A Hybrid Approach to Cyber-Physical Systems Verification

Pratyush Kumar, Dip Goswami[1], Samarjit Chakraborty[1],
Anuradha Annaswamy[2], Kai Lampka[3], Lothar Thiele

Computer Engineering and Networks Laboratory, ETH Zurich
[1]Institute of Real-Time Computer Systems, TU Munich
[2]Department of Mechanical Engineering, MIT
[3]Department of Information Technology, Uppsala University

ABSTRACT

We propose a performance verification technique for cyber-physical systems that consist of multiple control loops implemented on a distributed architecture. The architectures we consider are fairly generic and arise in domains such as automotive and industrial automation; they are multiple processors or electronic control units (ECUs) communicating over buses like FlexRay and CAN. Current practice involves analyzing the architecture to estimate *worst-case* end-to-end message delays and using these delays to design the control applications. This involves a significant amount of pessimism since the worst-case delays often occur very rarely. We show how to combine functional analysis techniques with model checking in order to derive a *delay-frequency interface* that quantifies the interleavings between messages with worst-case delays and those with smaller delays. In other words, we bound the frequency with which control messages might suffer the worst-case delay. We show that such a *delay-frequency interface* enables us to verify much tigher control performance properties compared to what would be possible with only worst-case delay bounds.

Categories and Subject Descriptors

C.3 [**Special-Purpose and Application-Based Systems**]: [Real-time and embedded systems]; D.2.4 [**Software/Program Verification**]: [Formal methods]; D.4.5 [**Reliability**]: [Fault tolerance]

General Terms

Design, Theory, Verification

Keywords

Cyber-Physical Systems, Frequency-Delay Metric, Stability, Real-Time Calculus, Timed-Automata

1. INTRODUCTION

Cyber-physical systems involve a tight interaction between the cyber (computational) and the physical components of an embedded system. In this paper we study cyber-physical systems in which multiple control applications are implemented in a distributed fashion on a set of electronic control units (ECUs) that communicate over buses like CAN and FlexRay (see Figure 1). Such a setup is fairly generic and can be found in automotive systems, industrial automation systems, large-scale robotic systems and in avionics.

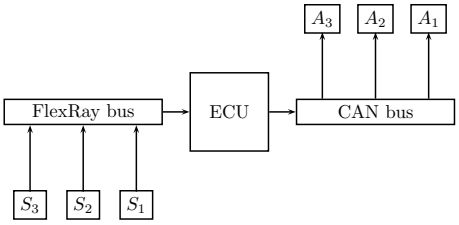

Figure 1: Architecture considered for example

Traditionally, the control theory community has largely ignored the implementation aspects (both software and architecture) of controllers and has focused on mathematical models, their analysis, and high-level simulation. In this process, a number of simplifying assumptions have been made when designing controllers. They include neglecting the computation times when evaluating control laws and neglecting the control message communication times. The implementation of the high-level control laws while taking into account implementation issues that have been ignored while designing these laws, mostly fell within the domain of embedded systems and software design. Several research threads within the embedded systems community have tried to systematically address these issues. They include the development of the synchronous language paradigm [4], time-triggered languages like Giotto [7] and other related formalisms such as PTIDES [16]. In parallel to these efforts, as the implementation platforms became more complex and the *semantic* gap between high-level control models and their implementations widens, the control theory community started investigating what is referred to as *networked control systems* (NCS) [13, 14], where the focus has been on distributed controllers communicating over a wireless network. Research with NCS takes into account issues such as delays suffered by control messages, message loss and jitter, and factors these issues into controller design.

In the case of distributed controller design, the current practice is to estimate worst case end-to-end message delays and design control laws that account for these delays. Within the NCS domain, statistical properties of these delays have also been taken into account while doing controller design. The problem in such cases boil down to analyzing

DAC 2012, June 3-7, 2012, San Francisco, California, USA.

the stability and performance of systems consisting of several switching subsystems, each designed for specific message delay values. Tools used within the control systems community to analyze such switching behavior include Multiple Lyapunov Functions [6], the Average Dwell time [15] approach, or common quadratic Lyapunov functions [9]. In all of these cases, the switching between multiple subsystems is assumed to be arbitrary and the behavior of the system is characterized by the *worst-case* switching pattern.

Our contribution: In this paper we observe that often the worst-case message delays occur very rarely. Further, in contrast to wireless networks, the architectures we study (multiple ECUs communicating over FlexRay or CAN networks and scheduled using deterministic scheduling policies) offer substantial *structure* to rule out certain delay (and hence switching) patterns. In other words, we can bound the occurrences of messages with worst-case delays, or quantify the interleavings between messages with worst-case delays and those with smaller delays. In this paper we refer to this as the *delay-frequency interface* between the underlying architecture and the control applications implemented on this architecture. We show that such *delay-frequency interfaces* may be used to verify much tighter control performance properties in comparison to using the worst-case delay alone (i.e., without the frequency with which the delay occurs).

Exploiting the above observation involves two technical challenges. (i) How to compute the *delay-frequency interface* for a distributed architecture? and (ii) How to use such an interface to give tighter control performance guarantees? Towards (i), we use a hybrid combination of (a) a functional approach to analyzing the timing properties of an architecture with (b) model checking. The functional approach is based on *real-time calculus* [12], where functions over time interval lengths are used to provide upper and lower bounds on the *service* provided by a resource (such as an ECU or a bus) or bounds on the arrival patterns of messages to be processed or transmitted by a resource. These bounds are then analyzed to estimate worst-case delays or backlogs suffered by message streams. For such analysis we have used the Matlab-based `Modular Performance Analysis` toolbox (available from `www.mpa.ethz.ch`). While the worst-case delays computed by this method consider *all* possible service and message arrival patterns that are specified by the upper and lower bounds/functions, we rule out certain architecture-specific service and arrival patterns through model checking (using UPPAAL [3]) and obtain our required *delay-frequency interface*, which is a significantly richer interface than the worst-case delay alone. Towards (ii), we utilize an analysis technique based on the existence of a common quadratic Lyapunov function [9]. As already mentioned, the main design task from control side is to guarantee control stability and performance in the presence of switching. Knowledge of the delay frequency metric essentially restricts the set of possible switching patterns. We address the stability and performance related aspects of the control loops for this restricted set of switching behavior (coming from the knowledge of delay frequency metric) by showing the existence of common Lyapunov functions among various switching subsystems as a proof of stability.

Related Works: Lately, there has been a considerable amount of activity in the area of *control/architecture* co-design. In particular, Alur *et al.* quantified the semantic gap between the control models and their implementations in [2]. In [11], an integrated co-design of control and architecture has been proposed. However, none of these and other efforts utilized the delay properties of the underlying architecture to bound control performance. Similarly, while stochastic delay distributions have been used in the NCS domain, to the best of our knowledge, this work is the first effort to use a deterministic distribution on control message delay values in order to give tighter control performance guarantees.

The rest of the paper is structured as follows. In Section 2, we discuss a motivating example that highlight the limitation of the existing approaches. In Section 3, we describe the delay-based feedback control strategy and we then introduce the idea of delay frequency metric in light of the proposed control strategy. Section 4 illustrates how a delay frequency metric can be verified for a given architecture. The delay frequency metric is richer timing interface that we argue for. In Section 5, we demonstrate how we can apply the knowledge of the proposed timing interface to tighten the control design considering the example shown in Section 2. Several of the computations and explanations have been included in Appendices A-F.

2. MOTIVATION

In this section, we describe a motivational example of a control application that is implemented over a distributed architecture, which is retained throughout this paper. The fundamental objective of any control application is to regulate the behavior of dynamical systems (or plants) which are more commonly referred to as plant dynamics. The behavior of such dynamical systems (feedback signals) is measured using sensors, and the control algorithm (running on a processing unit) decides the necessary actuation signal which is then realized by the actuator. Hence, the feedback loop is closed over a set of spatially separated sensors and actuators communicating via a bus system and processors in distributed cyber-physical systems. In this paper, we consider an example architecture from the automotive domain. However, the analysis presented is relevant to a variety of other settings too.

The considered example is representative of three characteristic properties of today's cyber-physical systems, such as those found in modern cars or in industrial automation applications. First, it depicts how the integrated nature of the system brings together several applications, which have to be analyzed simultaneously, even though the analysis challenges may be modular. Second, it highlights the distributed nature of the system, whereby the control feedback loop goes through several components: sensors, buses, ECUs and finally actuators. Third, it characterizes the heterogeneity of the system architecture, wherein several components such as specialized buses are used.

2.1 System architecture

As a representative system, we consider the example shown in Figure 1. It consists of three sensors S_1, S_2 and S_3 that are connected to three Electronic Control Units (ECUs) (not explicitly shown in the figure). The sensor readings from these ECUs are transmitted via a FlexRay bus to the ECU shown in Figure 1. Subsequently, the sensor readings are computed upon in the ECU and the generated output is transmitted over a Controller Area Network (CAN) bus to actuators A_1, A_2 and A_3 (which are connected to three other ECUs). Each sensor/actuator (S_i, A_i) pair forms a control loop which regulates the behavior of an independent plant P_i. Thus, three sensor/actuator pairs regulate three independent plants denoted as P_1, P_2 and P_3. In this paper, we study various aspects of stability and performance of the plant P_2.

The timing properties of the above system are character-

Plant	τ^{FR}	τ^{ECU}	τ^{CAN}
1	40	50	30
2	20	15	10
3	20	15	20

Table 1: Worst-case execution times of different tasks (in ms)

ized by three aspects: (a) how often the sensors are read, (b) what is the time required to process the feedback in the different components, and (c) what are the scheduling parameters in the different components. We now describe these three aspects.

We categorize the tasks in the above architecture into three classes. First, the task τ_i^{FR} responsible for sending the sensor reading from S_i via the FlexRay bus. Second, the task τ_i^{ECU} responsible for processing the sensor signal sent by τ_i^{FR}, computing the control signal and sending the control signal via the CAN bus. Third, the task τ_i^{CAN} is responsible to actuate A_i based on the control signal sent by τ_i^{ECU}. The worst-case execution times of these tasks on their respective components are shown in Table 1.

Essentially, it is assumed that the sensor S_i and the actuator A_i are attached to plant P_i. Based on the reading from S_i and the actuation from A_i, the plant P_i is regulated. We assume that P_2 and P_3 are time-triggered with period 100ms. That is, τ_i^{FR}, τ_i^{ECU} and τ_i^{CAN} are triggered periodically with periods 100ms for $i = \{2, 3\}$. For P_1, τ_1^{FR}, τ_1^{ECU} and τ_1^{CAN} are triggered with a period of 100ms and maximum jitter of 50ms.

The FlexRay bus is used by τ_i^{FR} to send the sensor reading to the ECU. The FlexRay bus is divided into communication cycles which consists of the time-triggered static and the event-triggered dynamic segments. The static segment is divided into multiple *slots* of equal length. On the other hand, the bus access in the dynamic segment is arbitrated in a fixed-priority fashion. In this example, we consider the FlexRay bus with static and dynamic segments of length 40 ms each, and a period or cycle length of 80ms. The task τ_1^{FR} transmits message over the static segment. The messages from τ_2^{FR} and τ_3^{FR} are transmitted over the dynamic segment with increasing order of priority. On the ECU, a hierarchical scheduler is used. τ_1^{ECU} is scheduled as a high-priority task. The other two tasks are scheduled within a low-priority task which executes two constant bandwidth servers [1] with utilization 50% each, to serve τ_2^{ECU} and τ_3^{ECU}. Finally, the CAN bus schedules the messages with priorities in decreasing order, with messages from τ_1^{CAN} and τ_3^{CAN} having highest and lowest priorities respectively.

2.2 Control applications

In the context of the architecture described in the previous section, we focus on the feedback loop for plant P_2. As a plant dynamics P_2, we consider a common discrete-time linear time-invariant (LTI) system of the form,

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

$$A = \begin{bmatrix} 0.0 & 1 \\ 0.9 & 0.2 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \quad (2)$$

where $x[k]$ is the vector of *state variables* and $u[k]$ is the *control input* to the system which needs to be designed. We consider a *regulation* problem which essentially means the design of $u[k]$ such that $x[k] \to 0$ from any initial condition with k. In this work, we consider static state-feedback control strategy for designing $u[k]$. In view of the above system, we make the following observations:

- The sensor S_2 reads all the states $x[k]$, i.e., the states are *measurable*. The measurement is done periodically with constant period of $p = 100$ms. Similarly, the actuation is periodical with $p = 100$ms using A_2.
- The time when S_2 reads feedback signal $x[k]$ to the point when A_2 is actuated utilizing that signal is called the *sensor-to-actuator* delay d. The delay d consists of the transmission/waiting time of the sensor signal, processing time of the sensor signal and the transmission/waiting time of the actuator signal. Clearly, d is not constant across jobs.
- With $u[k] = 0$, the resulting plant is open-loop and the behavior of the open-loop plant is dictated by properties of the system matrix A. In this case, the absolute value of the maximum eigenvalue of A is outside the unit circle, i.e., $|\lambda_{max}(A)| > 1$. That is, the open-loop plant is unstable.
- In the presence of a *sensor-to-actuator* delay d, the task is to design static state-feedback $u[k] = Kx[k - \left\lceil \frac{d}{p} \right\rceil])$ such that $x[k] \to 0$ with $k \to \infty$ where K is the state-feedback gain.
- With above control input, we denote the closed-loop system matrix by A_{cl}. We measure the quality of control using the stability margin, i.e., $Q = (1 - |\lambda_{max}(A_{cl})|)$. A larger Q indicates a higher stability margin.

As a design criterion, we are required to guarantee a stability margin $Q \geq 0.15$ in our example.

2.3 Design Motivation

Towards designing a regulator for the control loop under consideration, the easiest way is to design the system using the worst-case sensor-to-actuator (or end-to-end) delay. There are various well-known approaches in the real-time systems literature for computing such end-to-end delays. As mentioned before, we have chosen the *Modular Performance Analysis (MPA)* toolbox because it provides several advantages like compositionality. We modelled the timing properties and scheduling parameters within this tool and obtained the worst-case sensor-to-actuator delay $d_{max} = 320$ms. The detailed analysis is presented in Appendix A.

The above analysis implies that with periodic actuation of A_2 with $p = 100$ms, the worst-case sensor-to-actuator delay is $\left\lceil \frac{d_{max}}{p} \right\rceil = 4$ samples. Then, the obvious design possibility is to set $u[k] = Kx[k-4]$, where every feedback signal delayed by 4 samples. There are several standard approaches in the control theory literature for designing K with such delayed feedback signals. With static state-feedback control scheme [10], the best stability margin of $Q = 0.1455$ is obtained with $K = [-0.14 \ -0.05]$.

The computed stability margin does not meet the design criteria ($Q \geq 0.15$), and hence a design revision is deemed necessary. Such a design revision may need an upgrade of the network or the processing architecture or the removal or redesign of some of the other control applications in the system. However, it may be possible that the existing system itself can provide the required higher level of performance, which is not exposed by our analysis technique. In other words, the traditional approach to utilize worst-case end-to-end delay as an *interface* can lead to an *analysis gap*. Indeed, the existence of such a gap can be easily argued for. Not every feedback sample will suffer the worst-case sensor-to-actuator delay d_{max}. A large number of feedback messages

would suffer a delay no more than some smaller threshold delay, say d_{th}. This smaller delay can enable the guarantee for a better stability margin. This is the motivation for our work, i.e., deriving and demonstrating a richer interface between of the cyber and physical aspects of the system to enable tighter analysis.

To this end, we need to have (a) a controller design that exploits a guaranteed interleaving between messages with $d > d_{th}$ and those with $d < d_{th}$, (b) a metric that formally captures the delay variation, and (c) a computation technique to verify that this metric is indeed satisfied for the considered architecture. In the remainder of this paper, we will discuss these issues and indicate how they can be used to tighten the results for the stability margin of P_2.

3. DELAY-BASED FEEDBACK CONTROL

In this section, we will propose a controller feedback technique that will enable us to exploit the time-varying sensor-to-actuator delays to compute tighter stability margins. In addition, this will help us to define a richer interface between the cyber and the physical aspects of the system.

As discussed, not all feedback messages will suffer the largest delay d_{max}. There can exist a threshold delay, say d_{th}, such that *most* of the feedback messages suffer a delay $d \leq d_{th}$. With this observation, we propose the following control strategy: (a) we design a stabilizing controller with $u[k] = Kx[k - \left\lceil \frac{d_{th}}{p} \right\rceil]$, (b) actuator A_2 applies the control input $u[k] = Kx[k - \left\lceil \frac{d_{th}}{p} \right\rceil]$ only when delay is d_{th} or less and we denote the resulting closed-loop system by A_{cl}, and (c) whenever sensor-to-actuator delay $d > d_{th}$, we do not use that feedback signal and let the system run in open-loop, i.e., $u[k] = 0$ and the resulting system matrix is A. The control scheme is thus given as

$$\begin{aligned} u[k] &= Kx[k - \left\lceil \frac{d_{th}}{p} \right\rceil], && \forall\ d \leq d_{th} \\ &= 0, && \forall\ d > d_{th} \end{aligned} \tag{3}$$

The above control scheme leads to a switched system with two subsystems: (b) when feedback message suffers $d \leq d_{th}$ and the corresponding system is represented by A_{cl}, (a) when feedback message suffers a delay $d_{th} < d \leq d_{max}$ and the corresponding system is represented by A. For brevity of notation, we refer feedback messages of the first kind as "valid" samples while the messages of the second kind are called "invalid" samples.

There are standard techniques in the control theory literature to analyze the stability of the switched systems. However, most of these approaches assume worst-case or arbitrary switching behaviors. In our case, using our improved real-time systems analysis, we can rule out certain switching patterns. In other words, we can identify a restricted subset of switching possibilities, which can be finite for many practical design purposes. The switched system with such known switching possibilities are then analyzed using standard tools such as Average Dwell Time (ADT) [15] or Common Quadratic Lyapunov Function (CQLF) [9]. For this purpose, we propose the following metric as a richer class of interface.

DEFINITION 1 (DELAY FREQUENCY METRIC (d_{th}, n)). *If every feedback message with delay larger than d_{th} is followed by at least n feedback messages with delay no more than d_{th}, the* delay frequency metric *is said to be (d_{th}, n).*

Using the notion of valid and invalid samples, the above definition says that, every invalid sample is followed by at least n valid samples.

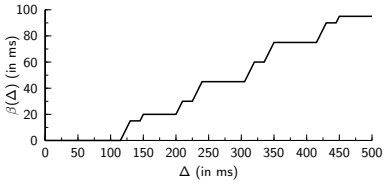

Figure 2: Service curve provided by ECU to $\tau_2^{\mathbf{ECU}}$

In the above context, the paper has two contributions – (a) From the real-time systems perspective, we *quantify* the interleaving between the number of messages with delay $d \leq d_{th}$ and $d_{th} < d \leq d_{max}$. (b) From the control system perspective, we utilize this more expressive *delay frequency metric* to obtain tighter bounds on the control system design. As already mentioned before, in the control systems literature such rich interfaces between the architecture and the controller has not been explored before.

In Section 4, we illustrated how a certain delay frequency metric can be verified for a given architecture. Next, we utilize the knowledge of the delay frequency metric in analyzing control stability and performance for a given architecture. We demonstrated the control theoretic analysis in Section 5 with the control loop P_2 in Figure 1.

4. VERIFYING DELAY-FREQUENCY

sIn the earlier section, we used the MPA toolbox to compute the worst-case sensor-to-actuator delay suffered by the feedback messages of P_2. However, this interface does not tightly interface the cyber and physical aspects of the system. We proposed the *delay frequency metric* along with a delay-based feedback system to allow for a tighter coupling. This section is devoted to computational techniques that can verify a given delay frequency metric for a given architectural setup. To this end, we use a hybrid approach combining analytical techniques of Real-Time Calculus [12] and state-based methods of the model checker UPPAAL [3].

4.1 Service curves

Service curves are used in Network Calculus [5] and Real-Time Calculus [12] to specify bounds on the available resource to a stream of tasks, in the interval domain. More concretely, $\beta(\Delta)$ denotes the *minimum* amount of execution time available to a stream of tasks within *any* busy window of length Δ. Techniques are known to compute the service curve for a wide class of resource arbitration policies. Examples include TDMA, fixed priority, EDF, round robin and servers. Within the scope of this paper, we consider the service curve as a given quantity and do not discuss its computation. As an example, the service curve available to τ_2^{ECU} on the ECU can be shown to be as in Figure 2.

When dealing exclusively with end-to-end timing properties, the service curves of multiple components in series can be *convolved* into a single service curve. For instance, in our example, the feedback message must go through the FlexRay bus, the ECU and the CAN bus. Each of these components is characterized by its own service curve. We can represent the timing properties of these components with a single service curve given as

$$\beta^{\text{end-to-end}} = \beta^{\text{FR}} \otimes \beta^{\text{ECU}} \otimes \beta^{\text{CAN}}, \tag{4}$$

where $\otimes$ is the convolution operation as defined in Real-Time Calculus [12]. For the scope of this work, we may abstract

the timing properties of the system, with respect to feedback messages of P_2, with a single service curve $\beta^{\text{end-to-end}}$.

4.2 Modelling a resource with UPPAAL

As discussed above, a service curve bounds the timing behavior exhibited by a resource. In other words, it characterizes a *set*, say $\mathbf{S}$, of possible timed behaviors of the resource. If this set can be translated into the set of possible behaviors of a timed automata, we can model the service curve as a component in a model checking tool such as UPPAAL. Such modelling can enable us to verify richer timing properties on the system such as the delay frequency metric.

A generic service curve represents a very large set of constraints: for every interval of a busy interval, a constraint must be satisfied. Translating such constraints into an UPPAAL component can lead to a complicated automaton, with intractable verification time. We therefore propose a conservative approximation of the service curve, whereby it is represented using a series of *periodic* service curves.

Consider an approximate service curve $\tilde{\beta}$ satisfying

$$\tilde{\beta}(\Delta) \leq \beta(\Delta), \forall\ \Delta \geq 0. \tag{5}$$

Let the set of possible timing properties exhibited by a resource with a service curve $\tilde{\beta}$ be denoted as $\tilde{\mathbf{S}}$. Then it follows that

$$\mathbf{S} \subseteq \tilde{\mathbf{S}} \tag{6}$$

The above relation implies that, if we verify some property to be always true with the service curve $\tilde{\beta}$, then the property is also always true for the service curve β. In this sense, the approximation $\tilde{\beta}$ is said to be conservative. Now consider a specific approximation $\tilde{\beta}$ given as

$$\tilde{\beta} = \max(0, \beta_1(o_1, P_1, Q_1), \beta_2(o_2, P_2, Q_2), \ldots) \tag{7}$$

where $\beta_i(o_i, P_i, Q_i)$ is a periodic service curve defined as

$$\beta_i(\Delta) = \left(\left\lfloor \frac{\Delta - o_i}{P_i} \right\rfloor + 1 \right) \times Q_i \tag{8}$$

As an illustration, the service curve available to τ_2^{ECU} (Figure 2) is approximated using periodic curves in Figure 4) in Appendix B. Periodic service curves can be easily modelled in separate UPPAAL components. These components can then be interfaced together to model a resource with service curve $\tilde{\beta}$ following (7). Such an interfaced component is shown in Appendix C.

4.3 Other components

Along, with the automaton that represents the behavior of the resource, we need an automaton to represent the generation of a stream of tasks (e.g., corresponding to the control task that needs to be executed for each sampled sensor input). Computing automatons to represent generation of a wide variety of streams has been discussed in [8].

Additionally, we need an observer automaton to encode the verification of the delay frequency metric. Let the delay frequency metric that we want to verify be given as (d_{avg}, n). Recall that this means that every invalid sample is followed by at least n valid samples. This can be easily verified using a UPPAAL observer component that (a) tracks the delay suffered by the feedback messages, (b) classifies messages as valid and invalid samples, and (c) verifies the delay frequency metric.

To conclude this section, recall that we were interested in verifying timing properties which are richer than the worst-case delay. To this end, we used the analytical formulation of service curves to represent the architecture in a modular fashion, used conservative approximation techniques to enable its efficient representation using the model checking tool, and then used an automata for the generation of the stream of tasks and an observer to verify properties such as the delay frequency metric. The observer automaton discussed here is illustrated in Appendix C.

5. THE CASE STUDY: REVISITED

In this section, we will revisit the example discussed in Section 2 (see Figure 1). Equipped with techniques from the previous two sections, we aim to demonstrate that we can tighten the computation of the stability margin for the considered architecture and the control loop P_2. In the following, we illustrate: (a) computation of delay frequency metric for the given architecture, (b) the control performance bound that we obtain without the knowledge of the delay frequency metric, and (c) the control performance bound that we obtain utilizing the knowledge of the delay frequency metric.

(a) Computing the delay frequency metric: With the worst-case delay of $d_{max} = 320$ms , $\left\lceil \frac{d_{max}}{p} \right\rceil = 4$. That is, working with the worst-case delay means setting $d_{th} = 400$ms. As demonstrated in Section 2, the control loop fails to meet design criterion ($Q \geq 0.15$). To improve upon this design, we set d_{th} to 300ms. For this value of d_{th}, we verify the delay frequency metrics for different values of n, using the UPPAAL model checker as described in the previous section. The largest value of n, in our example, for which the delay frequency metric is guaranteed by the considered architecture is 2. In other words, the architecture guarantees a delay frequency metric of (300 ms, 2). Note that this is a richer timing interface than specifying only the worst-case delay.

Coming back to the example of control loop P_2, $\left\lceil \frac{d_{th}}{p} \right\rceil = 3$ ($d_{th} = 300$ms, $p = 100$ms). Hence, we choose $u[k] = Kx[k-3]$ and $K = [0\ \ -0.28]$. We apply the control scheme described in Section 3. When $d \leq 3p$, we apply $u[k] = Kx[k-3]$ with $K = [0\ -0.28]$. The resulting closed-loop system A_{cl} is shown in Appendix D. With $d > 3p$, we run the system in open-loop and the resulting system matrix is A. The stability of this switched system depends on the interleaving between valid and invalid samples. The tighter design of the architecture needs to allow more frequent invalid samples. On the other hand, the occurrence of invalid samples causes degradation in stability margin and can potentially destabilize the system. The frequency of permissible invalid samples can be computed utilizing the traditional analysis tools such as Average Dwell Time (ADT) and Common Quadratic Lyapunov Function (CQLF).

(b) Design without the knowledge of delay frequency metric: In this case, we rely on ADT-based analysis. For ADT-based analysis, we consider switching between two subsystems A_{cl} when the delay is no more than $d_{th} = 300$ms and A when the delay exceeds 300ms. We illustrated the analysis in Appendix E. It clearly shows that only 1 invalid sample is allowed in any window of 25 samples for stability (i.e., for ensuring $Q > 0$). Hence, this does not meet the delay frequency metric computed earlier. Thus, this analysis which does not utilize the knowledge of delay frequency metric fails to guarantee desired stability margin. It should be noted that the CQLF-based analysis is not applicable to the systems with unstable subsystems. Hence, CQLF-based analysis is not suitable in this case as the open-loop system

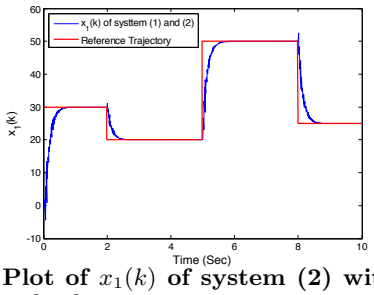

Figure 3: Plot of $x_1(k)$ of system (2) with our proposed control scheme.

A is unstable.

(c) Design exploiting the knowledge of delay frequency metric: For the delay frequency metric with $n = 2$, every invalid sample is followed by at least two valid samples when P_2 is implemented over the architecture under consideration. Hence, the overall system can be represented as follows,

$$x[k + 2 + n_i + n_j] = AA_{cl}^{n_i} \times AA_{cl}^{n_j} x[k], \quad (9)$$

where $n_i \neq n_j$ and $n_{i,j} \geq 2$. Hence, the resulting system has switching subsystems of the form $AA_{cl}^{n_i}$ with $n_i \geq 2$. For such a switched system, the stability could not be shown utilizing ADT-based analysis for the subsystems given by $AA_{cl}^{n_i}$ with $n_i \geq 2$. Now, we reply on Common Quadratic Lyapunov Function (CQLF) approach [9] for the stability analysis. The analysis presented in Appendix F shows that the above switched system is stable. The stability margin that we obtain for the given delay frequency metric ranges from $Q = 0.1812$ (when every two valid samples are followed by one invalid sample) to $Q = 0.2462$ (when no invalid sample occurs). Therefore, we could guarantee to provide a desired stability margin of $Q \geq 0.15$.

Fig. 3 shows the plot of state $x_1(k)$ of the control application (2) of control loop P_2 with the proposed control strategy. In this experiment, we have randomly dropped samples such that one invalid sample is followed by minimum two valid sample, i.e., according to the results coming from the above analysis. We have considered a tracking problem, i.e., ensuring that the state $x_1[k]$ follows the trajectory indicated by the red line. Towards this, we used an additional feedforward component (for changing the reference from *zero* to some other value) in the control input keeping the feedback component as described above. Fig. 3 shows that tracking can be done with the derived control scheme.

Hence, the system does indeed meet the design criterion on the stability margin of P_2. We have thus, highlighted how we can combine the delay-based feedback technique with the delay frequency metric to provide a tighter analysis of the cyber and physical aspects of the system.

6. CONCLUDING REMARKS

In this paper we quantify the interleavings between control messages experiencing different delay values. Further, we use this information to provide tight bounds on control performance. This a significant improvement over the case where controllers are designed only on the basis on worst-case delay values. Our proposed approach involves a computationally expensive model checking procedure and an exhaustive search over a parameter space in order to compute the delay frequency interface. As a part of future work, we will study the scalability of this procedure and make this process more efficient.

Acknowledgement This work is supported by European Community FP7 grant 248776.

7. REFERENCES

[1] L. Abeni and G. C. Buttazzo. Integrating multimedia applications in hard real-time systems. In *IEEE Real-Time Systems Symposium*, pages 4–13, 1998.

[2] R. Alur and G. Weissr. Regular specifications of resource requirements for embedded control software. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, 2008.

[3] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. UPPAAL - A tool suite for automatic verification of real-time systems. In *Proceedings of the DIMACS/SYCON Workshop on Hybrid Systems III: Verification and Control*, 1995.

[4] G. Berry and G. Gonthier. The esterel synchronous programming language: Design, semantics, implementation. *Sci. Comput. Program.*, 19(2):87–152, 1992.

[5] J.-Y. L. Boudec and P. Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Lecture Notes in Computer Science, Vol 2050. Springer, 2001.

[6] M. S. Branicky. Multiple Lyapunov functions and other analysis tools for switched and hybrid systems. *IEEE Transactions on Automatic Control*, 43(4):475 – 482, 1998.

[7] T. A. Henzinger, B. Horowitz, and C. M. Kirsch. Giotto: a time-triggered language for embedded programming. *Proceedings of the IEEE*, 91(1):84–99, 2003.

[8] K. Lampka, S. Perathoner, and L. Thiele. Analytic real-time analysis and timed automata: a hybrid method for analyzing embedded real-time systems. In *IEEE International conference on Embedded software (EMSOFT)*, 2009.

[9] O. Mason and R. Shorten. On common quadratic Lyapunov functions for stable discrete-time LTI systems. *IMA Journal of Applied Mathematics*, 69(3):271–283, 2002.

[10] W. Rugh. *Linear Systems Theory*. Prentice-Hall, N.J., 1996.

[11] S. Samii, A. Cervin, P. Eles, and Z. Peng. Integrated scheduling and synthesis of control applications on distributed embedded systems. In *Design Automation and Test in Europe (DATE)*, 2009.

[12] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *IEEE International Symposium on Circuits and Systems*, 2000.

[13] G. C. Walsh, H. Ye, and L. G. Bushnell. Stability analysis of networked control systemss. *IEEE Trans. on Control System Technology*, 10(3):438–446, 2002.

[14] M. Yongguang, C. Wenying, and L. Guangxiao. Compensation of networked control systems with time-delay and data packet losses. In *Chinese Control and Decision Conference*, 2009.

[15] G. Zhai, B. Ho, K. Yasuda, and A. N. Michel. Qualitative analysis of discrete-time switched systems. In *American Control Conference (ACC)*, 2002.

[16] J. Zou, S. Matic, E. A. Lee, T. H. Feng, and P. Derler. Execution strategies for PTIDES, a programming model for distributed embedded systems. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2009.

APPENDIX

A. WORST-CASE DELAY

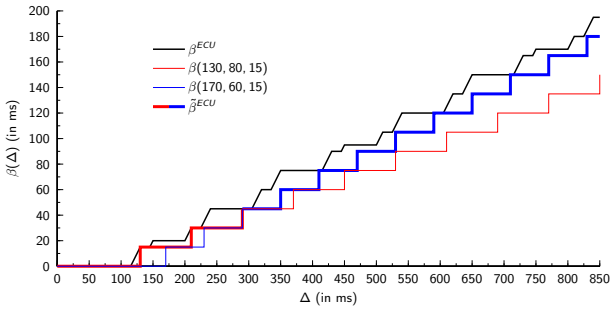

Figure 4: Approximation of a service curve using periodic service curves

Here, we will describe computation of the worst-case sensor-to-actuator delay for the feedback messages of plant P_2 on the architecture shown in Figure 1. We use the MPA toolbox to model the architectural units and the other tasks. Using this we compute the service curves provided by respective units to tasks τ_2^{FR}, τ_2^{ECU} and τ_2^{CAN}. These are shown in Figure 5. Then we use the toolbox to compute the convolution of the service curves. The worst-case end-to-end delay is then given by the horizontal distance between the convolved service curve and the periodic curve of the sensor input. This is also shown in Figure 5. The worst-case delay is obtained to be 320ms.

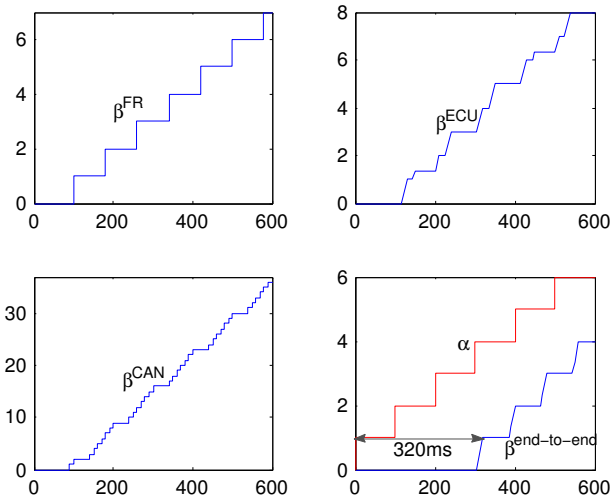

Figure 5: Computation of the worst-case end-to-end delay. Note that the y-axis shows service curve in units of messages, while the x-axis shows time in ms

B. APPROXIMATION OF SERVICE CURVES

We will describe how we can use the approximation step of (7) to conservatively and compactly represent a generic service curve. Consider the service curve of the ECU as provided to task τ_2^{ECU}, as shown in Figure 2. Clearly, the service curve is not a periodic service curve and representing it in a UPPAAL component can be very complicated. We can however, conservatively, represent the service curve by

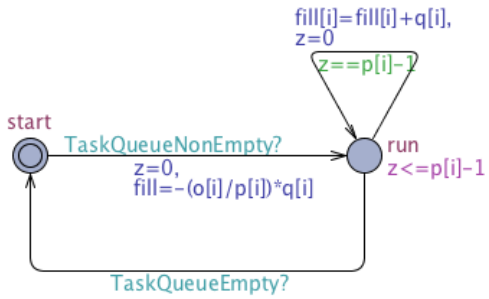

Figure 6: UPPAAL component for the periodic service curve

the following approximate service curve

$$\tilde{\beta}_2^{\text{ECU}} = \max(0, \beta(130, 80, 15), \beta(170, 60, 15)) \quad (10)$$

where β is defined as in (8). This approximation is graphically illustrated in Figure 4. As can be noted, the approximation closely follows the original service curve while being much more amenable to representation using a model checker.

C. UPPAAL MODELS

In this section, we will illustrate the UPPAAL models of the components used in the verification of delay frequency metric.

C.1 Service curve automaton

Recall that we proposed the approximation of a generic service curve by a set of periodic service curves. The automaton shown in Figure 6 represents the working of a periodic service curve $\beta(o_i, P_i, Q_i)$. It maintains a variable $\texttt{fill}_i$ which logs how the resource can be used. The initial value of $\texttt{fill}_i$ is set to $\frac{-o_i}{P_i}Q_i$. The resource *should* serve any pending task if the variable $\texttt{fill}_i$ is positive. If the task is indeed served, the $\texttt{fill}_i$ level is decreased by the amount of execution provided to the task. During busy intervals, the fill level is periodically increased by Q_i every P_i units of time. When the task queue is empty, the fill level is reset to the initial value.

Several such periodic service curves can be easily combined to represent a single service defined as in (5). This is done by implicitly maintaining a single `fill` given as

$$\texttt{fill} = \max_i(\texttt{fill}_i). \quad (11)$$

The resource serves any pending task if any of the fill values is positive. Executing a task will decrease each $\texttt{fill}_i$ value, independently. This is abstractly depicted in the resource automaton shown in Figure 7. Here the automaton, in the location `check` decides if pending tasks are to be served at all. If any of the $\texttt{fill}_i$ values is positive, the pending task is served for one time unit and the $\texttt{fill}_i$ values are all decremented by 1.

Note two aspects of this system. Firstly, the resource is allowed to execute a task even if the all $\texttt{fill}_i$ values are negative. This demonstrates the non-determinism in the definition of the service curve. Secondly, note that the above automatons actually represent constraints on every interval of time, but has a very efficient and compact UPPAAL component. This enables verification of properties such as the delay frequency metric.

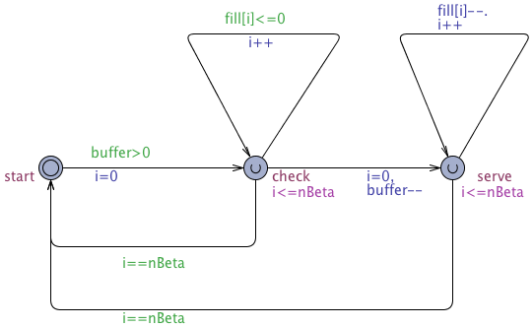

Figure 7: UPPAAL component for the resource obtained as the combination of multiple periodic curves

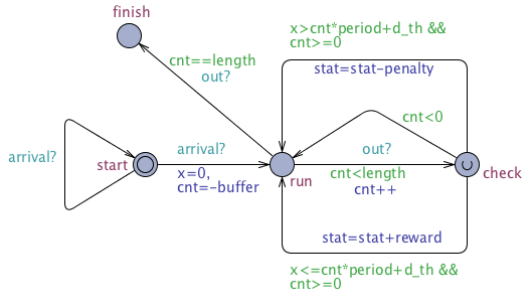

Figure 8: UPPAAL component for the observer

C.2 Observer automaton

The observer automaton is shown in Figure 8. The automaton depends on two synchronization channels `arrival` and `out` which denote the arrival of a new task and the completion of a task, respectively. The observer chooses to observe from some particular task arrival, non-deterministically. It first waits for already buffered tasks to be completed (while `cnt` is negative). Then, it begins to check whether tasks have a delay larger or smaller than d_{th}. For every invalid sample it decrements `stat` by `penalty` and for every valid sample it increments `stat` by `reward`. The variable `stat` is initialized to `penalty`. Then, the query asserting if `stat` is always non-negative, asserts whether the system guarantees a delay frequency metric $(d_{th}, \frac{penalty}{reward})$.

D. COMPUTATION OF A_{CL}

In this section, we show the computation of closed-loop system matrix A_{cl} with $u[k] = Kx[k-3]$ for the control loop P_2 with $\left\lceil \frac{d_{th}}{p} \right\rceil = 3$. We define the new system states,

$$\begin{aligned} z_1[k] &= x[k-3], \\ z_2[k] &= x[k-2], \\ z_3[k] &= x[k-1], \\ z_4[k] &= x[k]. \end{aligned} \tag{12}$$

Hence, we have the following system with $u[k] = Kx[k-3] = Kz_1[k]$,

$$\begin{aligned} z_1[k+1] &= z_2[k], \\ z_2[k+1] &= z_3[k], \\ z_3[k+1] &= z_4[k], \\ z_4[k+1] &= Az_4[k] + BKz_1[k]. \end{aligned} \tag{13}$$

The new system states are $z[k] = [z_1[k]\ z_3[k]\ z_3[k]\ z_4[k]]^T$ and closed-loop system is,

$$z[k+1] = A_{cl}z[k], \tag{14}$$

where

$$A_{cl} = \begin{bmatrix} 0_{2\times2} & I_{2\times2} & 0_{2\times2} & 0_{2\times2} \\ 0_{2\times2} & 0_{2\times2} & I_{2\times2} & 0_{2\times2} \\ 0_{2\times2} & 0_{2\times2} & 0_{2\times2} & I_{2\times2} \\ BK & 0_{2\times2} & 0_{2\times2} & A \end{bmatrix}. \tag{15}$$

To maintain equal dimensionality for later matrix operations, the open-loop matrix A is computed by putting $K = 0$ in (15).

E. ADT APPROACH

We have two subsystems: A_{cl} and A. A_{cl} is the stabilizing subsystem using control input $u[k] = Kx[k - \left\lceil \frac{d_{th}}{p} \right\rceil]$ (computation of A_{cl} is shown in Appendix D) whereas A is the open-loop system with $u[k] = 0$. The switching between A_{cl} and A depends on the occurrence of valid and invalid samples. With arbitrary oder of valid and invalid samples, we get,

$$x[k] = A_{cl}^{n_1} A^{n_2} A_{cl}^{n_3} A^{n_4} A_{cl}^{n_5} A^{n_6} ... x[0], \tag{16}$$

where n_i are integers such that $\sum_i n_i = k$ (integer). Using the analysis coming from Average Dwell Time (ADT) [15], one can answer questions like what combination of n_i guarantees stability of system (16). We follow similar technique to estimate how frequently invalid samples are allowed to occur in the context of our setting.

Considering the example control loop P_2, $\left\lceil \frac{d_{th}}{p} \right\rceil = 3$, $u[k] = Kx[k-3]$ and $K = [0 \quad -0.28]$, we know that the following inequality holds true,

$$\left\| A_{cl}^k \right\| \leq 23.5 \times 0.7538^k. \tag{17}$$

Similar property can be derived for the open-loop system,

$$\left\| A^k \right\| \leq 1.6 \times 1.0539^k. \tag{18}$$

To find how many valid samples must occur for every occurrence of invalid sample, system (16) becomes,

$$x[k] = A_{cl}^{n_1} A A_{cl}^{n_2} x[0]. \tag{19}$$

For the stability of the system, we need to make sure that $\|x[k]\| < \|x[0]\|$ for any k and $n = n_1 + n_2$. That is, the following must be true for assuring stability of (19),

$$\begin{aligned} \|A_{cl}^{n_1} A A_{cl}^{n_2}\| &\leq 1, \\ 23.5^2 \times 1.6 \times 0.7538^n \times 1.0539 &< 1, \\ n &> 24.2. \end{aligned} \tag{20}$$

Hence, we conclude that ADT approach allows a 1 invalid sample within a window of any 25 samples.

F. CQLF APPROACH

We illustrate the stability analysis of the system (9). We perform the analysis considering that the control loop P_2 is implemented over an architecture described in Section 2.1 and the delay frequency metric to be $(300\ ms, 2)$. At the valid samples, the closed-loop system is A_{cl} as shown in Appendix D whereas the corresponding system for invalid samples is A as per eq. (2). From (9), we have a system which switches among various subsystems A_j,

$$A_j = AA_{cl}^{n_i}, \forall n_i \geq 2. \tag{21}$$

We resort to the concept of strong CQLF [9] to investigate the stability of the switched system (9). Towards this, we present the following standard theorems of LTI systems which are utilized in our analysis.

THEOREM F.1. (discrete-time Lyapunov equation [10]) *Let $A \in \mathbb{R}^{n\times n}$. If there exists $P = P^T > 0$, $Q = Q^T > 0$ satisfying $A^TPA - P = -Q$, then all eigenvalues of matrix A are inside the unit circle (or the system is stable in our context).*

THEOREM F.2. (continuous-time Lyapunov equation [10]) *Let $A \in \mathbb{R}^{n\times n}$. If there exists $P = P^T > 0$, $Q = Q^T > 0$ satisfying $A^TP + PA = -Q$, then matrix A is Hurwitz (or the system is stable in our context).*

LEMMA F.1. (Cayley transform [9]) *Let $A \in \mathbb{R}^{n\times n}$, $P = P^T > 0$, $Q = Q^T > 0$. Let P be the solution of $A^TPA - P = -Q$. Consider the matrix,*

$$C(A) = (A - I)(A + I)^{-1}. \tag{22}$$

Then P is also a solution of the continuous-time Lyapunov equation $C(A)^TP + PC(A) = -Q'$ with $Q' = 2(A+I)^{-T}Q(A+I)^{-1}$.

LEMMA F.2. ([9]) *Let A_1, A_2 be the system matrices of two stable discrete-time LTI systems with strong CQLF given by $V(z) = z^TPz$, i.e.,*

$$A_1^TPA_1 - P = -Q_1 < 0,$$
$$A_2^TPA_2 - P = -Q_2 < 0.$$

Then the two matrices $C(A_1)C(A_2)$ and $C(A_1)C(A_2)^{-1}$ have no real negative eigenvalues.

LEMMA F.3. *Let A_1, A_2 be the system matrices of two stable discrete-time LTI systems with strong CQLF given by $V(z) = z^TPz$, i.e.,*

$$A_1^TPA_1 - P = -Q_1 < 0,$$
$$A_2^TPA_2 - P = -Q_2 < 0.$$

Then the switching between the two LTI systems is asymptotically stable.

PROOF. We have two discrete-time LTI systems (23) and (24). The switching between them is modeled as a new LTI system shown in (25).

$$z[k+1] = A_1z[k], \tag{23}$$
$$z[k+1] = A_2z[k], \tag{24}$$
$$z[k+2] = A_2A_1z[k]. \tag{25}$$

For the stability of (25), A_2A_1 should have all eigenvalues inside the unit circle, i.e., $A_1^TA_2^TPA_2A_1 - P < 0$ (utilizing Theorem F.1). Towards this,

$$\begin{aligned} &A_1^TA_2^TPA_2A_1 - P \\ &= A_1^T(P - Q_2)A_1 - P \\ &= A_1^TPA_1 - P - A_1^TQ_2A_1 \\ &= -Q_1 - A_1^TQ_2A_1 < 0 \end{aligned}$$

Hence, the switching between (23) and (24) is stable and the proof is complete. □

Hence, there exists a strong CQLF between any two given subsystems A_i and A_j (hence, switching is stable) if $C(A_i)C(A_j)$ and $C(A_i)C(A_j)^{-1}$ do not have any real negative eigenvalue. Thus, it is possible to verify switching stability of (9) for the class of subsystems of form (21) for up to any n_i of design interest. For the control loop P_2 under consideration, the delay frequency metric $(300\ ms, 2)$ results in stable switching for all $n_i \geq 2$.