Co-Design of Arbitrated Network Control Systems with Overrun Strategies

Damoon Soudbakhsh[1] Linh T.X. Phan[2] Anuradha Annaswamy[1] Oleg Sokolsky[2]
[1] *Department of Mechanical Engineering, Massachusetts Institute of Technology*
email: {damoon, aanna}@mit.edu
[2] *Computer and Information Science Department, University of Pennsylvania*
email: {linhphan, sokolsky}@cis.upenn.edu

***Abstract*— This paper addresses co-design of platform and control of multiple control applications in a network control system. Limited and shared resources among control and non-control applications introduce delays in transmitted messages. These delays in turn can degrade system performance and cause instabilities. In this paper, we propose an overrun framework together with a co-design to achieve both optimal control performance and efficient resource utilization. The starting point for this framework is an Arbitrated Network Control System (ANCS) approach, where flexibility and transparency in the network are utilized to arbitrate control messages. Using a two-parameter model for delays experienced by control messages that classifies them as nominal, medium, and large, we propose a controller that switches between nominal, skip and abort strategies. An automata-theoretic technique is introduced to derive analytical bounds on the abort and skip rates. A co-design algorithm is proposed to optimize the selection of the overrun parameters. A case study is presented that demonstrates the ANCS approach, the overrun framework and the overall co-design.**

I. Introduction and Related Work

Embedded computing systems (ECS) are ubiquitous in a wide range of applications including transportation, energy, and healthcare. Design of ECS faces several challenges, especially in the context of high performance, due to strict requirements such as safety, real-time deadlines, and minimum power consumption. These systems typically consist of several control and non-control applications in which their components communicate via shared resources. The presence of several applications with different priorities and limitations on the processing elements introduces resource contention. Specifically, messages can be occasionally delayed, arriving too late to be useful. It is therefore highly desirable to address the design of ECS that directly accommodates the presence of imperfect message transmissions, provides efficient resource utilization, and meets stringent performance specifications. This paper presents a co-design of implementation platform and control so as to result in efficient resource utilization and desired real-time control performance in the presence of overruns in messages when they do not meet their deadlines.

Existing research in co-design of control and implementation platform in the presence of non-ideal message transmissions can be categorized in two parts: i) Designing controllers to achieve desired performance and ii) Designing communication protocols to achieve efficient resource utilization. The former consists of procedures for the estimation of worst-case delays of messages and design of controllers that are robust to such delays. This approach however can be pessimistic and often leads to inefficient control performance or resource utilization. The latter consists of defining a deadline for messages and employing a switching control strategy that depends on this deadline; if the message does not exceed the deadline, a nominal controller is employed, and if it does, the message is aborted. A slight but important variation of the abort strategy is to skip the next message rather than aborting the current message, so as to free up resources at the next instant. Such skip and abort messages lead to different dynamic characteristics as well as different implications on the efficiency of resource utilization. This paper proposes an overrun framework that includes a nominal-skip-abort control strategy which switches between three modes depending on the delay that all messages experience. Based on this switching strategy, we introduce an automata-theoretic technique to derive analytical bounds on the abort and skip rates that an application experiences on the platform, and a numerical algorithm for the co-design of platform and control that utilizes the platform analysis and switching control design to achieve satisfactory control performance and efficient resource utilization.

The problem of control and platform design in the presence of non-ideal transmissions of closed-loop messages has been the focus of several investigations (see, for example, [1]–[23]). In the control systems domain, the main approach used is to design the controller based on the estimation of worst-case delays (ex. [1]–[6]). Although such approaches can improve over a baseline design that completely neglects any implementation delays, they still introduce a stringent constraint on the platform resources to guarantee such delays for all messages. Designing the controllers based on messages with worst-case delays often leads to inefficient performance during normal operation of the system, as such messages may occur rarely. Alternatively one can use scheduling techniques and corresponding resources to design desired delays that each message can experience [7]. This however may result in very high implementation costs. Furthermore, it may be impossible to design a system based on worst case delays, as the latter may be unbounded depending on the network configuration [5].

An abort strategy that drops any message whose arrival exceeds a specified deadline has been explored in the literature as well [5], [6], [8], [11]–[14]. This leads to a switched control system whose stability has been analyzed using Multiple Lyapunov Functions [5], [11], Norm-based approaches [11], and common quadratic Lyapunov functions [6], [8], [12], [13].

Modifying the control law for consecutive aborted signals to improve the overall performance has been studied in [12], [13], [15]–[17]. While all of these methods based are improvements over those based on worst-case delays, and can be proved to be stable, they may still lead to inefficient resource utilization. This is because all messages that are aborted have to be computed until the specified deadline thereby wasting resources over this period.

In contrast to the abort strategy, a skip strategy, which consists of dropping the next message when the current message exceeds a deadline, has been explored to a much lesser extent [18]–[20]. The skipping is implemented in some of these papers by doubling the sampling period of the next message when the current message exceeds the deadline. As mentioned earlier, the skip strategy has a direct advantage over the abort one in terms of resource utilization. When messages experience inordinately large delays, the skip strategy has obvious shortcomings, necessitating a careful stability analysis.

The above discussions clearly imply that a combined skip-abort strategy for messages with overruns together with an optimal set of parameters that characterize abort and skip conditions is desirable. Although control and platform design with overruns have been considered previously (see e.g., [21]–[23]), most existing research assumes either zero or a factor of the sampling time as the deadline of samples in their control design, and both the deadline and the number of deadline misses in a given window are given a priori in the platform analysis. To the best of our knowledge, our work is the first to provide a constructive approach to determine these parameters.

This paper proposes a co-design framework where transparencies and flexibilities in the implementation network, and connections between controllers and implementation can be accommodated, leading to an Arbitrated Network Control System [6]. Our main contributions can be summarized as follows:

- a nominal-skip-abort control strategy based on two delay threshold parameters to enable efficient resource use;
- an automata-theoretic approach for modeling the platform and for analyzing the maximum number of skip and abort samples under the proposed control strategy;
- an expanded dynamic model of the plant for each of the nominal, abort, and skip cases, and a stability analysis of the resulting switched system; and
- a co-design algorithm that optimizes the threshold parameters to result in efficient resource utilization and desired control performance.

Our evaluation results show that the proposed co-design with skip and abort can help save resource by an order of magnitude compared to a nominal co-design approach while still ensuring the desirable control performance.

This paper is organized as follows. Section II introduces the problem statement including the platform architecture and the dynamic model of the plants to be controlled. Before presenting the overrun framework, we first present the nominal case in §III when all messages meet their deadlines, and discuss both the platform design and the control design. The overrun framework is introduced in §IV, and a two-parameter model to represent message deadlines is presented. Section V presents the corresponding control designs and stability results for the closed-loop system. Section VI includes the platform analysis for the overrun framework. We present the co-design algorithm in §VII, which is evaluated through a case study with six applications in §VIII. Concluding remarks are given in §IX.

II. System Model and Problem Statement

Before stating the co-design problem, we first present the models of the platform and the control applications.

A. Platform Architecture

The typical platform we consider consists of a set of processing elements (PEs) connected via FIFO buffers, where each PE represents a processor (e.g., ECU) or a network (e.g., CAN bus). Each PE processes one or more tasks of control and non-control applications in the network[1]. The end-to-end **delay** of a sample, τ, is defined as the duration from the instant the sample arrives at the system until it is fully processed.

Figure 1 shows an example of the platform architecture with four ECUs and a CAN bus. This platform processes three control applications and a non-control application. In the figure, $\{T_1, T_2, T_3\}$, $\{T_4, T_5, T_6\}$, and $\{T_7, T_8, T_9\}$ are the sets of tasks of the first, second, and third control application, respectively, whereas T_{10} is the task of the non-control application.

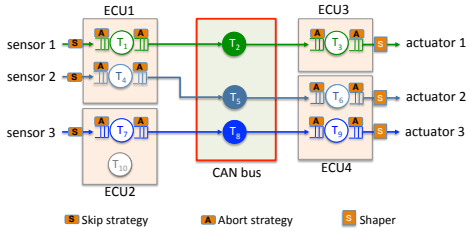


Fig. 1: An example of the platform architecture for the control applications. The shaper is discussed in §III-C and the skip and abort strategies are discussed in §IV.

Upon arriving at the system, the sensor input data items (samples) of each control application will be processed on a sequence of PEs before being used to actuate the physical plant. As an example, in Figure 1, the sensor data of the first application (produced by the sensor 1) will first be processed by T_1 on ECU1, whose output will then be transmitted on the bus via the message T_2 to ECU3. Upon arriving at ECU3, the data will then be processed by T_3, and the final output data will then be used by the actuator 1. The end-to-end delay of a sample in this example is the time duration from the instant it arrives at ECU1 until the instant it leaves ECU3.

We assume that the processors schedule their tasks using a fully preemptive fixed-priority scheduling algorithm (e.g., Rate Monotonic [24]), whereas the network schedules its messages according to a non-preemptive fixed-priority algorithm (as is the case for CAN bus). The sampling period and the priority of each application are assumed to be given a priori. All tasks of an application share the same priority as that of the application, and their worst-case execution demands are given a priori. In addition, the sizes of the buffers are set to be sufficiently large,

[1] In this paper, we use the term 'task' to indicate either a compute function executed on a processor or a message transmitted on a network. Further, we define the *execution demand* of a task to be the execution time of the task in the former case and the message size in the latter case.

e.g., equal to the maximum buffer sizes computed using the method in [25], to avoid buffer overflows.

B. Control Applications

The problem considered here is the control of n applications, whose plant models are assumed to be of the form

$$\dot{x}_i(t) = A_i x_i(t) + B_i u_i(t - \tau_i), \tag{1}$$

where $x_i(t) \in \Re^p$ and $u_i(t) \in \Re^q$ are the states and inputs of the system, respectively, and (A_i, B_i) are controllable for all $i = 1 : n$. Arbitrary delays τ_i occur due to shared resources. The problem is to carry out a CPS co-design and choose $u(t)$ so that $x(t)$ tends to zero asymptotically for all n plants, while consuming minimal resources in the implementation platform. For ease of exposition, we set

$$A_c \stackrel{\text{def}}{=} A_i, \quad B_c \stackrel{\text{def}}{=} B_i, \text{ and } \quad \tau \stackrel{\text{def}}{=} \tau_i, \quad i = 1 : n \tag{2}$$

and denote the corresponding state and input as x and u, respectively. Extension to the case when the plant dynamics varies with i is relatively straightforward. For ease of exposition, we assume that $\tau \leq h$, the sampling time. The corresponding sampled data model is given by

$$x[k+1] = Ax[k] + B_{11}(\tau)u[k] + B_{12}(\tau)u[k-1], \tag{3}$$

where

$$A \stackrel{\text{def}}{=} e^{A_c h}, \; B_{11}(\tau) \stackrel{\text{def}}{=} \Big(\int_0^{h-\tau} e^{A_c v} dv\Big) B_c, \; B_{12}(\tau) \stackrel{\text{def}}{=} \Big(\int_{h-\tau}^{h} e^{A_c v} dv\Big) B_c.$$

C. The End-to-End Delay τ

The main focus of this paper pertains to τ, its implications on control performance, and its dependence on the platform architecture. As mentioned in §II-A, τ is the duration from the instant the message arrives at the first task to the instant it is fully processed by the last task of the application. The fact that there are several control and non-control applications that have to be processed by the platform imply that this delay τ is (a) non-negligible, and (b) can vary significantly. In the rest of the paper, we address this aspect of the delay and show that by a co-design of the controller and the platform, the desired QoC can be met with the available platform resources.

The final point to note regarding the delay is the delay τ_p due to actuator dynamics. While in general τ, the end-to-end delay from a plant output to the plant input should include τ_p as well, for ease of exposition, we set $\tau_p = 0$. An extension to $\tau_p \neq 0$ is relatively straight forward.

III. Nominal Co-Design

The problem is the stabilization of n plants given by (3) in the presence of a non-zero delay τ. We focus in this section on the nominal case, which is defined as the case when $\tau \leq \tau_{th}$, where τ_{th} is a value that is small enough for closed-loop control to be effective. In §III-A, we describe the platform analysis for this case and in §III-B, we propose a nominal control design for this case, based on a linear-quadratic regulator.

A. Nominal Platform Analysis

In the nominal case, the control design requires that every sample of an application must be fully processed by the platform within the delay threshold τ_{th} of the application. We briefly describe how this feasibility condition can be analyzed using the Real-Time Calculus (RTC) method [25], [26].

In the RTC method, the arrival pattern of the input data stream of a task is modeled using a pair of arrival functions, (α^u, α^l), where $\alpha^u(\Delta)$ and $\alpha^l(\Delta)$ specify the maximum and minimum number of data items that arrive at the task's buffer over any interval of length Δ, for all $\Delta \geq 0$. Similarly, the resource availability of a PE can be modeled using a pair of service functions, (β^u, β^l), where $\beta^u(\Delta)$ and $\beta^l(\Delta)$ specify the maximum and minimum number of items that can be processed over any interval of length Δ, for all $\Delta \geq 0$.

Based on the models of the input data streams and resource availability of the PEs, we can compute the maximum bound on the end-to-end delay of an application in a compositional manner. For example, the end-to-end delay of the first application shown in Figure 1 is bounded above by the sum of the maximum processing (transmission) delays of its tasks, i.e.,

$$\tau_{wc} = d_1 + d_2 + d_3, \tag{4}$$

where d_1, d_2, and d_3 are the worst-case delays of T_1, T_2, and T_3, respectively. By definition, the upper arrival function of the input data of T_1 is $\alpha_1^u(\Delta) = \lceil \Delta / h \rceil$. Since T_1 is the highest-priority task, ECU1 first provides all its available resource to T_1 and only gives the remaining to the lower-priority tasks. As the ECU provides Δ execution units over any interval of Δ time units, the lower service function of the resource available to T_1 is $\beta_1^l(\Delta) = \lfloor \Delta / \mathsf{wcet}_1 \rfloor$, where wcet_1 is the worst-case execution demand of T_1. Then, d_1 is given by [25], [26]:

$$d_1 \stackrel{\text{def}}{=} \sup\big\{\inf\{\tau \geq 0 \mid \alpha_1^u(t) \leq \beta_1^l(t+\tau)\} \mid t \geq 0\big\}. \tag{5}$$

Further, the arrival function of the output data of T_1, which is also the input arrival function of T_2, is given by [25], [26]: $\alpha_2^u = ((\alpha_1^u \otimes \beta_1^u) \oslash \beta_1^l) \oplus \beta_1^u$, where $(f \otimes g)(t) = \inf_{0 \leq s \leq t}(f(s) + g(t-s))$, $(f \oslash g)(t) = \sup_{u \geq 0}(f(t+u) - g(u))$, and $\beta_1^u(\Delta)$ is the upper service function of the resource available to T_1. Based on α_2^u and resource availability of the CAN bus, we compute the maximum transmission delay d_2 of T_2 and the input arrival function α_3^u of T_3 in the same fashion. Similarly, the maximum delay d_3 of T_3 can then be computed based on the arrival function α_3^u.

Based on the results in [26], we can also derive the service functions of the remaining resource after processing T_1, T_2 and T_3 on ECU1, the bus, and ECU3, respectively. These service functions are then used to compute the worst-case delays of the tasks of the next highest-priority application, and so on.

Depending on whether the end-to-end delay of each application is always less than or equal to the application's threshold τ_{th}, we can then determine whether the platform is feasible for the applications. If it is not, the platform resource will be increased in an iterative manner until all applications have their end-to-end delays within their respective thresholds τ_{th}.

B. Nominal Control Design

We now derive a control design for the plant in (3) that explicitly accommodates τ with the assumption that $\tau_{th} = \tau_{wc}$. For the sake of analytical tractability, we assume that $\tau < h$ is a constant. By defining an extended state $X(k) = [x^T[k], u^T[k-1]]^T$, plant (3) can be written as

$$\begin{aligned} X[k+1] = & \begin{bmatrix} A & B_{12} \\ 0 & 0 \end{bmatrix} X[k] + \begin{bmatrix} B_{11} \\ I \end{bmatrix} u[k] \\ \stackrel{\text{def}}{=} & \Gamma_0(\tau,h)X[k] + \Theta_0(\tau,h)u[k], \end{aligned} \quad (6)$$

Equation (6) suggests that a state feedback controller in the form of

$$u[k] = K_0 x[k] + G_0 u[k-1] \quad (7)$$

can stabilize the system. The closed-loop system is then given by

$$X[k+1] = \begin{bmatrix} A + B_{11}K_0 & B_{12} + B_{11}G_0 \\ K_0 & G_0 \end{bmatrix} X[k] \stackrel{\text{def}}{=} \Gamma_n X[k]. \quad (8)$$

The controller in (7) is chosen so as to minimize a quadratic cost function J_n as

$$\min.\ J_n \stackrel{\text{def}}{=} \sum_0^\infty (X[k]^T Q X[k] + u[k]^T R u[k]), \quad (9)$$

where Q and R are the weighting matrices on augmented states and inputs, respectively. The optimal gain $K = [K_0 \quad G_0]$ is derived by solving the following discrete-time Riccati equation for a positive definite matrix $P_0 \succ 0$

$$\Gamma_0^T P_0 \Gamma_0 - P_0 - \Gamma_0^T P_0 \Theta_0 (\Theta_0 P_0 \Theta_0 + R)^{-1} \Theta_0^T P_0 \Gamma_0 + Q = 0 \quad (10)$$

and using the following relation

$$K = (\Theta_0^T P_0 \Theta_0 + R)^{-1} (\Theta_0^T P_0 \Gamma_0). \quad (11)$$

C. Implementation of the Nominal Co-design

The discussions in the above two sections imply that as long as $\tau_{wc} < h$, a control design can be carried out as in (7) for the plant in (6), where $\tau = \tau_{wc}$. The platform resources therefore have to be such that τ_{wc} computed using (4) does not exceed h. Any time-variations in τ between $(0, \tau_{wc})$ can be accommodated by using the shaper shown in Figure 1. By locating the shaper at the last *PE* and having it hold every fully processed sample for exactly $\tau_{th} - \tau$ time units before sending to the actuator, we can ensure that the sensor-to-actuator delay of each sample is always τ_{th}.

IV. An Overrun Framework

The implicit assumption made for the nominal co-design discussed in §III is that τ_{wc} is small compared to the sampling period, which is valid only when there are sufficient platform resources. In addition, the derivation of τ_{wc} was conservative, which implies that messages that actually experience a delay of τ_{wc} are rare. We therefore address in this section the possibility that τ varies, and allow some of the messages to be overrun, i.e. $\tau < \tau_{th}$ for some messages, and $\tau > \tau_{th}$ for others. The overrun framework proposed includes the delineation of overrun strategies with two parameters, control designs based on these strategies, stability guarantee, and a co-design that ensures desired QoC and minimal resource utilization.

A. Overrun framework with two delay-parameters

In this framework, we assume that there are two parameters τ_{th1} and τ_{th2}, where τ_{th2} is a value close to the worst-case upper bound that all delays are expected not to exceed, while τ_{th1} is an average value of delay experienced by messages. We consider three possible cases,

A1. Nominal: $\tau \leq \tau_{th1}$: That is, the message has a delay less than the threshold τ_{th1}.

A2. Skip: $\tau_{th1} < \tau \leq \tau_{th2}$: Here the computation of the control input at the next instant of time is skipped.

A3. Abort: $\tau > \tau_{th2}$: The computation of the current control input is aborted.

B. Implementation of the overrun framework

We now describe how the two-parameter frameworks can be implemented for each control application C_i executing on the platform. Let $T_1, \cdots, T_n$ be the sequence of tasks of the application. Further, let PE_i the processing element that processes T_i, for all $i = 1:n$. As an example, in Figure 1, the three tasks T_1, T_2 and T_3 of the first control application are processed by PE_1, PE_2, and PE_3. Here, PE_1 is ECU1, PE_2 is the CAN bus, and PE_3 is ECU3.

To implement the overrun strategy, we introduce a buffer control mechanism that proactively removes data items from the buffers based on their current delays, as follows:

- If the current delay of a fully processed data item in the output buffer of T_n is less than τ_{th1}, the item will be delayed by a shaper until its delay reaches exactly τ_{th1}; this corresponds to the nominal case (A1).
- If the delay of a fully processed data item in the output buffer of T_n is larger than τ_{th1} but less than or equal to τ_{th2}, then PE_n will send a notification message to PE_1, informing PE_1 to immediately discard the next sensor data item as soon as it arrives at the input buffer of T_1. This implements the skip strategy (A2).
- For each T_i, $i = 1:n$, if the current delay of a data item in the input buffer of T_i is equal to τ_{th2}, the item will be removed from the buffer[2]; this implements the abort strategy (A3),

The skip and abort strategies at the buffers, as well as the shaper in the nominal case are illustrated by the Abort, Skip, and Shaper blocks in Figure 1 through the orange blocks marked S, A, and S (in white letter), respectively.

Remark 1. *Since the removal actions and notification messages of the buffer control mechanism always have higher priority than the tasks and messages of the applications, their run-time overhead can be incorporated into the analysis.*

We note that this buffer control mechanism helps improve the resource use efficiency in two ways. First, since not all data items need to be processed, the amount of computation and communication resource required by each control application is reduced compared to the conventional platform design approach, where all data items must be fully processed. Second, since the platform discards data items as soon as they are not needed by the control application, the amount of resource

[2]Note that the end-to-end delay of this data item will always be larger than τ_{th2}.

needed to further process these data items can be saved or used to process other applications.

V. Control Designs for a Two-Parameter Overrun Framework

Using an *Abort Only* strategy may need very large thresholds in order to avoid excessive drops, leading to a conservative control performance. On the other hand, using skip strategy alone demands the system to accommodate worst case execution that may be large and rare, and introducing unnecessary wait times for useless and potentially destabilizing messages. In such scenarios, it is more efficient to use a two-parameter overrun framework which takes advantage of both strategies, outlined in §IV-A.

Starting with two threshold parameters τ_{th1} and τ_{th2} with $\tau_{th1} \le \tau_{th2}$. Cases A1, A2, and A3 are invoked as described in §IV-A. That is, if $\tau \le \tau_{th1}$, the messages belong to the nominal case. If $\tau_{th1} < \tau \le \tau_{th2}$, then the skip strategy is employed and we set

$$u[k+1] = u[k], \tag{12}$$

If $\tau > \tau_{th2}$, computation of $u[k]$ is aborted and $u[k]$ is set to a previously computed value. That is at any time k, if the delay τ continues to be larger than τ_{th2} for j consecutive instants, with $\tau < \tau_{th1}$ at $k-1$, or $\tau_{th1} < \tau \le \tau_{th2}$ at $k-2$, then it follows that

$$u[k+\ell] = u^*[k-1], \qquad \ell = 0 : j-1. \tag{13}$$

where $u^*[k-1]$ is a previously computed value.

In what follows, we discuss the underlying dynamics in all three cases and derive the corresponding control strategy. A summary of these cases can be found in Table I.

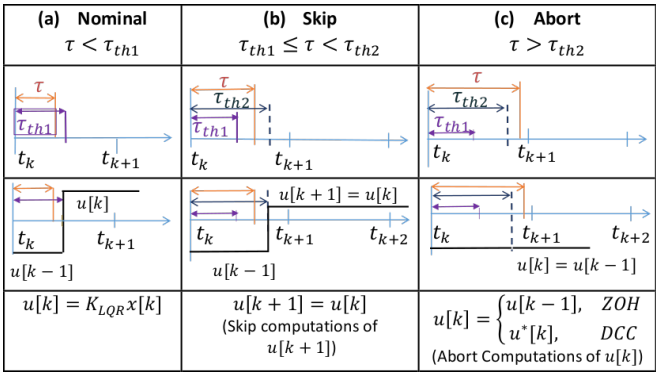


TABLE I: Two-parameter overrun framework: i) Nominal $\tau < \tau_{th1}$, ii) Skip $\tau_{th1} \le \tau < \tau_{th2}$, and iii) Abort $\tau > \tau_{th2}$, $u^*[k]$ is a function of previous states and inputs.

A1. Nominal mode, $\tau \le \tau_{th1}$ (see Table Ia): the dynamics is given by (6) with $\tau = \tau_{th1}$.

A2. Skip mode $\tau_{th1} < \tau \le \tau_{th2}$ (see Table Ib): a skip at k results in no new inputs at interval $[t_{k+1}, t_{k+2}]$. Therefore, the input at $[t_k, t_{k+1}]$ directly affect the dynamics at $k+2$ with the resulting dynamics

$$x[k+2] = Ax[k+1] + B_1 u[k]. \tag{14}$$

Noting that input $u[k]$ had arrived with a corresponding delay $\tau > \tau_{th1}$, the state at $k+1$ can be computed as

$$x[k+1] = Ax[k] + B_{22}u[k-1] + B_{21}u[k]. \tag{15}$$

where

$$B_{21} \stackrel{\text{def}}{=} \left(\int_0^{h-\tau_{th2}} e^{A_s v} dv\right) B_c, \text{ and } B_{22} \stackrel{\text{def}}{=} \left(\int_{h-\tau_{th2}}^{h} e^{A_s v} dv\right) B_c.$$

Using (15), augmented state $X[k] = [x^T[k], u^T[k-1]]^T$, and (7), we can write the underlying dynamics of A2

$$X[k+2] = \begin{bmatrix} A^2 + (AB_{21} + B_1)K_0 & AB_{22} + (AB_{21} + B_1)G_0 \\ K_0 & G_0 \end{bmatrix} X[k] \stackrel{\text{def}}{=} \Gamma_s X[k] \tag{16}$$

A3. Abort mode, $\tau > \tau_{th2}$ (see Table Ic): can happen after a nominal or a skip and results in aborting the computations of $u[k]$ and using a previously computed value instead. That is at any time k, the delay τ continues to be larger than τ_{th2} for j consecutive instants, with $\tau < \tau_{th2}$ at $k-1$, then it follows that

$$u[k+\ell] = u^*[k], \qquad \ell = 0 : j-1. \tag{17}$$

where $u^*[k]$ is a previously computed value. For example, if a standard zero order hold is used, with $j = 1$

$$u^*[k] = u[k-1]. \tag{18}$$

With such an abort strategy as in (17), and the closed-loop dynamics is given by

$$X[k+1] = \begin{bmatrix} A & B \\ 0 & I \end{bmatrix} X[k] \stackrel{\text{def}}{=} \Gamma_a X[k] \text{ for } k = k_1 + \ell. \tag{19}$$

Alternately, $u^*[k]$ can also be derived using other compensation strategies (see [12]).

The above discussions indicate that the underlying plant dynamics is given by (6) in the nominal mode, (16) in the skip mode, and (19) in the abort mode.

Suppose, in general, starting at k, there are i_ℓ instants of nominals, followed by j_ℓ skip instants (each with a length of $2h$), followed by r_ℓ instants of aborts, for $\ell = 1 : p$, with

$$n_n \stackrel{\text{def}}{=} \sum_{\ell=1}^{p} i_\ell \quad m_{sk} \stackrel{\text{def}}{=} \sum_{\ell=1}^{p} j_\ell \quad m_{ab} \stackrel{\text{def}}{=} \sum_{\ell=1}^{p} r_\ell \tag{20}$$

then the evolution of the composite switched system over a time window $[k, k+N]$, $N = 2m_{sk} + m_{ab} + n_n$, is given by

$$X[k+N] = \Gamma_a^{s_p} \Gamma_s^{j_p} \Gamma_n^{i_p} \cdots \Gamma_a^{r_1} \Gamma_s^{j_1} \Gamma_n^{i_1} X[k] \tag{21}$$

In addition, suppose that sufficient information is available about the implementation platform such that in an interval of N samples, m_{sk0}, upper-bound on the number of skipped messages and m_{ab0} upper-bound on the number of aborted messages exist, that is

$$m_{sk} \le m_{sk0}, \quad m_{ab} \le m_{ab0}, \quad \text{and} \quad n_n \ge n_{n0},$$

where $n_{n0} \stackrel{\text{def}}{=} N - 2m_{sk0} - m_{ab0}$, and m_{sk0} and m_{ab0} are known.

We now state and prove the stability of the switched system in (21) in Theorem 1. The following definition is useful:

$$\bar{\alpha}_{overall}^{-2}(m_{sk0}, m_{ab0}, N) \stackrel{\text{def}}{=} \gamma_a^{m_{ab0}} \cdot \gamma_s^{m_{sk0}} \cdot \gamma_n^{n_{n0}}. \tag{22}$$

where γ_n, γ_s, and γ_a are parameters determined in Theorem 1.

Theorem 1. *System* (21) *is stable (exponentially stable) if there exist positive definite matrix $P \succ 0$, and positive scalars $\gamma_n < 1$, and $\gamma_s, \gamma_a > 0$ such that the following LMI*

$$\begin{bmatrix} -\gamma_n P & * \\ P\Gamma_n & -P \end{bmatrix} \prec \mathbf{0}, \tag{23}$$

$$\begin{bmatrix} -\gamma_s P & * \\ P\Gamma_s & -P \end{bmatrix} \prec \mathbf{0}, \tag{24}$$

$$\begin{bmatrix} -\gamma_a P & * \\ P\Gamma_a & -P \end{bmatrix} \prec \mathbf{0}, \tag{25}$$

and

$$\bar{\alpha}_{overall}^{-2}(m_{sk0}, m_{ab0}, N) \leq 1 (< 1). \tag{26}$$

are satisfied. $\bar{\alpha}_{overall}(m_{sk0}, m_{ab0}, N)$ is a lower bound on the exponential decay rate of signals over interval of N samples.

Proof. See Appendix A. □

Corollary 2. *When there are no overruns, i.e. $m_{sk0} = m_{ab0} = 0$,*

$$\bar{\alpha}_{overall} = \gamma_{n0}^{-0.5N} \tag{27}$$

where γ_{n0} is the solution γ_n in (23)*.*

We define a normalized decay rate $\alpha_{overall}$ as $\bar{\alpha}_{overall}^{1/N}$, which can be shown to be independent of the observation window. Defining r_{skip}^p and r_{abort}^p the allowable skip and abort rates specified by the platform as

$$r_{skip}^p \stackrel{\text{def}}{=} \frac{m_{sk0}}{N} \tag{28}$$

$$r_{abort}^p \stackrel{\text{def}}{=} \frac{m_{ab0}}{N}, \tag{29}$$

it is easy to see that

$$\alpha_{overall}^{-2}(r_{skip}^p, r_{abort}^p) = \gamma_n^{1-2r_{skip}^p - r_{abort}^p} \gamma_s^{r_{abort}^p} \gamma_a^{r_{skip}^p}. \tag{30}$$

Equation (30) implies that the augmented states of the system decay at a rate greater than $\alpha_{overall}(r_{skip}^p, r_{abort}^p)$. A measure for quality of control can be defined based on this value as

$$J_c = \alpha_{overall}^{-2}(r_{skip}^p, r_{abort}^p). \tag{31}$$

From (30) in Theorem1, it follows that the control design is not implementable if $\alpha_{overall} < 1$. This will be ensured by defining a desired $\alpha^* > 1$, and requiring the stronger condition that $\alpha_{overall} > \alpha^*$ for the control design to be feasible.

A summary of the overall control design for the two-parameter overrun framework include the following steps:

1) Given a τ_{th1} and τ_{th2}, find the nominal control gains $K = [K_0 \quad G_0]$ by solving (10) for P_0 and using (11). This results in the closed-loop dynamics (8).
2) Construct the system dynamics in the skip mode (eq.(16)).
3) Construct the system dynamics in the abort mode (eq. (19)), if ZOH strategy is used; a similar equation can be derived if the DCC method is used instead.
4) Given m_{sk0}, m_{ab0}, and N, compute parameters $(\gamma_n, \gamma_s, \gamma_a)$ of the system in each mode by solving the LMIs (23)-(25).
5) Compute actual maximum drop rates $(r_{skip}^p, r_{abort}^p)$ as in (28)-(29), and the normalized overall decay rate, $\alpha_{overall}$ as in (30).

The stability result in Theorem 1 can be extended to general nonlinear systems in the presence of nominal, skip, and abort modes, with maximum skip and abort rates. This is stated in Corollary 3.

Corollary 3. *Suppose that the underlying switching nonlinear dynamics is given by*

$$x[k+1] = \begin{cases} f_{nominal}(x[k], \tau) & \text{for } \tau \leq \tau_{th1} \quad (32a) \\ f_{skip}(x[k], \tau) & \text{for } \tau_{th1} < \tau \leq \tau_{th2} \quad (32b) \\ f_{abort}(x[k], \tau) & \text{for } \tau > \tau_{th2}. \quad (32c) \end{cases}$$

System (32) *is stable if*

1) *There exist Lyapunov-Like Functions [11], [27] $V_i(x[k]) > 0$ for nominal, skip, and abort modes of the system.*
2) *There exist a window of N consecutive samples such that*

$$V_i(x(k_1+N)) - V_j(x(k_1)) < 0, \qquad \forall k_1, i, j, \tag{33}$$

VI. Platform Analysis under Overrun Semantics

In this section, we introduce an automata-theoretic technique for analyzing the maximum long-term abort and skip rates that an application experiences on the platform, for a given resource availability, under the implementation strategy described in Section VI-A1. For this, we first present the automata model of the platform. We then discuss how automata verification can be used to derive the maximum number of aborts m_{ab0} and skips m_{sk0} experienced by an application within a sliding window of length N, for any given pair of delay thresholds (τ_{th1}, τ_{th2}) with $0 \leq \tau_{th1} \leq \tau_{th2} \leq h$. The long-term abort (skip) rate can then be bounded by the ratio of the maximum number of drops (skips) within the sliding window to the window size, and their results are used in each step of the exploration of threshold parameters in the co-design algorithm. Typically, a larger window size leads to tighter abort and skip rates but longer analysis time. One possibility is to choose the smallest window size such that the corresponding abort (skip) rates do not decrease as the window size increases; however, our analysis is safe under any window size.

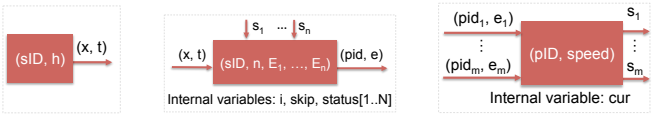
A. Automata-theoretic modeling of the platform

The platform can be modeled in a compositional manner as a composition of three basic components, as shown in Figure 2:

- *Sensor*: models the generation of the sensor data stream of an application;
- *Application*: models the task processing of an application, according to the overrun semantics;
- *PE*: models the amount of resource that a PE provides to each connected application based on its scheduling policy.

We first explain the interfaces of these components, and then present the automata models of their internal semantics.

As an example, Figure 3 shows the model of part of the platform that processes the highest-priority application App_1 of the architecture shown in Figure 1, which is formed by connecting the *Sensor*, *Application* and *PE* components of the application based on the components' interfaces.



(a) *Sensor*. (b) *Application*. (c) *PE*.

Fig. 2: Basic components of the platform.

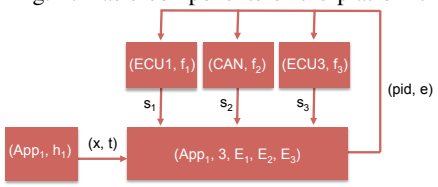


Fig. 3: A composition of platform components (concerning App_1).

1) Interfaces of the basic components

As shown in Figure 2, the *Sensor* component is characterized by two parameters, sID and h, which denote the identification and sampling period of the corresponding application. It has two output variables, (x,t), where x denotes the number of new data items the component generates, and t is their discrete time stamp. These data items serve as inputs to the corresponding *Application* component.

The *Application* component is characterized by: sID, the application identification; n, the number of tasks; and E_k, the worst-case execution demand of the k^{th} task, for all $k = 1:n$. It has two types of inputs: (1) the variables (x,t) produced by the *Sensor* of the application; and (2) the amount of resource provided to task k by the PE that executes this task, for all $k = 1:n$. In addition, the *Application* has three internal variables:

- i: the index of the task that is processing the current data sample of the application. Note that, since $\tau_{th2} \leq h$ and all unfinished items with current delays equal to τ_{th2} will be aborted, at most one sample of each application is in the platform at any time. We refer to this item as the *current item*, and we refer to the task that is currently processing this item as the *current task* of the component;
- skip: a binary value denoting whether the next data sample should be skipped (skip $= 1$) or not (skip $= 0$);
- status$[1..N]$: an array of N elements representing the status of N most recent samples (including the current one) of the application. Specifically, status_l takes value 0, 1, or 2 if the l^{th} most recent data item is processed successfully, skipped, or aborted, respectively.

Finally, the component has two output variables, pid and e, which represent the identification of the PE executing the current task and the task's remaining execution demand.

The *PE* component is characterized by the identification, pID, and speed of the corresponding PE. It has m input ports that are connected to m *Application* components that execute on this PE, with *Application* j having higher priority than *Application* $j+1$, for all $j = 1:m-1$. Thus, the PE only provides service to *Application* j when $\mathsf{pid}_j = \mathsf{pID}$. Finally, for each input port j of the PE, there is a corresponding output port that is associated with the output variable s_j, which denotes the amount of service (in terms of execution time units) available to the corresponding *Application* component. When the PE implements a non-preemptive fixed-priority scheduling policy, the component also has an internal variable, cur, which represents the index of the task that was processed but has not yet been completed in the previous time unit.

2) Semantics of the Sensor component

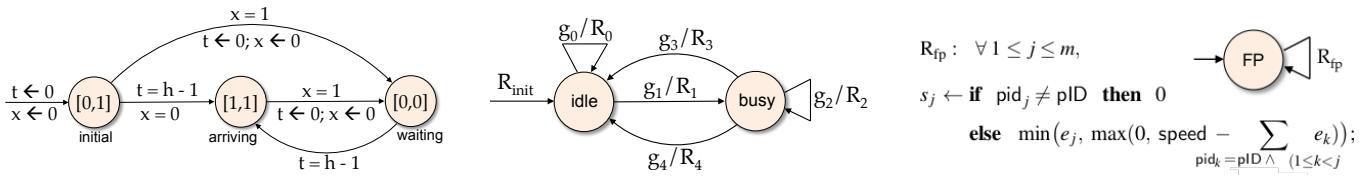
The semantics of a *Sensor* component is captured by an Event Count Automata (ECA) [28] extended with a clock variable, which is shown in Figure 4(a). This ECA has a single count variable, x, which counts the number of data items that are generated by the automaton since the last time x was reset. Each state of the automaton is associated with a rate vector, $[l,u]$, where l and u denote the minimum and maximum number of data items that are generated by the automaton in each unit of time when the automaton is in this state. For instance, while the automaton is in the initial state, which is associated with the rate vector $[0,1]$, it generates 0 to 1 data item in each time unit. Each transition is associated with a guard on the count variables or the clock variable, which specifies the condition under which the transition is enabled. In addition, it may also be associated with a reset of the variables, which takes place when the transition is taken.

The ECA in Figure 4(a) describes the arrival of the sensor samples of an application. The first sample of an application can arrive at the system any time between time 0 and h, and every subsequent item arrives exactly h time units after the previous one. As is shown in the figure, initially the component is in the state initial, where it generates at most one item per time unit (modeled by the invariant $[0,1]$). If no item is generated after $h-1$ time units (modeled by the guard $t = h-1$ and $x = 0$), the component will move to the state arriving. At arriving, the component generates the first data item in the next time unit (modeled by the invariant $[1,1]$) and then moves to the state waiting (captured by the guard $x = 1$). In contrast, if the first item is generated before $h-1$ time units have passed, the component will move directly to the state waiting. In either case, the component will reset both x and t to zero upon entering waiting. It will then remain in waiting for exactly $h-1$ time units (during which no item is generated, as captured by the guard $[0,0]$) and will move to the state arriving. The component will then stay in arriving for exactly one time unit and generates exactly one data item before transitioning to waiting.

3) Semantics of the Application component

The semantics of the *Application* component is modeled by the finite automaton shown in Figure 4(b), whose guards are described in Table II. Initially, the component is in the idle state and the variable skip is set to zero, which indicates that the next incoming item will not be skipped. While the component is in this state, if the guard g_0 is true, i.e., a new item has arrived ($x = 1$) and this item will be skipped (skip $=$ 1), then the component will perform the reset R_0. Specifically, it will reset the value of skip to zero (so that the next data item will not be skipped) and update the status array to indicate that the most recent item is skipped ($\mathsf{status}_1 \leftarrow 1$) and that the status of the existing items remains unchanged ($\mathsf{status}_j \leftarrow \mathsf{status}_{j-1}$)[3]. If the guard g_1 is true, i.e., a new item has arrived

[3]Note that since there is a new item, the existing $(j-1)^{th}$ most recent item now becomes the j^{th} most recent item.



(a) ECA model of *Sensor* ($\tau_{th2} < h$). (b) Automaton model of *Application*. (c) Automaton model *PE* with FP scheduling.

Fig. 4: Automata models of the *Sensor*, *Application* and *PE* components.

Case	Guard	Reset	Explanation
Init		R_{init} : $\mathsf{skip} \leftarrow 0$; $\forall\, 1 \le i \le N$, $\mathsf{status}_i \leftarrow 0$;	Variable initialization
0	$g_0 : x = 1 \wedge \mathsf{skip} = 1$	$R_0 : \mathsf{skip} \leftarrow 0$; $\mathsf{status}_1 \leftarrow 1$; $\forall\, 2 \le i \le N$, $\mathsf{status}_i \leftarrow \mathsf{status}_{i-1}$;	New sample arrives, being skipped.
1	$g_1 : x = 1 \wedge \mathsf{skip} = 0$	$R_1 : i \leftarrow 1; e \leftarrow E_1$; $\mathsf{status}_1 \leftarrow 0$; $\forall\, 2 \le j \le N$, $\mathsf{status}_j \leftarrow \mathsf{status}_{j-1}$;	New sample arrives, not being skipped.
2	$g_2 : i \le n \wedge t < \tau_{th2}$	$R_2 : i \leftarrow$ **if** $e = s$ **then** $i+1$ **else** i; $e \leftarrow$ **if** $e = s_i$ **then** E_{i+1} **else** $e - s_i$;	Current sample is unfinished, with current delay $< \tau_{th2}$.
3	$g_3 : i \le n \wedge t = \tau_{th2}$	$R_3 : \mathsf{skip} \leftarrow 0$; $\mathsf{status}_1 \leftarrow 2$;	Current sample is unfinished, with current delay $= \tau_{th2}$, thus being aborted.
4	$g_4 : i > n$	$R_4 : \mathsf{skip} \leftarrow$ **if** $\tau_{th1} < t \le \tau_{th2}$ **then** 1 **else** 0;	Current sample is finished.

TABLE II: Transition guards and transitions for the automata in Figure 4(b).

($x = 1$) and this item will not be skipped ($\mathsf{skip} = 0$), then the component will move to the state busy while performing the reset R_1. In particular, the index of the task processing this data item is reset to 1 ($i \leftarrow 1$), indicating that the item will be processed by the first task; the remaining execution time of this task is set to its worst-case execution demand ($e \leftarrow E_1$); the array status is updated to indicate that the the new item is not skipped or aborted ($\mathsf{status}_1 \leftarrow 0$) and the status of the existing items remains unchanged.

Once entering the busy state, the component will remain in this state as long as the current data item is not fully processed by all n tasks and its current delay is less than τ_{th2}, i.e., the guard g_2 holds. In addition, at each time unit while g_2 holds, the component will update the index of the task that will be processing the current item (i) in the next time unit and its remaining execution time (e) based on whether the service available (s_i) is sufficient to complete the task that is currently processing the item (see reset R_2). In contrast, if the current delay of the task reaches τ_{th2} (guard g_3 holds) or the item has been fully processed (guard g_4 holds), the component will return to the idle state and wait for the next item to arrive. In the former case, the execution of the current item is aborted and thus, its status is changed to aborted ($\mathsf{status}_1 \leftarrow 2$) and the next item will not be skipped ($\mathsf{skip} \leftarrow 0$), which is reflected by the reset R_3. In the latter case, the current item is successfully processed and thus, the next item will be skipped if the delay of the current item is greater than τ_{th1} and less than or equal to τ_{th2}, which is reflected by the reset R_4.

4) Semantics of the PE component

Figure 4(c) shows the automaton that models the processing semantics of a PE that implements the fully-preemptive fixed-priority scheduling (FP) policy. As was discussed earlier, the PE executes m *Application* components; where, the current task of *Application* j is only executed by the PE when $\mathsf{pid}_j = \mathsf{pID}$. Therefore, the service provided to *Application* j (denoted by s_j) is zero if $\mathsf{pid}_j \neq \mathsf{pID}$. Otherwise, the service provided to *Application* j is the minimum of the execution demand e_j of the application and the remaining service of the ECU after having processed all higher-priority *Application* components k with $\mathsf{pid}_k = \mathsf{pID}$. This is reflected by the reset R_{fp} shown in the automaton.

B. Computing the skip and abort bounds

The maximum number of aborts (skips) in a sliding window of N can be established using verification technique. Recall that the processing status of the samples in the current window of an application is captured by the status array, where status_l is equal to 0, 1, or 2 if the l^{th} most recent data samples is processed successfully, skipped, or aborted, respectively, for all $l = 1 : N$. Therefore, the numbers of skipped and aborted items in the current window are given by $\mathsf{numSkips} = \sum_{1 \le l \le N} \{\mathsf{status}_l \mid \mathsf{status}_l = 1\}$ and $\mathsf{numAborts} = \sum_{1 \le l \le N} \{\mathsf{status}_l \mid \mathsf{status}_l = 2\}$, respectively.

As a result, given any constant value U, we can verify whether U is a valid upper bound on the number of skips in any window of N (consecutive) samples by verifying the Linear Temporal Logic (LTL) formula:

$$\Box(\mathsf{numSkips} \le U), \tag{34}$$

which states "*Always,* numSkips *is less than or equal to* U."

Thus, to determine the maximum number of skips in any window of length N, we perform a binary search on the value U, starting with the largest value $\lceil N/2 \rceil$[4]. The maximum number of skips over any window of length N, denoted by m_{sk0}, is then chosen as the smallest value of U for which (34) holds. The maximum number of aborts in a window of length N, denoted by m_{ab0}, can be obtained in a same manner, except that we initially start with N as the maximum value of U.

VII. Co-Design Algorithm

With the overrun framework and the corresponding control designs described in §IV to §V, and the platform analysis for overruns in §VI, we propose a co-design of control and platform in this section, using the two-parameter overrun strategy. The discussions with implementation platform in §VI

[4]Note that at most one sample is skipped for any two consecutive samples.

showed that the platform analysis starts with τ_{wc} and h to compute m_{sk0}, m_{ab0}, and N associated with the pair (τ_{th1}, τ_{th2}). The control design in §V requires $(\tau_{th1}, \tau_{th2}, m_{sk0}, m_{ab0}, N)$ and returns a decay rate $\alpha_{overall}$ (see Figure 5), with the overall control design becoming infeasible if $\alpha_{overall} < 1$. Together, the overall performance of the controller and the platform is then quantified by a $J_{overall} = \rho_1 J_c + \rho_2 J_p$, where J_c is the control performance cost, J_p is the platform cost, and $\rho_1, \rho_2 \in \mathbb{R}$ are constant parameters. J_c is determined by (31), which depends on the control parameters $\gamma_n, \gamma_a, \gamma_s$, and the skip and abort rates r^p_{skip}, r^p_{abort}. J_p is chosen so as to reflect the overall average resource utilization of the applications, and discussed below. The goal of the co-design algorithm is to find the optimal parameters τ_{th1} and τ_{th2} that minimize a cost $J_{overall}$.

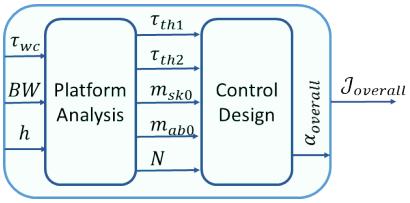


Fig. 5: A snapshot of the proposed co-design.

Suppose we choose $J_p = \sum_{1 \le j \le n} J^k_p$, where J^k_p is the overall resource utilization of the application k, which can be approximated by

$$J^k_p \approx \mathsf{wcet}/h \times (1 - r^p_{skip} - r^p_{abort}/2). \tag{35}$$

where wcet is worst-case execution demand. The insight of this approximation is that (i) whenever a sample is skipped, all the resource demanded by the sample is saved, and (ii) when a sample is aborted, the system may have executed a fraction of its demand, which can be as small as one execution time unit and as large as $(\mathsf{wcet} - 1)$ execution time units. J^k_p can therefore be viewed as a platform cost for application k using average values for delays between threshold values and the worst-case delay [5].

Our co-design algorithm proceeds in four steps, which correspond to 1) initialization, 2) determining thresholds for application C_i, 3) updating worst case delays for application $C_j, j \ge i$, and 4) repeating steps 2 and 3 for $i = 1 \cdots n$. If the above steps result in a feasible control design, then the fourth step of the codesign reduces the network bandwidth and returns to step 1. All parameters related to the i^{th} application are denoted with a superscript i. Details of these steps are as follows.

1. Initialization:
   - 1.a. Compute τ_{wc} for all applications $C_1, \cdots, C_n$ using techniques presented in §III-A, where C_1 and C_n are the highest and lowest priority applications, respectively.
   - 1.b. Set $\tau^i_{th1} = \tau^i_{th2} = \tau^i_{wc}$ for application C_i. In this case, there are no overruns and $r^{pi}_{skip} = r^{pi}_{abort} = 0$.
   - 1.c. Compute the control cost J^i_c in (31) using corollary 2, the platform cost J^i_p, and the overall cost $J^i_{overall}$ for a fixed set of parameters ρ^i_1 and ρ^i_2.
   - 1.d. Define a set S to include $(\tau^i_{wc}, \tau^i_{wc})$ as well as two perturbed delay points $(\tau^i_{wc} - \delta\tau^i, \tau^i_{wc})$ and $(\tau^i_{wc} - \delta\tau^i, \tau^i_{wc} - \delta\tau^i)$, with $\delta\tau^i \ll \tau^i_{wc}$.
2. Exploration of application C_i:
   - 2.a. For each element in S, find the maximum abort and skip rates r^{pi}_{skip} and r^{pi}_{abort} from platform analysis.
   - 2.b. compute the parameters $(\gamma_n, \gamma_s, \gamma_a)^i$ by solving LMI (23)-(25). The co-design is said to be feasible if $\alpha^i_{overall}$ in (30) is less than the desired exponential decay α^*.
   - 2.c. For each element in S, if the co-design is feasible,
     - 2.c.i. Compute J^i_c, J^i_p, and $J^i_{overall}$ for all elements in S, replace the previous $J^i_{overall}$ with the new value if $J^{i^{new}}_{overall} < J^{i^{old}}_{overall}$ and update $(\tau^i_{th1}, \tau^i_{th2})$ as in
     $$(\tau^i_{th1}, \tau^i_{th2}) = \arg\min(J^i_{overall}(\tau_1, \tau_2)).$$
     - 2.c.ii. Expand the search area with adding more elements to the set S by exploring $(\tau_1 - \delta\tau, \tau_2)$, $(\tau_1, \tau_2 - \delta\tau)$, and $(\tau_1 - \delta\tau, \tau_2 - \delta\tau)$ for each $(\tau_1, \tau_2) \in S$. In this step, the previously visited pairs and the pairs which violate the constraint $\tau_1 \le \tau_2$ are not considered.
   - 2.d. Repeat steps 2.*a*-2.*c* until there are no more points to explore.
3. Exploration of lower priority applications. Use (τ_{th1}, τ_{th2}) for i^{th} application and update τ_{wc} for lower priority applications $(i = i+1)$. Proceed from 1.*b*. Note that the updated τ_{wc} are smaller than previously computed values due to the overrun framework of the higher priority applications.
4. Reduce network Bandwidth. If feasible design achieved in step 3 for all applications, reduce the network bandwidth, return to step 1, otherwise return the previous bandwidth.

The end-result of this co-design returns optimal values $(\tau^*_{th1}, \tau^*_{th2})$, and optimal network bandwidth which optimizes $J_{overall}$ for each application.

Remark 2. *A trivial (sub-optimal) solution of the co-design problem presented here is a system with no overruns. This ensures the feasibility of the algorithm.*

VIII. Case Study

This section presents a case study of a network control system to demonstrate the utility of our co-design methods and the benefits of the overrun co-design method against the nominal co-design method.

A. Experimental setup

The system consists of $n = 6$ control applications, each of which corresponds to lane keeping of a vehicle, with an underlying computational architecture that consists of two ECUs connected via a CAN bus. Each application i consists of two control tasks, T^1_i on ECU1 and T^2_i on ECU2, and a message m_i. Each sensor value that arrives from the sensor cluster of the application i is first processed by T^1_i, and the processed sensor value is then sent to T^2_i via the message m_i. Based on the received value, T^2_i computes the control output to the corresponding actuator.

[5] We note that (35) gives one possible computation of J_p that we use to optimize the resource use; however, our co-design algorithm works for any other approximations of J_p, and it always produces a safe design regardless of the choice of J_p.

In our evaluation, the sampling periods h_i of the applications range between 5 ms and 35 ms. Both ECU1 and ECU2 employ the preemptive fixed-priority scheduling policy, whereas the CAN bus employs a non-preemptive fixed-priority scheduling policy, with application i having a higher priority than application $i+1$ for all $1 \leq i < n$. We assumed a fixed frame length for every CAN frame in the system.

Objectives. Our evaluation focuses on three aspects of the two co-design methods: (1) the minimum speed that the ECUs and the CAN network can operate to guarantee the control quality of every application; (2) the feasibility design regions of the platform; and (3) the impact of the delay thresholds on the resource savings. Towards this, we performed the following three sets of experiments:

In the first experiment, we considered different processor frequencies of the ECUs and for each frequency value, we determined the minimum network speed such that the control performance of every application i is satisfied for some delay thresholds τ^i_{th1} and τ^i_{th2} within its valid range (i.e., $0 < \tau^i_{th1} \leq \tau^i_{th2} \leq h_i$). At the same time, we computed as a baseline the minimum network speed for which a feasible design exists under the nominal co-design method, where the delay threshold τ_{th} of each application was set equal to its sampling period.

In the second experiment, we fixed the network bandwidth to be 400 Kbits/s and computed the minimum processor frequency of ECU2 required to find a feasible solution under different processor frequency values of ECU1. The computation is done in the same fashion as in the previous experiment.

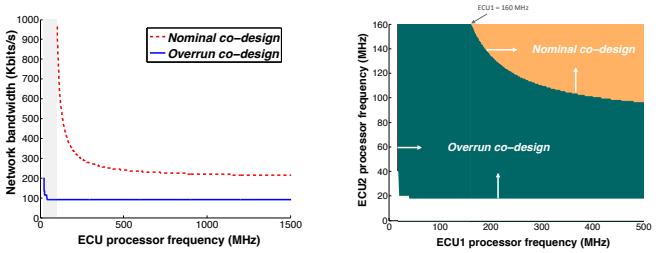
Finally, in the third experiment, we focused on the highest priority application only, with the network is fixed as before. We varied the delay thresholds within their valid ranges, and for each pair of threshold values we computed the resource required by the ECUs to guarantee the control quality of the application under the overrun co-design method with the chosen thresholds.

For the dynamical system, we consider the dynamic model of a vehicle for a lane-keeping application [29], the numerical values of which can be found in [11]. The controllers were designed using LQR of §III for various sampling times considered in the next section.

B. Evaluation results

Resource savings. Figure 6(a) shows the minimum network bandwidth required under the overrun and nominal co-design methods when varying the processor frequency of the ECUs. (Here, the frequency of ECU1 was always set equal to the frequency of ECU2.) We observe that the overrun co-design method consistently outperforms the nominal co-design method. Specifically, at the processor frequencies for which a feasible design exists for both methods, the overrun co-design method reduces the network resource bandwidth required by at least 2.4× and up to more than 10.7× compared to the nominal co-design method.

The results in Figure 6(a) also show that the smaller the processor frequency, the higher the resource savings. In fact, when the processor frequency is between 20 MHz and 100 MHz (the shaded area), no design solutions exist under the nominal co-design method even if the network bandwidth is arbitrarily large; in contrast, the overrun co-design method produces a feasible design using only a small network bandwidth, which is even smaller than the bandwidth that can be achieved by the nominal co-design method under an arbitrarily large processor frequency. For instance, under the overrun co-design method, a network bandwidth of 200 Kbits/s and 90 Kbits/s are sufficient to guarantee the control quality of all applications when each ECU operates at 20 MHz and 100 MHz, respectively. On the contrary, the nominal co-design method cannot find any feasible solution at the processor frequency within 20–100 MHz, and even when the processor frequency is arbitrary large, it requires a network bandwidth of at least 215 Kbits/s.



(a) Minimum network resource requirement.

(b) Feasibility design regions of the ECUs.

Fig. 6: Platform resource design space exploration under overrun and nominal co-design methods.

We also observe from Figure 6(a) that increasing the processor frequency beyond 100 MHz does not help reducing the network requirement. In other words, the portion of the overrun co-design curve at the processor frequency 20–100 MHz also forms the Pareto design curve for the overrun co-design method.

Feasibility design regions. Figure 6(b) illustrates the feasibility design regions for the two ECUs using the two methods when we fixed the network bandwidth to be 400 Kbits/s. The areas above the overrun and the nominal co-design curves in the figure correspond to the regions for which a feasible frequency exists for the ECUs under the overrun co-design and the nominal co-design, respectively. We observe that as the frequency of ECU1 increases, the feasible region is also widen for both methods, enabling smaller processor frequency for ECU2. These feasible regions can be used to optimize the platform resource under a given resource constraint.

It can also be observed from Figure 6(b) that the feasible region of the nominal co-design method falls strictly inside that of the overrun co-design method. Similar to the previous experiment, in contrast to the overrun co-design method, no solution exists for the nominal co-design method when the frequency of ECU1 falls below 160 MHz. Thus, the overrun co-design not only saves significant resources but also provides more flexibility for the platform design compared to the nominal co-design.

In summary, our evaluation demonstrates that not only does the overrun co-design method save the resource requirement by an order of magnitude but it also has a much larger feasible design space compared to the nominal co-design method.

IX. Conclusion

This paper addresses the problem of implementation of multiple control applications in a network control system where resources are limited and shared thereby resulting in varying delays in transmitted messages. Using a two-parameter model for these delays, a switching control strategy is proposed that varies between nominal, skip, and abort modes based on the magnitude of the delay. The underlying dynamic models in each of these cases are utilized in order to derive the stability of the switched system. An automata-theoretic approach is used for modeling the platform and for analyzing the maximum number of skip and abort samples under the proposed control strategy. Using both the platform and switched system analyses, a co-design algorithm is proposed that further optimizes the two-parameter delay thresholds to result in an efficient platform resource utilization as well as the desired control performance. A case study with six control applications implemented on a shared network with one bus and two ECUs is presented, which is shown to result in an order of magnitude reduction in the resource requirement and a much larger feasible design space

References

[1] J. Hespanha, P. Naghshtabrizi, and Y. Xu, "A survey of recent results in networked control systems," *Proceedings of the IEEE*, vol. 95, no. 1, pp. 138–162, 2007.

[2] M. B. G. Cloosterman, N. Van de Wouw, W. Heemels, and H. Nijmeijer, "Stability of networked control systems with large delays," in *CDC'07*. IEEE, 2007, pp. 5017–5022.

[3] N. Vatanski, J. Georges, C. Aubrun, E. Rondeau, and S. Jämsä-Jounela, "Networked control with delay measurement and estimation," *Control Engineering Practice*, vol. 17, no. 2, pp. 231–244, 2009.

[4] M. B. G. Cloosterman, N. van de Wouw, W. P. M. H. Heemels, and H. Nijmeijer, "Stability of networked control systems with uncertain time-varying delays," *IEEE Transactions on Automatic Control*, vol. 54, no. 7, 2009.

[5] W. Zhang, M. Branicky, and S. Phillips, "Stability of networked control systems," *IEEE Control Systems*, vol. 21, no. 1, pp. 84–99, 2001.

[6] A. Annaswamy, S. Chakraborty, D. Soudbakhsh, D. Goswami, and H. Voit, "The arbitrated networked control systems approach to designing cyber-physical systems," in *NecSys*, 2012.

[7] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973.

[8] M. Branicky, S. Phillips, and W. Zhang, "Scheduling and feedback codesign for networked control systems," in *Proceedings of the 41st IEEE Conference on Decision and Control*, vol. 2, dec. 2002, pp. 1211 – 1217 vol.2.

[9] P. Naghshtabrizi and J. Hespanha, "Analysis of distributed control systems with shared communication and computation resources," in *ACC*, 2009.

[10] H. Voit, R. Schneider, D. Goswami, A. Annaswamy, and S. Chakraborty, "Optimizing hierarchical schedules for improved control performance," in *SIES*, 2010.

[11] D. Soudbakhsh, L. Phan, O. Sokolsky, I. Lee, and A. Annaswamy, "Co-design of control and platform with dropped signals," in *ICCPS'13*, April 2013.

[12] M. Kauer, , D. Soudbakhsh, D. Goswami, S. Chakraborty, and A. Annaswamy, "Fault-tolerant control synthesis and verification of distributed embedded systems," in *Design, Automation & Test in Europe*, Dresden, Ge, 2014 (to appear).

[13] M. Yu, L. Wang, and T. Chu, "Stabilization of networked control systems with data packet dropout and network delays via switching system approach," in *IEEE International Symposium on Computer Aided Control Systems Design*. IEEE, September 2004, pp. 362–367.

[14] E. Garone, B. Sinopoli, and A. Casavola, "On the Effect of Packet Acknowledgment on the Stability and Performance of Networked Control Systems," in *Modelling, Estimation and Control of Networked Complex Systems*, ser. Understanding Complex Systems, A. Chiuso, L. Fortuna, M. Frasca, A. Rizzo, L. Schenato, and S. Zampieri, Eds. Springer Berlin Heidelberg, 2009, pp. 191–206.

[15] M. Lemmon and X. S. Hu, "Almost sure stability of networked control systems under exponentially bounded bursts of dropouts," in *HSCC*, 2011.

[16] E. Henriksson, H. Sandberg, and K. Johansson, "Predictive compensation for communication outages in networked control systems," in *47th IEEE Conference on Decision and Control*, 2008, pp. 2063–2068.

[17] Q. Ling and M. Lemmon, "Optimal dropout compensation in networked control systems," in *CDC'13*, vol. 1, Dec. 2003, pp. 670–675 Vol.1.

[18] G. Koren and D. Shasha, "Skip-over: algorithms and complexity for overloaded systems that allow skips," in *IEEE RTSS*, 1995, pp. 110–117.

[19] P. Ramanathan, "Graceful degradation in real-time control applications using (m, k)-firm guarantee," in *Twenty-Seventh Annual International Symposium on Fault-Tolerant Computing*, 1997, pp. 132–141.

[20] A. Cervin, "Analysis of overrun strategies in periodic control tasks," in *16th IFAC World Congress*, 2005.

[21] P. Ramanathan, "Overload management in real-time control applications using (m, k)-firm guarantee," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 6, pp. 549–559, 1999.

[22] G. Quan and X. Hu, "Enhanced fixed-priority scheduling with (m, k)-firm guarantee," in *RTSS*, 2000.

[23] E. Poggi, Y. Song, A. Koubaa, Z. Wang *et al.*, "Matrix-dbp for (m, k)-firm real-time guarantee," *RTSS*, 2003.

[24] J. W. S. Liu, *Real-Time Systems*. Prentice-Hall, Inc, 2001.

[25] S. Chakraborty, S. Künzli, and L. Thiele, "A general framework for analysing system properties in platform-based embedded system designs," in *DATE*, 2003.

[26] C.-W. Lin, M. D. Natale, H. Zeng, L. T. X. Phan, and A. Sangiovanni-Vincentelli, "Timing Analysis of Process Graphs with Finite Communication Buffers," in *IEEE RTAS'13*, 2013.

[27] M. S. Branicky, "Multiple Lyapunov functions and other analysis tools for switched and hybrid systems," *IEEE Transactions on Automatic Control*, vol. 43, no. 4, pp. 475 – 482, 1998.

[28] S. Chakraborty, L. T. X. Phan, and P. Thiagarajan, "Event count automata: A state-based model for stream processing systems," in *RTSS'05*, 2005.

[29] D. Soudbakhsh and A. Eskandarian, "Vehicle lateral and steering control," in *Handbook of Intelligent Vehicles*, A. Eskandarian, Ed. Springer London, 2012, pp. 209–232.

Appendix

Appendix A
Proof of Theorem 1

Proof. Inequalities (23) implies that the following inequalities are satisfied as well, with the Schur complement:

$$\Gamma_n^T P \Gamma_n \prec \gamma_n P, \tag{36}$$

similarly, inequalities (24) and (25) imply that

$$\Gamma_s^T P \Gamma_s \prec \gamma_{sk} P, \tag{37}$$

$$\Gamma_s^T P \Gamma_a \prec \gamma_{ab} P, \tag{38}$$

Also, we note from (21) that starting at time k_1, there are n_n nominal signals, m_{sk} skipped signals and m_{ab} aborted signals; with $1 \geq \gamma_n \geq 0$, we have:

$$\left(\Gamma_a^{s_p}\Gamma_s^{j_p}\Gamma_n^{i_p}\cdots\Gamma_a^{r_1}\Gamma_s^{j_1}\Gamma_n^{i_1}\right)^T P \left(\Gamma_a^{s_p}\Gamma_s^{j_p}\Gamma_n^{i_p}\cdots\Gamma_a^{r_1}\Gamma_s^{j_1}\Gamma_n^{i_1}\right) < \gamma_n^{n_n}\cdot\gamma_s^{m_{sk}}\cdot\gamma_a^{m_{ab}} P \leq \alpha^{-2} P \tag{39}$$

with $\alpha^{-2} \stackrel{\text{def}}{=} \gamma_n^{n_0}\gamma_s^{m_{sk0}}\gamma_a^{m_{ab0}}$.

These inequalities imply that a quadratic Lyapunov function in the form of $V = X[k]^T P X[k]$ exists for systems (23), (24), and (25), and it is decreasing with a decay rate of at least α for any interval $N = 2m_{sk0} + m_{ab0} + n_{n0}$, proving Theorem 1. □