Self-Triggered Controllers and Hard Real-Time Guarantees

Amir Aminifar[1], Paulo Tabuada[2], Petru Eles[1], Zebo Peng[1]

[1]Department of Computer and Information Science, Linköping University, Sweden
[2]Department of Electrical Engineering, University of California at Los Angeles, USA

***Abstract*—It is well known that event-triggered and self-triggered controllers implemented on dedicated platforms can provide the same performance as the traditional periodic controllers, while consuming considerably less bandwidth. However, since the majority of controllers are implemented by software tasks on shared platforms, on one hand, it might no longer be possible to grant access to the event-triggered controller upon request. On the other hand, due to the seemingly irregular requests from self-triggered controllers, other applications, while in reality schedulable, may be declared unschedulable, if not carefully analyzed. The schedulability and response-time analysis in the presence of self-triggered controllers is still an open problem and the topic of this paper.**

***Keywords*- schedulability analysis, response-time analysis, self-triggered control, event-triggered control.**

I. Introduction and Related Work

Self-triggered and event-triggered controllers are being actively considered as substitutes of traditional periodic controllers. As opposed to the traditional controllers where sampling is periodic in the time domain, typically, the event-triggered and self-triggered controllers execute when the expected performance is about to be violated. This, in turn, leads to less resource usage compared to the traditional periodic controllers since the controllers execute only if it is necessary to guarantee the expected performance.

Today, many control applications are implemented on shared platforms, alongside other hard real-time or safety-critical applications. The control-scheduling co-design of traditional periodic controllers has been investigated for more than a decade [1]–[9]. There has also been considerable amount of research on event-triggered and self-triggered control mechanisms [10]–[17]. In the case of self-triggered controllers, however, *the lack of clear execution patterns* has been the main obstacle in efficiently implementing such applications alongside hard real-time applications on shared platforms. Due to the seemingly irregular requests from self-triggered controllers, current practice often leads to under-utilized resources and over-provisioned designs, which defeats the purpose of the self-triggered control, i.e., less resource usage.

In this paper, we discuss the fact that self-triggered controllers actually exhibit certain execution patterns when carefully examined. Note that the next triggering instant for the self-triggered controllers depends on the state of the plant. The core idea here is to capture the dependency between the states of the plant at each triggering point. This means that, for each initial state, the following states are not arbitrary, and exploiting this fact results in a less pessimistic analysis. This, in particular, is important from the schedulability point of view.

A naive approach to schedulability analysis is to consider the least inter-execution time with respect to all initial states the plant can be in. To perform schedulability analysis, then it is safe to consider the self-triggered controller as a periodic task with the least inter-execution time. However, this is an overly pessimistic analysis since in every step it considers the worst-case possible state for the plant (with respect to inter-execution time). In this case, the calculated interference from the self-triggered controller is considerably larger than what occurs in reality, since always the worst-case scenario is considered, eliminating the potential advantage of self-triggered controllers versus the periodic controllers. And, essentially, this leads to a pessimistic analysis method.

Over the last few years, schedulability analysis of self-triggered controllers has gained attention. Velasco et. al. [18] considered the problem under both fixed-priority and earliest-deadline-first scheduling policies. However, the problem of finding the worst-case triggering pattern was left open. Lemmon et. al. [19] considered online scheduling of self-triggered controllers using elastic scheduling, but no stability guarantees are provided. Anta and Tabuada [20] discussed the benefits of relaxing periodicity constraint over communication networks. However, to provide schedulability guarantees, the authors consider the minimum inter-arrival time for all possible initial states, which is extremely pessimistic and defeats the purpose of the self-triggered controller. Antunes and Heemels [21] found the optimal sampling instants and control inputs in a given interval with respect to quadratic cost functions. However, they do not attempt to address the schedulability and response-time analysis problem. Finally, it has been shown that in certain self-triggered schemes no positive minimum inter-event time can be guaranteed [22].

In this paper, we focus on the self-triggered approach proposed by Donkers et. al. [23], adapted for real-time analysis. We address the response-time and schedulability analysis for a mixed set of periodic hard real-time tasks and self-triggered control tasks. The basic idea is to make use of the fact that there actually exist certain patterns in the execution of self-triggered controllers. To our knowledge, this is the first attempt to perform offline schedulability analysis in the presence of self-triggered control tasks that allows to leverage the potential advantages of self-triggered control compared to periodic control.

II. System Model

A. Task Model

Given is an independent taskset, where each task is denoted by τ_i. The computation time (execution time) and priority of task τ_i are denoted by c_i and ρ_i, respectively. Task τ_i has higher priority than task τ_j iff $\rho_i > \rho_j$. The set of higher priority tasks for task τ_i is denoted by $hp(\tau_i)$.

The j^{th} instance (job) of task τ_i is denoted by $\tau_{i,j}$. The inter-arrival time (or inter-execution) between the two instances $\tau_{i,j}$ and $\tau_{i,j+1}$ is denoted by $h_{i,j}$. It is clear that for a periodic task τ_i, $h_{i,j} = h_{i,k}$, $\forall j, k$, which means that the inter-arrival time is constant for the periodic tasks. Therefore, for the periodic task τ_i, we drop the index j for the period $h_{i,j}$ when convenient and denote the period by h_i.

The worst-case interference (in terms of the number of instances) of task τ_i in an interval of length t is denoted by $I_i(t)$.

B. Plant Model

The plant associated with a self-triggered control task τ_i is modeled by a continuous-time system of differential equations [24],

$$\dot{\boldsymbol{x}}_i = \boldsymbol{A}_i \boldsymbol{x}_i + \boldsymbol{B}_i \boldsymbol{u}_i, \tag{1}$$

where $\boldsymbol{x}_i$ and $\boldsymbol{u}_i$ are the plant state and the control signal, respectively.

C. Self-Triggered Controller

In event-based control, the plant is constantly monitored and a new control input is applied only if the performance requirements of the plant are about to be violated. This is as opposed to the periodic scheme, where the plant is controlled uniformly in the time domain. The self-triggered control was first introduced in [12]. In self-triggered control, as opposed to constant monitoring of the plant, at each execution instant, in addition to computing the control input, the controller also computes the latest instant at which a new control input should be applied in order to guarantee the required control performance. And, this is the next execution instant for the self-triggered controller.

The inter-execution time for a self-triggered controller depends on the current state of the plant, the dynamics of the plant, and the performance metrics used.

III. Problem Formulation

Given a mixed set of self-triggered control tasks and periodic hard real-time tasks, we would like to find out if all hard real-time tasks meet their deadlines and all plants associated with the self-triggered controllers are guaranteed to be stable, under the fixed-priority scheduling policy.

The main step towards schedulability analysis is to find the worst-case scenario of triggering of a single self-triggered controller. The worst-case scenario, in this context, refers to the triggering scenario of the controller that produces the maximum interference on other tasks.

IV. The Self-Triggered Controller

In this section, we shall briefly discuss how our self-triggered scheme works. The approach comprises of an offline design time step which is prior to the actual execution of self-triggered control task, and an online step, where the next execution instant and the control input are determined at runtime.

A. Offline Step

At design time, the state space of the plant is partitioned into several convex polytopes. For each polytope, we shall calculate the maximum time that the plant could run in open-loop before instability (i.e., violating the expected performance), considering that the initial state could be anywhere in the polytope. Moreover, we shall find the polytopes in which the plant state could end up after it runs in open-loop for this amount of time. This information will be encoded in the form of a transition graph, where each node corresponds to one polytope. The transitions among the polytopes are captured by edges and the weight associated with each edge is the maximum time the plant could run in open-loop. For instance, an edge from node p to node q with weight h indicates that: if the initial state of the plant is in polytope p, the plant can run in open-loop for h time units and the final state of the plant after h time units could be in polytope q. We discuss these techniques further in Sections V and VI.

B. Online Step

At runtime, the initial state $\boldsymbol{x}(0)$ is known. Every time the self-triggered controller is executed, there are two procedures to be performed: (1) to compute the next time the self-triggered controller needs to execute, and (2) to compute the constant control input until the next execution.

To this end, we shall first find the polytope to which the initial state $\boldsymbol{x}(0)$ belongs. Note that there exist efficient algorithms to determine if a point is inside a convex polytope.

The corresponding control input and the next time the controller needs to execute for an initial state inside the polytope are obtained based on a slightly modified version of the self-triggered controller approach in [23]. From the transition graph, we know that if the initial state is in a particular polytope, the trajectory can only end up in a subset of polytopes. Then, we shall solve the minimum attention control problem [23], but enforcing extra constraints such that the final state of the plant after running in open-loop is guaranteed to be within this subset of polytopes. The problem remains a linear feasibility problem and, therefore, is of the same complexity order.

It is of significant importance to observe that the maximum time h the plant can run in open-loop, which is calculated at runtime, may be longer or equal to what is indicated in the transition graph. That is, the actual interference of the self-triggered controller at runtime may not be larger than the interference found in the offline step, and hence the safety of our offline real-time analysis is preserved.

V. The Big Picture

Under fixed-priority preemptive scheduling, assuming constrained deadlines (deadlines less than or equal to the period) and an independent taskset with periodic tasks, the exact worst-case response time of a task τ_i, denoted by R_i, is computed by the following equation [25],

$$R_i = c_i + \sum_{\tau_j \in hp(\tau_i)} \left\lceil \frac{R_i}{h_j} \right\rceil c_j, \tag{2}$$

where $hp(\tau_i)$ denotes the set of higher priority tasks for task τ_i. Since Equation 2 cannot be solved analytically, it has to be solved by fixed-point iteration (starting with, e.g., $R_i = c_i$) and has pseudo-polynomial complexity. The above can be extended to periodic tasks with arbitrary deadlines [26].

In Equation (2), since only periodic tasks are considered, we have $I_j(R_i) = \left\lceil \frac{R_i}{h_j} \right\rceil$. To consider the self-triggered tasks as well, Equation (2) can be rewritten as follows,

$$R_i = c_i + \sum_{\tau_j \in hp(\tau_i)} I_j(R_i) \cdot c_j, \tag{3}$$

where $I_j(t)$ is the maximum interference of the higher priority task τ_j in an interval of length t.

It should now be clear that the schedulability problem is reduced to finding the worst-case interference scenario in an interval of length t for a single self-triggered task τ_i, i.e., $I_i(t)$. In other words, we would like to find the *request bound function* for each self-triggered controller. For the sake of presentation, in the next section, we shall only consider one single self-triggered task and, therefore, we can drop the index identifying the task.[1]

VI. Finding Request Bound Function

In this section, we shall discuss the design time procedure to find the request bound function for a single self-triggered control task. Towards this, first we shall divide the state space into several subregions. Then, it is determined if at runtime a transition from one subregion to another subregion is possible, and this information is modeled as a graph (see Section VI-A). The second step is to use dynamic programming in order to find the shortest interval of time with at least k triggering events, from which we compute the request bound function (see Section VI-B).

A. Extraction of the Transition Graph

To find the transition graph, we shall take three steps described in the following subsections:

1) Partitioning the state space

In this step, we shall partition the state space of the plant into m convex polytopes. The main idea is to partition the state space such that each component of the state space has the same sign in the entire polytope and the dominating (maximum in terms of absolute value) component of the state space remains the same. For example, for the two-dimensional state space we have,

$$\boldsymbol{y} = \begin{bmatrix} \boldsymbol{y}_1 \\ \boldsymbol{y}_2 \end{bmatrix} = \begin{bmatrix} \boldsymbol{P}_{11} & \boldsymbol{P}_{12} \\ \boldsymbol{P}_{21} & \boldsymbol{P}_{22} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{bmatrix} = \boldsymbol{P}\boldsymbol{x}.$$

The polytopes are identified by the following lines:

$$\begin{aligned}
&\boldsymbol{y}_1 = 0 \Rightarrow \boldsymbol{P}_{11}\boldsymbol{x}_1 + \boldsymbol{P}_{12}\boldsymbol{x}_2 = 0, \\
&\boldsymbol{y}_2 = 0 \Rightarrow \boldsymbol{P}_{21}\boldsymbol{x}_1 + \boldsymbol{P}_{22}\boldsymbol{x}_2 = 0, \\
&\boldsymbol{y}_1 = \boldsymbol{y}_2 \Rightarrow (\boldsymbol{P}_{11} - \boldsymbol{P}_{21})\boldsymbol{x}_1 + (\boldsymbol{P}_{12} - \boldsymbol{P}_{22})\boldsymbol{x}_2 = 0, \\
&\boldsymbol{y}_1 = -\boldsymbol{y}_2 \Rightarrow (\boldsymbol{P}_{11} + \boldsymbol{P}_{21})\boldsymbol{x}_1 + (\boldsymbol{P}_{12} + \boldsymbol{P}_{22})\boldsymbol{x}_2 = 0.
\end{aligned}$$

[1]Note that there are no limiting assumptions on the number of self-triggered controllers or the priorities assigned to them.

The first two lines make sure that the sign of each component of vector $\boldsymbol{y}$ remains the same in each polytope, whereas the next two lines partition the state space such that in each polytope, the infinity norm of $\|\boldsymbol{y}\|_\infty$ always depends on one component of vector $\boldsymbol{y}$. The vertices of the polytopes are the origin and where the lines cross the boundary of the state space (see also Section VIII for an example). The generalization of the above partitioning technique to higher dimensions is trivial.

Although it is possible to partition the state space even further [27], [28], for the simplicity of presentation, we shall only consider this partitioning throughout this paper.

2) Calculation of the maximum time h for each polytope

For each polytope, we shall calculate the maximum time h that the plant could run in open-loop before instability (i.e., violating the expected control performance), considering that the initial state could be anywhere in the polytope. This is done based on a slightly modified version of the proposed approach in [23].

The plant is guaranteed to be stable after running in open-loop for h time units if,

$$V(\boldsymbol{x}(h)) - e^{-\alpha h} V(\boldsymbol{x}(0)) \leq 0, \tag{4}$$

where $V(\cdot)$ denotes the Lyapunov function. Similar to [23], here, we consider the Lyapunov function $V(\boldsymbol{x}) = \|\boldsymbol{P}\boldsymbol{x}\|_\infty$,

$$\|\boldsymbol{P}\boldsymbol{x}(h)\|_\infty - e^{-\alpha h} \|\boldsymbol{P}\boldsymbol{x}(0)\|_\infty \leq 0. \tag{5}$$

The plant state $\boldsymbol{x}$ at time h is as follows, assuming constant control input $\boldsymbol{u}$ in the interval $[0, h)$,

$$\begin{aligned}
\boldsymbol{x}(h) &= e^{\boldsymbol{A}h}\boldsymbol{x}(0) + \int_0^h e^{\boldsymbol{A}(h-t)} dt \boldsymbol{B}\boldsymbol{u}, \\
&= \boldsymbol{\Phi}(h)\boldsymbol{x}(0) + \boldsymbol{\Gamma}(h)\boldsymbol{u}.
\end{aligned} \tag{6}$$

where

$$\begin{aligned}
\boldsymbol{\Phi}(h) &= e^{\boldsymbol{A}h}, \\
\boldsymbol{\Gamma}(h) &= \int_0^h e^{\boldsymbol{A}(h-t)} dt \boldsymbol{B}.
\end{aligned}$$

To find the maximum time h where the plant could run in open-loop, for a given initial state $\boldsymbol{x}(0)$, h is increased iteratively until there does not exist any control input to satisfy inequality (5).

Let us assume for two vertices of the polytopes, namely $\underline{\boldsymbol{x}}(0)$ and $\overline{\boldsymbol{x}}(0)$, constraint (5) is satisfied if the system runs in open-loop for h time units,

$$\begin{aligned}
&\|\boldsymbol{P}(\boldsymbol{\Phi}\underline{\boldsymbol{x}}(0) + \boldsymbol{\Gamma}\underline{\boldsymbol{u}})\|_\infty - \sigma \|\boldsymbol{P}\underline{\boldsymbol{x}}(0)\|_\infty \leq 0, \\
&\|\boldsymbol{P}(\boldsymbol{\Phi}\overline{\boldsymbol{x}}(0) + \boldsymbol{\Gamma}\overline{\boldsymbol{u}})\|_\infty - \sigma \|\boldsymbol{P}\overline{\boldsymbol{x}}(0)\|_\infty \leq 0,
\end{aligned} \tag{7}$$

with constant $\sigma(h) = e^{-\alpha h}$.

Now we should show that for $\boldsymbol{x}(0) = \lambda\underline{\boldsymbol{x}}(0) + (1 - \lambda)\overline{\boldsymbol{x}}(0)$, with $0 \leq \lambda \leq 1$, constraint (5) is satisfied, i.e.,

$$\|\boldsymbol{P}(\boldsymbol{\Phi}\boldsymbol{x}(0) + \boldsymbol{\Gamma}\boldsymbol{u})\|_\infty - \sigma \|\boldsymbol{P}\boldsymbol{x}(0)\|_\infty \leq 0. \tag{8}$$

Note that, in general, the above inequality does not hold. However, within each polytope, thanks to the careful partitioning in the first step, we have,

$$\begin{aligned}
\|\boldsymbol{P}\boldsymbol{x}(0)\|_\infty &= \|\boldsymbol{P}(\lambda\underline{\boldsymbol{x}}(0) + (1 - \lambda)\overline{\boldsymbol{x}}(0))\|_\infty \\
&= \lambda \|\boldsymbol{P}\underline{\boldsymbol{x}}(0)\|_\infty + (1 - \lambda) \|\boldsymbol{P}\overline{\boldsymbol{x}}(0)\|_\infty.
\end{aligned} \tag{9}$$

Algorithm 1 Worst-Case Response-Time Analysis R_i

1: Initialization: $R_i = c_i$; $t = 0$;
2: Initialization: $k_j = 1$; $s_j(1,p) = 0, \forall p, j$;
3: **while** $t < R_i$ **do**
4: $\quad t = R_i$;
5: $\quad R_i = c_i$;
6: $\quad$ **for all** $\tau_j \in hp(\tau_i)$ **do**
7: $\quad\quad$ **if** τ_j is periodic **then**
8: $\quad\quad\quad I_j = \left\lceil \frac{t}{h_j} \right\rceil$;
9: $\quad\quad$ **else**
10: $\quad\quad\quad$ **while** $s_j(k_j) \leq t$ **do**
11: $\quad\quad\quad\quad k_j = k_j + 1$;
12: $\quad\quad\quad\quad s_j(k_j, p) = \min\limits_{q=1\ldots m_j} \{s_j(k_j - 1, q) + \boldsymbol{G}_j(q,p)\}$;
13: $\quad\quad\quad\quad s_j(k_j) = \min\limits_{p=1\ldots m_j} \{s(k_j, p)\}$;
14: $\quad\quad\quad$ **end while**
15: $\quad\quad\quad I_j = k_j - 1$;
16: $\quad\quad$ **end if**
17: $\quad\quad R_i = R_i + I_j \cdot c_j$;
18: $\quad$ **end for**
19: **end while**
20: **return** R_i

Let us assume that the control input is given by $\boldsymbol{u} = \lambda\underline{\boldsymbol{u}} + (1-\lambda)\overline{\boldsymbol{u}}$. Using the triangular property of norms and Equation (9), we can show that inequality (8) is satisfied,

$$\begin{aligned}&\left\|\boldsymbol{P}(\boldsymbol{\Phi x}(0) + \boldsymbol{\Gamma u})\right\|_\infty - \sigma \left\|\boldsymbol{P x}(0)\right\|_\infty \leq \\ &\lambda \left(\left\|\boldsymbol{P}(\boldsymbol{\Phi}\underline{\boldsymbol{x}}(0) + \boldsymbol{\Gamma}\underline{\boldsymbol{u}})\right\|_\infty - \sigma \left\|\boldsymbol{P}\underline{\boldsymbol{x}}(0)\right\|_\infty\right) + \\ &(1-\lambda)\left(\left\|\boldsymbol{P}(\boldsymbol{\Phi}\overline{\boldsymbol{x}}(0) + \boldsymbol{\Gamma}\overline{\boldsymbol{u}})\right\|_\infty - \sigma \left\|\boldsymbol{P}\overline{\boldsymbol{x}}(0)\right\|_\infty\right) \leq 0.\end{aligned}$$

This implies that if the system can run in open-loop for h time units considering the initial state to be any of the vertices of the convex polytope, then for any initial state within the convex polytope also the system can run in open-loop for h time units.

3) *Construction of the transition graph*

In this step, we shall construct the graph $\boldsymbol{G}$ corresponding to the transitions between the polytopes. Since the systems considered in this paper are linear, the convex polytopes after the system runs in open-loop will be mapped to convex polytopes.

The new polytopes are easily found by considering the dynamics of the system (6) for the vertices of the initial convex polytopes [29]. Note that the transition graph $\boldsymbol{G}(p,q) = h$, if the p^{th} polytope after h time unit, which was found in the previous step, has overlap with the q^{th} polytope. And $\boldsymbol{G}(p,q) = +\infty$, if there can be no transition from the p^{th} polytope to the q^{th} polytope.

B. Extraction of the Worst-Case Request Pattern

Having the transition graph, it is now possible to find the worst-case request bound function. To this end, we shall use dynamic programming. Note that the length of the shortest interval of time with k triggers inside, denoted by $s(k)$, is obtained as follows:

$$\begin{aligned} s(k,p) &= \min_{q=1\ldots m} \left\{s(k-1,q) + \boldsymbol{G}(q,p)\right\}, \\ s(k) &= \min_{p=1\ldots m} \left\{s(k,p)\right\}, \end{aligned} \tag{10}$$

where $s(k,p)$ is the shortest path with k nodes, which ends in the p^{th} node of the graph. Equivalently, $s(k,p)$ is the shortest interval of time with k triggers, which ends in the p^{th} polytope. From the structure of Equation (10), it can be observed that this problem could be solved using dynamic programming.

Table I
EXAMPLE: TASKSET DATA

i	ρ_i	c_i	h_i	d_i	self-triggered/hard real-time
1	3	0.3	–	0.8	self-triggered
2	2	1.0	2.0	2.0	hard real-time
3	1	1.0	6.0	6.0	hard real-time

The request bound function $I(t)$ has to be calculated by computing the pseudo-inverse of $s(k)$,

$$I(t) = s^{-1}(t), \tag{11}$$

where $I(t)$ is the maximum number of requests in any interval of length t.

VII. SCHEDULABILITY ANALYSIS

In this section, we shall perform schedulability analysis for the self-triggered controllers. Having found the request bound function $I_i(t)$ for each self-triggered task τ_i, we shall now introduce Algorithm 1 to compute the worst-case response-time of a task. If all tasks have worst-case response-times less than their deadlines, the system is schedulable.

For each hard real-time task τ_i, execution time c_i, period h_i, deadline d_i, and the set of higher priority tasks $hp(\tau_i)$ are known. However, for each self-triggered control task τ_i, only execution time c_i, deadline d_i, and the set of higher priority tasks $hp(\tau_i)$ are known. The deadline d_i for a self-triggered task τ_i can be obtained as follows,

$$d_i = \min_{\forall p,q} \left\{\boldsymbol{G}_i(p,q)\right\}. \tag{12}$$

This is based on the fact that each self-triggered job should complete its execution before the next triggering instant, and in the worst-case scenario, we should consider the minimum among all. Observe that there is no pessimism introduced by our approach in computing this deadline.

The worst-case response-time of task τ_i is computed as follows,

$$R_i = c_i + \sum_{\tau_j \in hp(\tau_i)} I_j(R_i) \cdot c_j, \tag{13}$$

For a periodic hard real-time task τ_j, the worst-case interference function is $I_j(t) = \left\lceil \frac{t}{h_j} \right\rceil$. For a self-triggered control task, $I_j(t)$ is calculated based on Equation (10) and Equation (11).

The difference between the proposed algorithm and the traditional response-time analysis algorithm for periodic tasks under fixed-priority analysis is in Lines 10–15, where we compute the number of triggers of the self-triggered controller (interference in terms of number of events) in an interval of length t. Basically, we increase the number of triggers, k, iteratively, until the length of the shortest interval including k triggers is larger than t. Note that Algorithm 1 has the dynamic programming problem in Equation (10) embedded in Lines 12–13 of Algorithm 1.

The algorithm has pseudo-polynomial complexity, similar to response-time analysis for periodic tasks under fixed-priority policy.

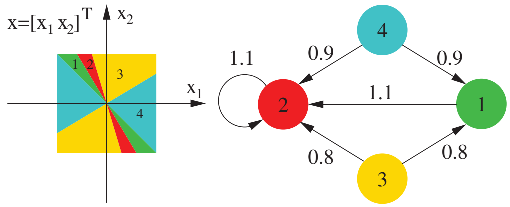

Figure 1. The state space partitioning (on the left) and the corresponding transition graph (on the right).

VIII. Illustrative Example

Let us consider a taskset consisting of three tasks $\mathbf{T} = \{\tau_1, \tau_2, \tau_3\}$. Task τ_1 is a self-triggered control task, has the highest priority and worst-case execution-time $c_1 = 0.3$. Task τ_2 is a hard-real time task with period $h_2 = 2.0$ and worst-case execution-time $c_2 = 1.0$. Task τ_3 is also a hard real-time task with period $h_3 = 6.0$ and worst-case execution-time $c_3 = 1.0$, and has the lowest priority. This information is summarized in Table I. While we consider only one self-triggered task for the simplicity of presentation, the approach is by no means limited to a single self-triggered task.

In this example, we would like to find the response-time of task τ_3, i.e., to check if it is schedulable.

Towards this, we need to find the request bound function for the self-triggered task τ_1. The plant associated with the self-triggered controller is identified by the following matrices (see Equation (1)),

$$\boldsymbol{A} = \begin{bmatrix} 1 & 5 \\ 0 & 2 \end{bmatrix}, \qquad \boldsymbol{B} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

Note that the eigenvalues of this plant are not in the left half of the complex plane and, therefore, the plant is unstable. Finally, we assume that the state space is bounded, i.e., $\|\boldsymbol{x}\|_\infty \leq 1$.

As discussed in Section VI-A, first we partition the space into convex polytopes, as shown in Figure 1. Secondly, for each polytope, we shall find the maximum time the plant could run in open-loop before instability. Thirdly, from this information, we can construct the transition graph. It turns out that the transition graph is as follows,

$$\boldsymbol{G} = \begin{bmatrix} +\infty & 1.1 & +\infty & +\infty \\ +\infty & 1.1 & +\infty & +\infty \\ 0.8 & 0.8 & +\infty & +\infty \\ 0.9 & 0.9 & +\infty & +\infty \end{bmatrix}.$$

The graph corresponding to the above transition matrix is shown in Figure 1. For example, for node 3 (corresponding to polytope 3) of the graph, it can be observed that the plant, in the worst-case, could only run in open-loop for 0.8 time units and after that the trajectory will end up in node 1 (corresponding to polytope 1) or node 2 (corresponding to polytope 2).

Now, using the dynamic programming algorithm discussed in Section VI-B, we can find the worst-case request pattern. The worst-case request bound function (or trigger pattern) is shown in Figure 2. Already at this stage, it is obvious that considering the worst-case scenario with respect to all initial states is in fact very conservative. To clarify this, note that in the worst-case scenario, the minimum inter-arrival time is 0.8 time units. However,

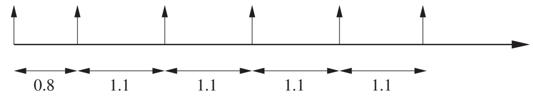

Figure 2. The worst-case triggering (arrival) pattern of the self-triggered controller.

from the transition graph, it is clear that this worst-case scenario can only occur once.

Let us now compute the worst-case response time of task τ_3:

$$\begin{aligned}
R_3^0 &= c_3 = 1; \\
R_3^1 &= c_3 + \sum_{\tau_j \in hp(\tau_i)} I_j(R_3^0) \cdot c_j = 1 + 2 \cdot 0.3 + 1 \cdot 1 = 2.6, \\
R_3^2 &= c_3 + \sum_{\tau_j \in hp(\tau_i)} I_j(R_3^1) \cdot c_j = 1 + 3 \cdot 0.3 + 2 \cdot 1 = 3.9, \\
R_3^3 &= c_3 + \sum_{\tau_j \in hp(\tau_i)} I_j(R_3^2) \cdot c_j = 1 + 4 \cdot 0.3 + 2 \cdot 1 = 4.2, \\
R_3^4 &= c_3 + \sum_{\tau_j \in hp(\tau_i)} I_j(R_3^3) \cdot c_j = 1 + 5 \cdot 0.3 + 3 \cdot 1 = 5.5, \\
R_3^5 &= c_3 + \sum_{\tau_j \in hp(\tau_i)} I_j(R_3^4) \cdot c_j = 1 + 6 \cdot 0.3 + 3 \cdot 1 = 5.8, \\
R_3^6 &= c_3 + \sum_{\tau_j \in hp(\tau_i)} I_j(R_3^5) \cdot c_j = 1 + 6 \cdot 0.3 + 3 \cdot 1 = 5.8.
\end{aligned}$$

The worst-case response-time of task τ_3 is $R_3 = 5.8$ and, therefore, the task meets its deadline $d_3 = 6.0$. This scenario is shown in Figure 3. The green task is the self-triggered task τ_1, the red task is periodic hard real-time task τ_2, and the blue task is τ_3 for which we would like to find the worst-case response-time.

Lastly, let us also consider the state of the art approach which considers the worst-case inter-arrival time with respect to all initial states in every step, i.e., to ignore the dependency between states. In this approach, the self-triggered task is modeled as a periodic task with period $h_1 = \min_{\forall p,q} \{\boldsymbol{G}(p,q)\} = 0.8$. Based on the real-time schedulability analysis for periodic tasks, in this case, the worst-case response-time of task τ_3 is not finite, i.e., it misses its deadline $d_3 = 6.0$ and the system is deemed unschedulable, since the total taskset utilization $\sum_{i=1}^{3} \frac{c_i}{h_i}$ is above 1. The system designer in such situations either needs to, unnecessarily, remove some of the tasks to guarantee schedulability or, again unnecessarily, to use a processor which is faster. Either way, this leads to an over-provisioned design and under-utilized resource.

This example demonstrates the importance of performing tight schedulability analysis in the presence of self-triggered controllers and the efficiency of our proposed approach. Note that even though we considered a single self-triggered control task at the highest priority level, our approach is by no means limited to this case and there is absolutely no restricting assumption on the number of self-triggered controllers or the priorities assigned to these controllers.

In the original self-triggered schemes, it is often assumed that the computation of the control input and the next activation instant is instantaneous. However, this is different from what occurs in practice. To account for the delay experienced by each self-triggered task, at each execution, the self-triggered task computes the plant state at the next

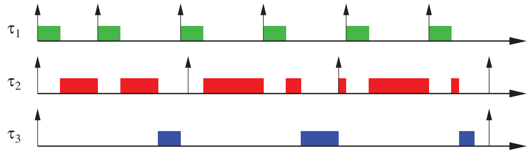

Figure 3. The worst-case response time of task τ_3.

triggering instant based on the dynamics of the systems and current control input. Then, based on the plant state at the next triggering instant, the control input after the next triggering instant and the amount of time the plant could run in open-loop after the next triggering instant are calculated.

IX. Conclusions

The lack of efficient schedulability analysis of real-time systems in the presence of self-triggered controllers has been the main obstacle in implementing such applications alongside hard real-time applications on shared platforms. In this paper, we have proposed an approach for response-time analysis in the presence of self-triggered control tasks, under fixed-priority scheduling policy. The proposed approach can be extended to other scheduling policies (e.g., earliest-deadline-first) and task models (e.g., digraph or arbitrary deadlines) [26], [30].

Acknowledgements

The authors would like to acknowledge Prof. M.C.F. Donkers for providing us with the MATLAB implementation of their proposed approach in [23]. The authors also would like to acknowledge the Swedish national computer science graduate school (CUGS) for supporting the visit to the University of California at Los Angeles, USA, which led to these research results.

References

[1] D. Seto, J. P. Lehoczky, L. Sha, and K. G. Shin, "On task schedulability in real-time control systems," in *Proceedings of the* 17^{th} *IEEE Real-Time Systems Symposium*, 1996, pp. 13–21.

[2] H. Rehbinder and M. Sanfridson, "Integration of off-line scheduling and optimal control," in *Proceedings of the* 12^{th} *Euromicro Conference on Real-Time Systems*, 2000, pp. 137–143.

[3] A. Cervin, B. Lincoln, J. Eker, K. E. Årzén, and G. Buttazzo, "The jitter margin and its application in the design of real-time control systems," in *Proceedings of the* 10^{th} *International Conference on Real-Time and Embedded Computing Systems and Applications*, 2004, pp. 1–10.

[4] T. Nghiem, G. J. Pappas, R. Alur, and A. Girard, "Time-triggered implementations of dynamic controllers," in *Proceedings of the* 6^{th} *ACM & IEEE International conference on Embedded software*, 2006, pp. 2–11.

[5] E. Bini and A. Cervin, "Delay-aware period assignment in control systems," in *Proceedings of the* 29^{th} *IEEE Real-Time Systems Symposium*, 2008, pp. 291–300.

[6] F. Zhang, K. Szwaykowska, W. Wolf, and V. Mooney, "Task scheduling for control oriented requirements for cyber-physical systems," in *Proceedings of the* 29^{th} *IEEE Real-Time Systems Symposium*, 2008, pp. 47–56.

[7] P. Naghshtabrizi and J. P. Hespanha, "Analysis of distributed control systems with shared communication and computation resources," in *Proceedings of the 2009 American Control Conferance (ACC)*, 2009.

[8] A. Aminifar, S. Samii, P. Eles, Z. Peng, and A. Cervin, "Designing high-quality embedded control systems with guaranteed stability," in *Proceedings of the* 33^{th} *IEEE Real-Time Systems Symposium*, 2012, pp. 283–292.

[9] A. Aminifar, E. Bini, P. Eles, and Z. Peng, "Designing bandwidth-efficient stabilizing control servers," in *Proceedings of the* 34^{th} *IEEE Real-Time Systems Symposium*, 2013, pp. 298–307.

[10] K. J. Åström and B. Bernhardsson, "Comparison of periodic and event-based sampling for first-order stochastic systems," in *Preprints of the* 14^{th} *World Congress of IFAC*, 1999.

[11] K. E. Årzén, "A simple event-based pid controller," in *Preprints of the* 14^{th} *World Congress of IFAC*, 1999.

[12] M. Velasco, J. M. Fuertes, and P. Marti, "The self-triggered task model for real-time control systems," in *the* 24^{th} *IEEE Real-Time Systems Symposium*, 2003, pp. 67–70.

[13] T. Henningsson, E. Johannesson, and A. Cervin, "Sporadic event-based control of first-order linear stochastic systems," *Automatica*, vol. 44, no. 11, pp. 2890–2895, 2008.

[14] W. P. M. H. Heemels, J. H. Sandee, and P. P. J. V. D. Bosch, "Analysis of event-driven controllers for linear systems," *International Journal of Control*, vol. 81, no. 4, pp. 571–590, 2008.

[15] M. Mazo, A. Anta, and P. Tabuada, "On self-triggered control for linear systems: Guarantees and complexity," in *2009 European Control Conference (ECC)*, 2009, pp. 3767–3772.

[16] X. Wang and M. Lemmon, "Self-triggered feedback control systems with finite-gain stability," *IEEE Transactions on Automatic Control*, vol. 54, no. 3, pp. 452–467, 2009.

[17] M. Velasco, P. Marti, and E. Bini, "Optimal-sampling-inspired self-triggered control," in *the IEEE International Conference on Event-based Control, Communications & Signal Processing*, 2015.

[18] ——, "Control-driven tasks: Modeling and analysis," in *Real-Time Systems Symposium, 2008*, 2008, pp. 280–290.

[19] M. Lemmon, T. Chantem, X. S. Hu, and M. Zyskowski, "On self-triggered full-information h-infinity controllers," in *Proceedings of the 10th International Conference on Hybrid Systems: Computation and Control*, 2007, pp. 371–384.

[20] A. Anta and P. Tabuada, "On the benefits of relaxing the periodicity assumption for networked control systems over CAN," in *Proceedings of the* 30^{th} *IEEE Real-Time Systems Symposium*, 2009, pp. 3–12.

[21] D. Antunes and W. Heemels, "Rollout event-triggered control: Beyond periodic control performance," *IEEE Transactions on Automatic Control*, vol. 59, no. 12, pp. 3296–3311, Dec 2014.

[22] D. Borgers and W. Heemels, "Event-separation properties of event-triggered control systems," *IEEE Transactions on Automatic Control*, vol. 59, no. 10, pp. 2644–2656, 2014.

[23] M. Donkers, P. Tabuada, and W. Heemels, "Minimum attention control for linear systems," *Discrete Event Dynamic Systems*, vol. 24, no. 2, pp. 199–218, 2014.

[24] K. J. Åström and B. Wittenmark, *Computer-Controlled Systems*, 3rd ed. Prentice Hall, 1997.

[25] M. Joseph and P. Pandya, "Finding response times in a real-time system," *The Computer Journal*, vol. 29, no. 5, pp. 390–395, 1986.

[26] J. Lehoczky, "Fixed priority scheduling of periodic task sets with arbitrary deadlines," in *Proceedings of the* 11^{th} *IEEE Real-Time Systems Symposium*, 1990, pp. 201–209.

[27] C. Sloth and R. Wisniewski, "Complete abstractions of dynamical systems by timed automata," *Nonlinear Analysis: Hybrid Systems*, vol. 7, no. 1, pp. 80–100, 2013.

[28] E. Aydin Gol, X. Ding, M. Lazar, and C. Belta, "Finite bisimulations for switched linear systems," *IEEE Transactions on Automatic Control*, vol. 59, no. 12, pp. 3122–3134, 2014.

[29] S. Boyd and L. Vandenberghe, *Convex Optimization*. New York, NY, USA: Cambridge University Press, 2004.

[30] M. Stigge, N. Guan, and W. Yi, "Refinement-based exact response-time analysis," in *the* 26^{th} *Euromicro Conference on Real-Time Systems (ECRTS)*, 2014, pp. 143–152.