems, autonomous vehicles, heterogeneous resource, resource-efficient design, formal verification

1 INTRODUCTION

In cyber-physical systems (CPS), there is a tight coupling between the dynamics of the physical plants and the cyber platform, implementing control algorithms using computation and communication resources. Autonomous systems (AS), such as autonomous vehicles (AVs), are instances of CPS, where safety is of utmost importance. It is required to guarantee safe behavior of the AS under *all* possible scenarios. Currently, the electrical and electronic (E/E) architectures of AS are highly heterogeneous consisting of different types of resources. A high-quality resource often has straightforward timing guarantees while being scarce and expensive. The challenge is to *achieve tighter dimensioning of the high-quality resources while satisfying the performance requirements*. This is particularly important for the cost-sensitive domains, such as AVs.

This work was supported by Deutsche Forschungsgemeinschaft (DFG) through the TUM International Graduate School of Science and Engineering (IGSSE).

DAC '19, June 2–6, 2019, Las Vegas, NV, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6725-7/19/06…$15.00
https://doi.org/10.1145/3316781.3317925

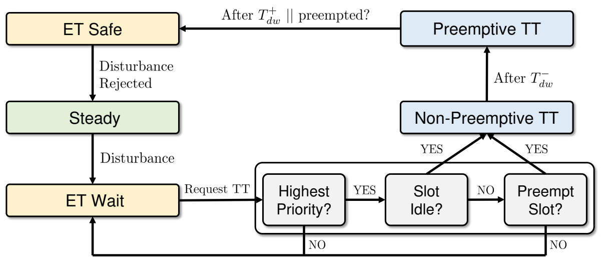


Figure 1: The proposed switching control strategy.

High-level idea: In this context, we consider a multi-resource CPS setup. A switching control strategy is proposed that exploits the heterogeneity in platform architecture to achieve better resource utilization. A formal verification problem is formulated to ensure that the control performance requirements are satisfied under all scenarios. With the interplay of control theory, scheduling, and model checking, the resource provisioning is tightened and safe behavior is guaranteed. While formal methods often suffer from state space explosion, we are able to get a result within reasonable time (seconds to 15 minutes).

Technical contributions and related work: We study a problem setting, where multiple distributed control applications in an AV exchange control signals over a FlexRay [5] bus. FlexRay allows both time-triggered (TT) communication (high-quality resource) and event-triggered (ET) communication (low-quality resource). Note that the essential idea proposed in this paper can be generally applied to other types of heterogeneous resources, such as wired and wireless communication. Previous works have considered a straightforward switching control strategy [9], where multiple applications share one TT slot and a non-preemptive scheduling policy decides which application gets the slot. An application uses the ET communication in the steady state and requests for a TT slot during a disturbance. Once it gets the TT slot, it will continue to use it until the disturbance is completely rejected. Such a switching strategy leads to a very conservative provisioning of TT slots.

In this work, we *precisely study the switching dynamics and propose a control strategy that enables a tighter resource dimensioning*. For a certain time T_w that an application has waited for the TT slot after a disturbance, we can predetermine the minimum time T_{dw}^{-} that it must use the slot to meet the performance requirement. Corresponding to T_w, there is also a maximum time T_{dw}^{+} after which the performance will not get improved even if the application continues to use the TT communication. Hence, the proposed switching strategy is as depicted in Fig. 1. Here, once an application gets the TT slot, it uses the slot free of preeemption till T_{dw}^{-}. After T_{dw}^{-}, the priority of the application is set to be the lowest, i.e., any application experiencing disturbance and requesting the TT slot, is able to preempt it. However, if the application is not preempted then it continues to use the TT slot till T_{dw}^{+} to improve its performance. After T_{dw}^{+}, it switches back to the ET communication. *Essentially, we provide each application with the minimum TT resource to guarantee the required performance. And the goal is to let as many applications share one TT slot as possible.*

Towards this goal, we need to solve a verification problem. That is, when multiple applications share a TT slot, can we formally guarantee that each will meet its performance requirement in all possible scenarios? Previous work has approached this as a standard schedulability analysis problem introducing significant conservativeness [9]. We formulate an exact *timed automata model of the system* and apply *model checking for the verification*. While there have been previous works on formulating schedulability analysis as model checking problems (e.g., [6], [4]), the main challenge here is to accurately characterize the interplay between the control system dynamics and the scheduling policy.

Studying the closed-loop dynamics of the switched system, we can determine a maximum time T_w^* that an application can wait (after a disturbance) for the TT slot without violating its performance requirement. We can also calculate T_{dw}^{-} and T_{dw}^{+} for all possible values of T_w. Using these timing variables T_{dw}^{-}, T_{dw}^{+}, T_w and T_w^*, it is possible to abstract the behavior of the system during a disturbance as a timed automaton. Furthermore, we consider that when multiple applications are simultaneously contesting for the TT slot, the application closest to its maximum waiting time T_w^* has the highest priority and gets the slot. Such a scheduling policy can also be represented as a timed automaton. Therefore, the whole system can be modeled as a network of timed automata. And we can verify that each of the applications must get TT slot before its T_w^* expires.

Note that the controllers are implemented according to a certain sampling period, and therefore, control signals are also sent at regular intervals. Correspondingly, switching between ET and TT communications can only take place at discrete time instants. Moreover, we are only interested in switching instants that satisfy the performance requirement while all other possibilities are abstracted by T_w^*. Therefore, the timing variables can only take a finite number of values (very few). And there is no issue of state-space explosion for this problem formulation.

2 PRELIMINARIES

In this section, we briefly discuss the background on heterogeneous communication resources and how to model and design control systems for different types of resource.

Heterogeneous communication resources: In this work, we consider a provision of both TT and ET communication, such as in FlexRay. A FlexRay bus cycle is composed of a static and a dynamic segment. The static segment exhibits TT communication and comprises a number of TDMA (time division multiple access)-like slots of equal length Ψ. A message assigned to a static slot is transmitted within the corresponding time window. Thus, the start and the end of a message transmission are precisely known. The dynamic segment implements ET communication and is partitioned into a number of mini-slots of equal length ψ, where typically $\psi \ll \Psi$. A message assigned to the dynamic segment may consume more than one mini-slot. Thus, the timing of a message depends on other preceding messages. This results in time-varying transmission delay while the worst case may still be determined [11]. Note that the proposed switching between TT and ET communication would require FlexRay to be runtime-configurable, which is not the case. Towards this, a communication middleware is proposed in [8].

Control systems: We consider discrete-time linear time-invariant (LTI) plant models given by

$$x[k+1] = \phi x[k] + \Gamma u[k], \qquad y[k] = Cx[k], \tag{1}$$

where x, u and y represent respectively the plant states, the control input, and the system output. ϕ, Γ, and C are system matrices. $x[k]$ is the plant state at the time instant $t[k]$. In a typical distributed setting, control data is sent over a communication bus, such as FlexRay. The sampling period between two consecutive sensing operations at $t[k]$ and $t[k+1]$ is a constant h.

For FlexRay, time-deterministic static slots allow to optimally implement controllers with negligible sensing-to-actuation delay. The state-feedback controller can be designed as

$$u[k] = -K_T x[k], \tag{2}$$

where K_T is the feedback gain. Combining 1 and 2, the closed-loop system dynamics is

$$x[k+1] = (\phi - \Gamma K_T)x[k]. \tag{3}$$

Eigenvalues of the closed-loop system matrix $\phi - \Gamma K_T$ determine the control performance, such as settling time. Optimization-driven pole-placement controller design technique can be found in [2].

On the other hand, when the control message is transmitted over the dynamic segment, the worst-case sensing-to-actuation delay needs to be considered. In this work, we assume a one sample delay where at $t[k]$, $u[k-1]$ is applied to the plant and is held till $t[k+1]$. Thus, the plant model becomes

$$x[k+1] = \phi x[k] + \Gamma u[k-1], \qquad y[k] = Cx[k]. \tag{4}$$

In this case, an augmented state vector $z[k] = \begin{bmatrix} x[k] & u[k-1] \end{bmatrix}^T$ is considered, and therefore, the control law becomes

$$u[k] = -K_E z[k] = -K_E \begin{bmatrix} x[k] \\ u[k-1] \end{bmatrix}. \tag{5}$$

For the system with augmented state z[k], standard pole-placement can be applied to design K_E. Details can be found in [2].

3 SWITCHING CONTROL STRATEGY

We consider a problem setting where several distributed control applications $\{C_i\}$ share a FlexRay bus. These applications can send control data in either or both of the static and the dynamic segments. For communication over the static segment, negligible sensing-to-actuation delay can be realized, and accordingly, a fast stabilizing controller K_T can be designed. However, when data is sent over the dynamic segment, provisioning for the worst-case may only allow to design a slow controller K_E.

Typically, safety-critical control applications have stringent performance requirements. As a performance metric, we define settling time J as the time taken by a controller to bring the system back to a steady state after a disturbance. We denote J^* as the minimum settling time requirement. Let J_T (or J_E) be the settling time of the system when K_T (or K_E) is applied and static (or dynamic) segment is used for data transmission. If $J_T < J^* < J_E$, the current practice is to reserve a TT slot for the application to guarantee safety. When several applications share a FlexRay bus, TT slots might become scarce. And the main challenge is to use these slots efficiently.

In this context, we exploit the hybrid communication protocol and propose a bi-modal switching control strategy. In mode M_T, K_T is applied to the system and a TT slot is used. Thus, the system dynamics is given by (1) and (2). In mode M_E, control input is determined by K_E and is sent over the dynamic segment. Accordingly, the closed-loop system model is represented by (4) and (5). When the system is in steady state, it is sufficient to apply stable control, and thus, the controller can stay in mode M_E. However, when a disturbance arrives, the controller must be fast enough to stabilize the system within J^*. As $J_E > J^* > J_T$, the controller must switch to M_T for a certain minimum time duration to ensure $J \leq J^*$.

Based on the above observation, we can infer that a TT slot might be shared among multiple applications. However, when disturbance arrives simultaneously at two or more application, then only one gets the slot while others wait. Let a controller switch from M_E to M_T after a wait time T_w. For a certain T_w, let T_{dw}^- be the minimum dwell time the controller must stay at M_T to meet $J \leq J^*$ while T_{dw}^+ be the maximum dwell time after which even if it stays at M_T it will not improve the settling time. Accordingly, the proposed switching strategy considers that once an application gets a TT slot, it uses the slot non-preemptively till T_{dw}^- after which it can be preempted by a waiting application. However, when no other application is contesting the slot then it can be used till T_{dw}^+. Once an application is preempted it will not come back to M_T even if it has not stayed till T_{dw}^+.

Given the requirement J^* and the controllers K_T and K_E, we can simulate the system for all possible switching sequences allowed by the proposed strategy. Thus, we can precalculate T_{dw}^- and T_{dw}^+ for all possible T_w and also find the maximum time T_w^* after which switching to M_T will not give $J \leq J^*$. Note that we can choose T_w with a certain granularity to enhance scalability. This approach also allows to implement the control strategy using less memory. There is a trade-off between conservativeness and memory requirement.

Comments on switching stability: In this work, we assume the standard sporadic disturbance model with a minimum disturbance inter-arrival time r, where $J^* < r$. If we can guarantee that $J \leq J^*$ using the proposed control strategy then we can conclude that the systems is stable. However, the objective here is to meet the requirement with minimal samples of M_T. Intuitively, if two control modes are not switching stable, the energy of the system may increase on switching. For certain switching sequences, this might lead to a higher settling time. Thus, we suggest that the controllers K_T and K_E are designed considering switching stability, i.e., the closed-loop systems in M_T and M_E must have a common Lyapunov function [7]. In Sec. 3.1, using a motivational example, we experimentally evaluate the impact of switching stability on the settling time.

3.1 A motivational example

A DC motor position control system [13] is considered for which the discrete-time plant model is given by

$$\phi = \begin{bmatrix} 1 & 0.0182 & 0.0068 \\ 0 & 0.7664 & 0.5186 \\ 0 & -0.3260 & 0.1011 \end{bmatrix}, \Gamma = \begin{bmatrix} 0.0015 \\ 0.1944 \\ 0.2717 \end{bmatrix}, C = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}. \quad (6)$$

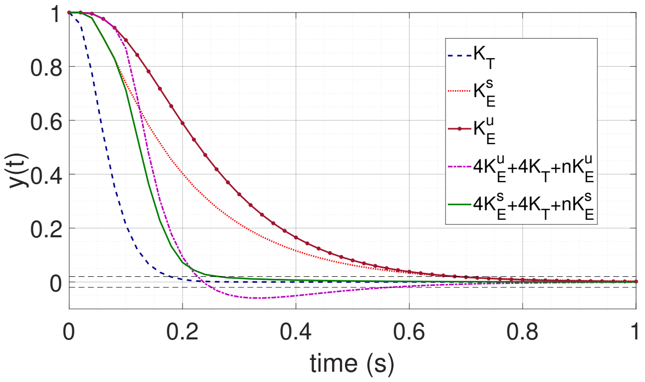


Figure 2: Response curves for different control strategies.

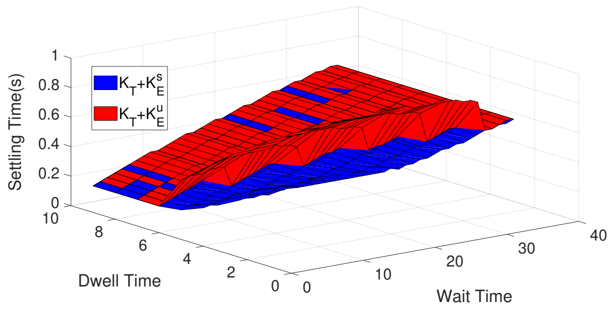


Figure 3: Performance with and without switching stability.

It is desired to keep the position at $y = 0$ and on disturbance the system moves to a state given by $x = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^T$. The settling time threshold is assumed as $||y[k]|| \leq 0.02, \forall k \geq J$.

For a sampling period of $h = 0.02$s, we design a controller K_T for mode M_T and two controllers K_E^s and K_E^u for mode M_E. They are given as follows:

$$K_T = \begin{bmatrix} 30 & 1.2626 & 1.1071 \end{bmatrix}, \quad (7)$$

$$K_E^s = \begin{bmatrix} 13.8921 & 0.5773 & 0.8672 & 1.0866 \end{bmatrix}, \quad (8)$$

$$K_E^u = \begin{bmatrix} 2.9120 & -0.6141 & -1.0399 & 0.1741 \end{bmatrix}. \quad (9)$$

Switching between K_E^s and K_T is stable while this is not the case for K_E^u and K_T.

As shown in Fig. 2, the settling time for K_T is 0.18s while for K_E^s and K_E^u it is 0.68s. The system response using the switching control strategy for two different cases are also plotted. In both cases, the application stays in M_E for 4 samples after a disturbance, followed by 4 samples in M_T, before returning to M_E again. As expected, better settling times are obtained by using 4 samples of M_T as compared to none. However, when K_E^s and K_T are used it gives a better settling time of 0.28s, as compared to 0.58s which is obtained when K_E^u and K_T are used. This difference in performance strongly suggests that, for our proposed strategy, the two controllers used in modes M_T and M_E must satisfy switching stability constraint.

We have also simulated the system for all possible switching combinations considering both pairs of controllers, i.e., $K_T + K_E^s$ and $K_T + K_E^u$. J as a function of T_w and T_{dw} (dwell time) is plotted in Fig. 3. Here, T_w and T_{dw} are measured in terms of no. of samples. The result shows that the system design without considering switching stability is resource-inefficient.

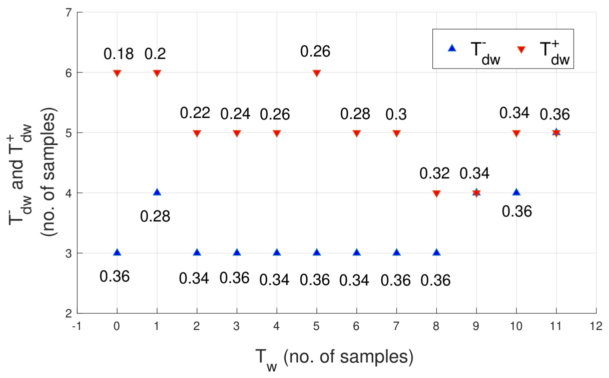


Figure 4: Minimum and maximum dwell times vs wait time.

Now, using K_T and K^s_E, the system is simulated for all possible switching combinations according to the proposed strategy. For $J^* = 0.36$s, we determine the minimum and maximum dwell times (T^-_{dw} and T^+_{dw}) for each possible wait time T_w, as shown in Fig. 4. Each point is annotated with the corresponding settling time values in seconds. Note that the minimum achievable settling time (corresponding to T^+_{dw}) is non-decreasing with increase in T_w. For $T_w = 0$, if we switch from M_T to M_E after 6 samples, we can still get the same performance as with a dedicated TT slot. Thus, it is very pessimistic to stay in M_T till the whole disturbance is rejected. It can be observed that T^-_{dw} and T^+_{dw} varies with T_w.

4 CONTROL PERFORMANCE VERIFICATION

We consider a scheduling policy similar to the earliest deadline first. Here, the deadline D is the time till which an application must be allocated a TT slot, i.e., $D = T^*_w - T_w$. When a slot is idle or the application using it can be preempted, the waiting application with the lowest value of D will get it. The scheduler can therefore be implemented as a multiplexer which selects the application that will use the TT slot for data transmission. We understand that such an implementation has a certain computation overhead, however, the main focus of this work is to minimize the use of expensive communication resource.

For this policy and the proposed control strategy, we must verify that all applications mapped on a slot will meet their requirements in all possible scenarios. The problem can be reformulated to verify that each application gets the TT slot before T^*_w has elapsed. Towards this, we propose to model the whole system as a network of timed automata and verify the model using UPPAAL [1]. Here, we abstract the control dynamics using timing variables like T_w, T^*_w, T^-_{dw} and T^+_{dw}. For each application, T^*_w and the variation of T^-_{dw} and T^*_{dw} with T_w can be predetermined. Here, the assumption is that the control models are fully known with no uncertainties.

A timed automaton (TA) is a finite state automaton with a finite set of real-valued clocks which progress synchronously. Different TAs can communicate via shared variables and synchronization channels. Our system model consists of TAs representing the applications, the scheduler and the arbitration policy.

There are two *main challenges* in modeling the system as timed automata. (i) The system under study is discrete-time, i.e., disturbances can be sensed only at periodic instants. However, timed automata has continuous-time semantics. Thus, it is challenging to model that the scheduler sees multiple slot requests at the same time. (ii) For a certain T_w, the values of T^-_{dw} and T^+_{dw} need to be looked up from a precomputed table. T_w can be measured using a clock. However, this clock cannot be used to reference table elements. Next, we describe the TAs used to model the system and explain how these challenges are addressed.

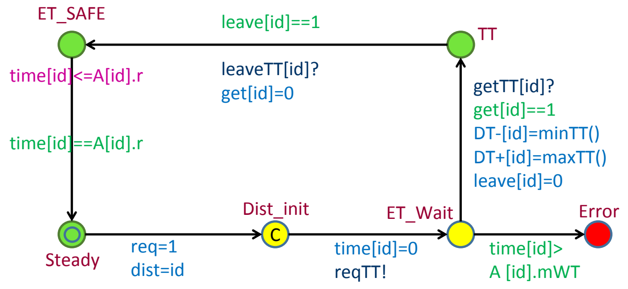
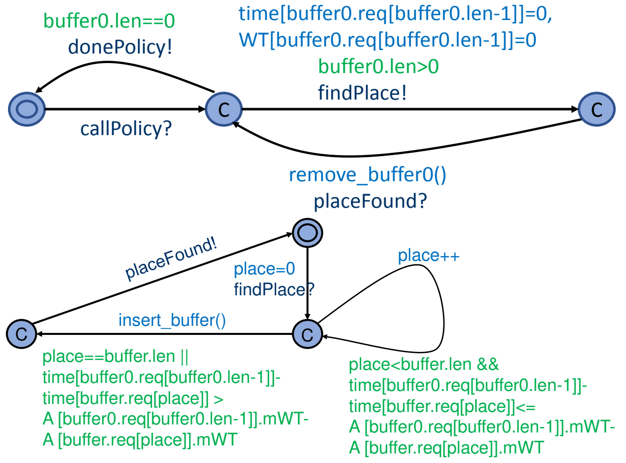


Figure 5: An application automaton.



buffer0.len==0
donePolicy!
time[buffer0.req[buffer0.len-1]]=0,
WT[buffer0.req[buffer0.len-1]]=0
buffer0.len>0
findPlace!
C
C
callPolicy?
remove_buffer0()
placeFound?
placeFound!
place=0
findPlace?
place++
insert_buffer()
C
C
place==buffer.len \|\|
time[buffer0.req[buffer0.len-1]]-
time[buffer.req[place]] >
A [buffer0.req[buffer0.len-1]].mWT-
A [buffer.req[place]].mWT
place<buffer.len &&
time[buffer0.req[buffer0.len-1]]-
time[buffer.req[place]]<=
A [buffer0.req[buffer0.len-1]].mWT-
A [buffer.req[place]].mWT

Figure 6: Policy (top) and Sort (bottom) automata.

Application automaton: Each application automaton has an *id* and uses a clock *time[id]*. As depicted in Fig. 5, it starts in the *Steady* state. In the event of a disturbance, it requests the scheduler for a TT slot using the synchronization channel *reqTT* and moves to the state *ET_Wait*. In this state, it waits for the TT slot. Here, *time[id]* measures the time that has elapsed since the disturbance is sensed. The transition from the *ET_Wait* state to the *Error* state can be taken if maximum waiting time has elapsed, i.e., *time[id]* is greater than T^*_w. Thus, an application meets the requirement in all scenarios only if it never reaches the *Error* state. On the other hand, TT slot allocation is notified via the synchronization channel *getTT[id]*. Correspondingly, the automaton takes the transition to the state *TT*. During this transition, the minimum and the maximum dwell times (*DT-[id]* and *DT+[id]*) is looked up based on the shared variable *WT[id]* which stores T_w. The scheduler preempts the application from the slot via the synchronization channel *leaveTT[id]*. The automaton correspondingly moves to the state *ET_SAFE*. It waits here till the minimum disturbance inter-arrival time elapses, i.e., *time[id]* $\leq r$, after which it moves to the *Steady* state again.

Automata for arbitration policy: Two nested TAs (*Policy* and *Sort*), as shown in Fig. 6, implement the arbitration policy. They basically sort the slot requests and keep them in the order it will be served. The scheduler maintains two queues, i.e., *buffer0* and *buffer*. Any requests coming in between two time samples are first

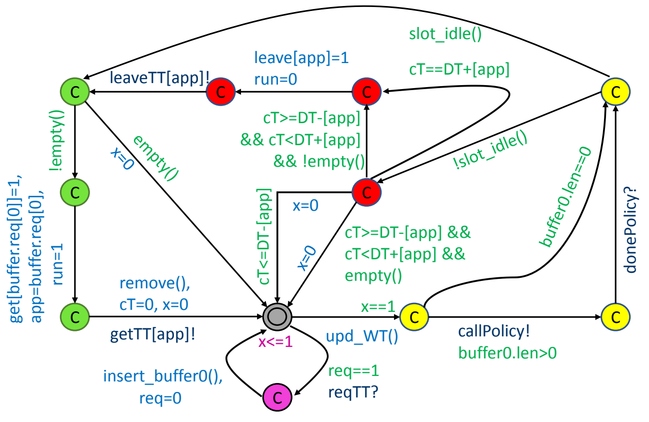


Figure 7: Scheduler automaton.

stored in *buffer0*. At each time sample, the scheduler invokes these automata to transfer requests from *buffer0* to *buffer* and sort them according to their respectively deadlines. Time does not pass during this operation and requests are served from *buffer* only after this operation. This partially addresses the first challenge where scheduler sees the disturbances that arrive during a period together at the next time instant. Here, *Policy* automaton checks for new requests in *buffer0*. For each new request in *buffer0*, *Sort* is invoked to insert the request correctly in *buffer*. Note that when a request is transferred from *buffer0* to *buffer*, the corresponding clock *time[id]* and the waiting time counter *WT[id]* are reset. This marks the time sample when the scheduler sees the disturbance for the first time. To place a new request correctly, *Sort* iterates through the requests in *buffer* one by one and compares their absolute deadlines to that of the incoming request.

Scheduler automaton: It implements the scheduler and is shown in Fig. 7. It can register asynchronous requests from applications in between time samples via the synchronization channel *reqTT*. These requests are stored with the application *id* in *buffer0*. Besides, it also invokes a sequence of operations at every time sample based on a clock x which resets every time unit. This is done to implement a discrete-time scheduler and thereby addresses the first challenge. At each sample, the scheduler first increments the waiting time counter of each application *WT[id]* in *ET_Wait* state. *WT[id]* is used to reference the look-up table of minimum and maximum dwell times during the transition to the *TT* state. This addresses the second challenge. After updating the wait time counters, the scheduler automaton calls *Policy* and *Sort* if *buffer0* is not empty. After the requests are arranged in *buffer*, it checks if the slot is idle. When the slot is free and the buffer is not empty, the first application in the buffer gets the slot and the corresponding request is removed from the buffer. The application is simultaneously notified via the synchronization channel *getTT* and the clock *cT* is reset. On the other hand, if the slot is occupied then the scheduler checks if the allocated application can be preempted. Here, it uses the clock *cT* and the shared variable DT-[·] to check if the minimum dwell time has elapsed. If yes, the application is preempted via the synchronization channel *leaveTT[·]*. After an application is preempted, the free slot can be assigned to a waiting application (if any). After completion of all the operations, the clock x is reset.

Verification: The whole system is schedulable or the performance requirements of all applications will be met if no application reaches its *Error* state. Thus, the problem boils down to reachability analysis.

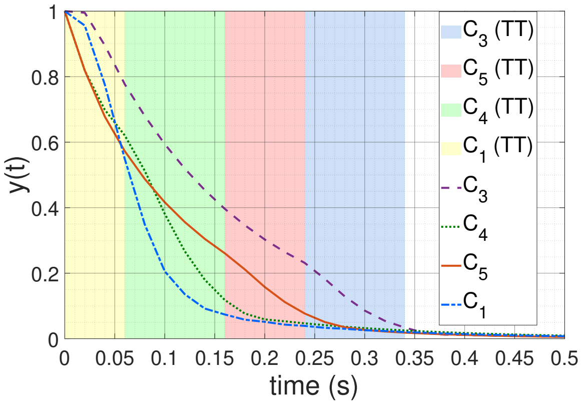


Figure 8: Responses of C_1, C_3, C_4 and C_5 using slot S_1.

5 EXPERIMENTAL RESULTS

Case Study: 6 control applications are considered. C_1 [13] and C_2 [10] are DC motor position control. C_3 [3], C_4 [10] and C_5 [12] represent DC motor speed control. C_6 [10] is a cruise control. The plant models are provided in Table 1. For $h = 0.02$s, controllers K_T and K_E (see Table 1) are designed considering the switching stability condition. For the given J^*, K_T and K_E, we can simulate the system to calculate J_T, J_E, T_w^*, T_{dw}^- and T_{dw}^+. All these results are shown in Table 1. Note that T_{dw}^- and T_{dw}^+ are arrays and array indices give T_w where $0 \leq T_w \leq T_w^*$. These arrays can be stored in a memory-efficient way exploiting the fact that T_{dw}^- and T_{dw}^+ take only a fewvalues.

Resource mapping: We apply the first-fit heuristic to map applications to slots. Applications are first sorted in the ascending order of T_w^*. When two applications have the same T_w^*, the one with a lower value of T_{dw}^{-*} is considered first (T_{dw}^{-*} is the maximum value of T_{dw}^- for $0 \leq T_w \leq T_w^*$). Accordingly, the applications are sorted as $\{C_1, C_5, C_4, C_6, C_2, C_3\}$. We start with one slot and map the first application in this list to the slot. Subsequently, we pick one application at a time in the sorted order and try to first map it to an existing slot. If performance verification fails, we assign a new slot for the application. To verify control performance of applications mapped onto a slot, we use UPPAAL (as described in Sec. 4). Applying this mapping algorithm, we obtain the following slot partitions: (i) $\{C_1, C_5, C_4, C_3\}$ mapped to slot S_1 and (ii) $\{C_6, C_2\}$ share slot S_2.

We also apply the two scheduling strategies proposed in [9] on the case study. The first strategy is similar to the standard non-preemptive deadline monotonic scheme. In the second strategy, the slot requests from the lower priority applications are delayed to reduce the blocking time for higher priority applications. Using the schedulability analysis and the first-fit heuristic proposed in [9], these applications require minimum 4 slots to meet their requirements. The slot partitions are $\{C_1, C_5\}$, $\{C_4, C_3\}$, $\{C_6\}$ and $\{C_2\}$. Thus, our proposed switching control strategy allows a tighter and accurate dimensioning of TT slots and *saves* 50% *slots*.

Simulation results: Using UPPAAL, we simulate the timed automata models representing two slot partitions for the following two cases: (i) Disturbances arrive simultaneously at C_1, C_3, C_4 and C_5. (ii) Disturbance arrives at C_6 10 samples after the disturbance at C_2. Using the obtained switching sequences, we simulate the control loops in MATLAB. The response curves for the two cases are shown in Fig. 8 and 9 respectively. The shaded regions indicate

Table 1: Case study data and results (Time is measured in no. of samples)

C_i	Plant Model	K_T	K_E	r	J^*	J_T	J_E	T_w^*	T_{dw}^-	T_{dw}^+
C_1	Eq.(6)	Eq. (7)	Eq. (8)	25	18	9	35	11	[3,4,3,3,3,3, 3,3,3,4,4,5]	[6,6,5,5,5,6, 5,5,4,4,5,5]
C_2	$\phi = \begin{bmatrix} 1 & 0.0117 & 0.0001 \\ 0 & 0.3059 & 0.0018 \\ 0 & -0.0021 & -1.2228 \times 10^{-5} \end{bmatrix}$, $\Gamma = \begin{bmatrix} 0.2966 & 24.8672 & 0.0797 \end{bmatrix}^T$, $C = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0.1198 \\ -0.0130 \\ -2.9588 \end{bmatrix}^T$	$\begin{bmatrix} 0.0864 \\ -0.0128 \\ -1.6833 \\ 0.4059 \end{bmatrix}^T$	100	25	15	50	13	[7,7,6,7,6,7,6, 7,6,7,6,7,7,8]	[10,10,9,10,8,9, 9,10,8,8,9,8,8,8]
C_3	$\phi = \begin{bmatrix} 0.9900 & 0.0065 \\ -0.0974 & 0.0177 \end{bmatrix}$, $\Gamma = \begin{bmatrix} 2.8097 \\ 319.7919 \end{bmatrix}$, $C = \begin{bmatrix} 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0.0500 \\ -0.0002 \end{bmatrix}^T$	$\begin{bmatrix} 0.0336 \\ 0.0004 \\ 0.4453 \end{bmatrix}^T$	50	20	10	31	15	[4,4,4,4,4,4,4,4, 4,4,4,4,4,4,4,4]	[8,8,7,7,7,6,6,6, 6,5,5,5,5,4,4,4]
C_4	$\phi = \begin{bmatrix} 0.8187 & 0.0178 \\ -0.0004 & 0.9608 \end{bmatrix}$, $\Gamma = \begin{bmatrix} 0.0004 \\ 0.0392 \end{bmatrix}$, $C = \begin{bmatrix} 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 100.0000 \\ 15.6226 \end{bmatrix}^T$	$\begin{bmatrix} -77.8275 \\ 24.3161 \\ 1.0265 \end{bmatrix}^T$	40	19	10	31	12	[5,5,5,5,5,5,5, 5,5,5,5,5,5]	[9,8,8,8,8,7,7, 7,7,6,6,6,5]
C_5	$\phi = \begin{bmatrix} 0.8187 & 0.0156 \\ -0.0031 & 0.7408 \end{bmatrix}$, $\Gamma = \begin{bmatrix} 0.0034 \\ 0.3456 \end{bmatrix}$, $C = \begin{bmatrix} 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 10.0000 \\ 1.0524 \end{bmatrix}^T$	$\begin{bmatrix} -2.4223 \\ 0.7014 \\ 0.2950 \end{bmatrix}^T$	25	18	10	25	12	[4,3,3,3,3,3,3, 4,4,4,4,4,4]	[9,8,7,8,7,6,7, 6,5,5,4,4,4]
C_6	$\phi = -0.999, \Gamma = 1.999 \times 10^{-5}, C = 1$	15000	$\begin{bmatrix} 8125.6 \\ 0.8659 \end{bmatrix}^T$	100	20	11	41	12	[7,8,7,8,7,8,7, 8,7,8,7,8,8]	[11,11,10,10,10, 10,9,9,9,8,8,8,8]

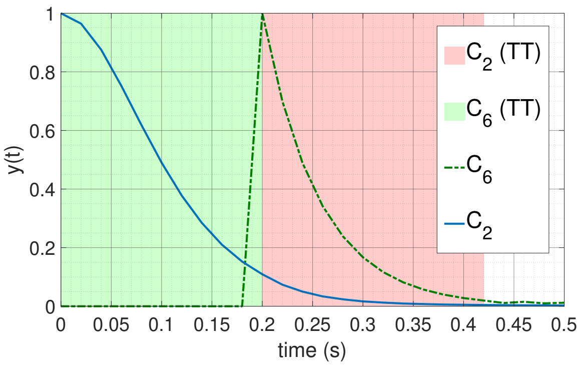


Figure 9: Responses of C_2 and C_6 using slot S_2.

the slot occupants. It can be verified that all applications meet their requirements. C_3 uses S_1 for $T_{dw}^+ = 5$ slots as there is no preemption while all others using S_1 are preempted at T_{dw}^-. C_2 and C_6 are not preempted and can achieve the maximum performance equal to J_T. C_2 uses only 10 TT samples to achieve a $J = J_T = 0.3$s while the conservative switching in [9] would require C_2 to stay in TT for 15 samples and still obtain the same performance.

Comments on verification time: In our case study, all except one verification took less than a minute. A particular case of mapping $\{C_1, C_5, C_4, C_3\}$ to one slot took close to 5 hours. However, it is possible to accelerate the verification if we do not consider infinite instances of disturbance. And for each application, we can calculate the maximum number of disturbance instances in other applications that can coincide with its disturbance. Accordingly, we can adapt the model and verify. With this approach, we can verify the mapping of $\{C_1, C_5, C_4, C_3\}$ in one slot within 15 minutes (i.e., a speed up of 20 times). We used a computer with Intel(R) Core(TM) i7 $-$ 5600U CPU @2.60 GHz processor and 8 GB RAM for the verification. Note that for the problem setting we study here, it may not be required to analyze too many applications in one slot. And moreover, since this whole process is offline, time is not the main constraint here.

6 CONCLUDING REMARKS

In this paper, we propose a resource-efficient switching control strategy for autonomous CPS exploiting heterogeneous platform architectures. The proposed strategy ensures that an application gets the high quality resource for a certain minimum time such that its performance requirement is met. To accurately provision for the high quality resources, we model the applications and the scheduling policy as timed automata. We apply model checking to verify that each application meet its control requirement. Although our proposed scheme can achieve tighter resource dimensioning, it does not optimize the average control performance. We preempt an application as soon as the minimum dwell time expires in case there is a waiting application. However, in certain cases, delaying the preemption might improve the performance of the current occupant of the high quality resource without degrading the performance of the waiting applications. In the future, we can investigate if machine learning techniques can improve the decision making while still guaranteeing a minimum control performance.

REFERENCES

[1] G. Behrmann, A. David, and K. G. Larsen. 2006. A Tutorial on UPPAAL 4.0. Retrieved Nov 24, 2018 from http://www.uppaal.org/

[2] W. Chang and S. Chakraborty. 2016. Resource-Aware Automotive Control Systems Design: A Cyber-Physical Systems Approach. *Foundations and Trends in Electronic Design Automation* 10, 4 (2016), 249–369.

[3] W. Chang, A. Pröbstl, D. Goswami, M. Zamani, and S. Chakraborty. 2014. Battery- and Aging-Aware Embedded Control Systems for Electric Vehicles. In *Real-Time Systems Symposium (RTSS).*

[4] A. David, J. Illum, K. G. Larsen, and A. Skou. 2009. Model-based framework for schedulability analysis using uppaal 4.1. Model-Based Design for Embedded Systems.

[5] Flexray Consortium. 2005. *The FlexRay Communications System Specifications.* Ver. 2.1.

[6] Z. Gu, Z. Wang, H. Chen, and H. Cai. 2014. A model-checking approach to schedulability analysis of global multiprocessor scheduling with fixed offsets. *International Journal of Embedded Systems* 6 (2014), 176–187.

[7] H. Lin and P. J. Antsaklis. 2009. Stability and Stabilizability of Switched Linear Systems: A Survey of Recent Results. *IEEE Trans. Automat. Control* 54, 2 (Feb. 2009), 308–322.

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

[9] A. Masrur, D. Goswami, S. Chakraborty, J.-J. Chen, A. Annaswamy, and A. Banerjee. 2012. Timing analysis of cyber-physical applications for hybrid communication protocols. In *Design, Automation and Test in Europe (DATE).*

[10] W. C. Messner and D. M. Tilbury. 1998. Control tutorials for MATLAB and Simulink: a web-based approach. Retrieved Nov 23, 2018 from http://ctms.engin.umich.edu/CTMS

[11] T. Pop, P. Pop, P. Eles, Z. Peng, and A. Andrei. 2008. Timing Analysis of the FlexRay Communication Protocol. *Real-Time Systems* 39, 1 (2008), 205–235.

[12] R. Schneider, D. Goswami, S. Zafar, M. Lukasiewycz, and S. Chakraborty. 2011. Constraint-driven synthesis and tool-support for FlexRay-based automotive control systems. In *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS).*

[13] N. Thomas and P. Poongodi. 2009. Position Control of DC Motor Using Genetic Algorithm Based PID Controller. In *World Congress on Engineering (WCE).*