Sample-Drop Firmness Analysis of TDMA-Scheduled Control Applications

Amir R. B. Behrouzian*, D. Goswami*, M. Geilen*, M. Hendriks†, H. Alizadeh Ara*, E. P. van Horssen*, W. P. M. H. Heemels* and T. Basten*†
*Eindhoven University of Technology, Eindhoven, The Netherlands
†TNO Embedded Systems Innovation, Eindhoven, The Netherlands

***Abstract*—This paper proposes methods for verification of (m, k)-firmness properties of control applications running on a shared TDMA-scheduled processor. We particularly consider dropped samples arising from processor sharing. Based on the available processor budget for any sample that is ready for execution, the Finite-Point (FP) method is proposed for quantification of the maximum number of dropped samples. The FP method is further generalized using a timed automata based model to consider the variation in the period of samples. The *UPPAAL* tool is used to validate and verify the timed automata based model. The FP method gives an exact bound on the number of dropped samples, whereas the timed-automata analysis provides a conservative bound. The methods are evaluated considering a realistic case study. Scalability analysis of the methods shows acceptable verification times for different sets of parameters.**

I. Introduction

Many application domains, including healthcare and automotive, require to run several applications simultaneously. Sharing resources among applications is a widely used trend towards a cost-efficient product development. This imposes new challenges in hardware and software design. Application interference, for example, on a shared resource is a potential issue. A *Budget scheduler* provides temporal predictability on a shared resource by guaranteeing a fixed access time for every scheduled application [1]. Time Division Multiple Access (TDMA) is a common scheduling policy for realizing temporal predictability for such applications [2] [3]. It allocates identical constant time slots to applications in a work cycle.

Due to the safety-critical nature of control applications, timing plays a key role in guaranteeing their Quality of Control (QoC) [2]. Running control applications on a shared processor for computation reasons can cause control samples to miss the computational deadline. This affects the QoC. A sample should be processed before the next sample arrives and therefore, each sample has a computational deadline equal to the sampling period of the application. The samples with missed deadlines are referred to as Dropped Samples (DSs). A potential reason for a sample to miss a deadline can be that sufficient resources are not available for the application when the sample is ready for processing.

Under the (m, k)-firmness condition [4] a control application can still satisfy QoC requirements in presence of DSs. That is, at least m samples out of k consecutive samples must meet the computational deadline to satisfy application level requirements. In other words, $k - m$ samples out of k consecutive samples can miss the computational deadline without violating the requirements.

The possibility of missing computation deadlines of a given task running on a TDMA-scheduled processor is traditionally verified by calculation of the best-case and worst-case response time of the task [5]. That is, if the sampling period is larger than the worst-case response time of the control application no sample misses the deadline. On the other hand, if the sampling period is less than the best case response time of the control application, all the deadlines are missed.

We consider control applications running on a shared processor under a TDMA policy. We are particularly interested in a range of sampling periods that lies between the best and the worst case response time of the control task. For such a range of sampling periods, a certain (m, k)-firmness condition is given for each control application. We aim to formally verify the satisfaction of such conditions and in effect, guarantee QoC. We propose an analytic method to quantify the number of DSs. We propose a method to obtain the maximum number of DSs by verifying the number of DSs for a finite window of arrival of samples. We further extend the method by considering parameter variation, e.g. jitter in the sampling period. A timed automata based method is presented to address this aspect. Because of an over-approximation in building the state space, the results of the timed automata method are conservative though.

The paper is organized as follows. Related work is discussed in Section II. The setup for the applications and platforms under consideration is explained in Section III. The problem of the DS quantification is formally defined in Section IV. The analytic approach for quantification of DSs is proposed in Section V. A timed automata based extension of the problem for addressing deviations in the sampling period is illustrated in Section VI. The results obtained from a realistic case-study are reported and discussed in Section VII. Section VIII makes concluding remarks.

II. Related Work

A control application that has DSs has been analysed from two points of view in previous work: (i) Study the effects of DSs on the performance of control applications and improvement of controller design to tolerate the DSs. [6] investigates the effects of DSs on the control loops of networked control systems. [7] analyses the stability and the optimality of a system governed by the (m, k)-firmness condition. [8] and [9] assign a scheduling architecture for a set of control applications to meet their requirements in presence of DSs. (ii) Quantification of a bound for DSs for a given set of applications and platform parameters. [10] addresses the DSs arising from the delay of the communication network between sensors and processor. The work investigates bounds on the number of deadline misses such that the control application retains its stability and satisfies the QoC requirements. The focus of [10] is mainly on an automotive specific communication network as a platform and hard to generalize for typical multi-processor settings. [11] provides a general formulation in the context of systems where some tasks occasionally experience sporadic overload for obtaining a tight bound of (m, k)-firmness properties. The need for overload models makes results in [11] not directly applicable to periodic control tasks. Our work belongs to category (ii). [10] and [11] focus on the dropped samples arising from sensor to actuator delay while our work address the deadline missed samples arising from the response time of the processing unit. This paper focuses on (m, k)-firmness analysis of a given TDMA-scheduled platform. Our results on (m, k)-firmness bounds can be taken into account in the controller design phase (i.e., category (i)) and vice versa.

III. Setup Under Consideration

A. Control applications

A typical control application regulates a continuous time linear time-invariant plant by periodically updating a feedback control action. Often such a scheme is implemented with a zero-order-hold static state feedback control action, possibly with sensor-to-actuator delay, as commonly done in sampled-data control [12]. The discrete-time behaviour of the state $x_t \in \mathbb{R}^{n_x}$ of such a system at sampling instance $t \in \mathbb{N}$ can be modeled by a linear difference equation of the form

$$x_{t+1} = Ax_t + Bu_t = (A - BK)x_t, \ t \in \mathbb{N}, \tag{1}$$

where matrices A and B describe the properties of the plant and $u_t \in \mathbb{R}^{n_u}$ is the control input of the system, defined as $u_t = -Kx_t$. Here $t_{k+1} - t_k = h$, where h is sampling period. After each sample of the state x_t is read, a new control input u_t is computed. In this work, we consider that gain K is appropriately designed, but the computation of u_t can be incomplete in case of a DS. Then, u_t is taken to be zero. Thus, the system exhibits two types of dynamics, as in a standard switched system,

$$x_{t+1} = A_\sigma x_t, \ \sigma \in \{c, o\}, \quad t \in \mathbb{N}, \tag{2}$$

where $A_c = A - BK$ (when a sample meets the deadline) and $A_o = A$ (for a DS) describe the so-called *closed loop* and *open loop* dynamics, respectively.

Stability of such a system can be assessed by, e.g., Lyapunov-based techniques. That is, if we can find mode-dependent $\{P_\sigma \succ 0\}, \{\sigma \in \{o, c\}\}$ and positive scalars $\rho_c, \gamma_{co}, \rho_o, \gamma_{oc}$ for which

$$A_\sigma^\intercal P_\sigma A_\sigma \preceq \rho_\sigma P_\sigma, \qquad \sigma \in \{c, o\}, \tag{3}$$

$$P_\sigma \preceq \gamma_{\sigma\tau} P_\tau, \qquad \sigma, \tau \in \{c, o\}, \tag{4}$$

hold, then $V(x) = x^\intercal P_c x$ is a k-step Lyapunov function for the system guaranteeing stability, if in any k subsequent samples at least m are not DSs, and the pair (m, k) satisfies the bound

$$\frac{m}{k - m} \geq \frac{\log(\frac{1}{\gamma_{co}\gamma_{oc}\rho_o})}{\log(\rho_c)}. \tag{5}$$

Thus any implementation that satisfies this bound achieves stability for the tuple (m, k, K). Based on the stability analysis, an (m, k)-firmness requirement can be derived, stating that at most $k - m$ out of k samples can be missed without violating the control requirements.

B. Platform architecture

We consider a control application with a given (m, k)-firmness requirement derived as per the above analysis. The execution of the corresponding control task is referred to as *samples*. The samples are released by dedicated hardware, e.g. a dedicated (plant) Electronic Control Unit (ECU) in automotive systems [10]. Such a hardware is responsible for operating the sensors and the actuators in a typical application scenario. The sensors send the samples to a main processor unit, e.g. a controller ECU [10], through a communication network. The time instance a sample is ready to be executed in the processing unit is referred to as *arrival time* of the sample. We consider the processing unit to run under a TDMA policy. In TDMA policy, a work cycle named *time wheel* consists of multiple *time slots* – allocated to applications. This provides a fixed periodic access time for each application. Once a sample arrives on the processor, it is processed only if the current time slot is allocated to the application that the sample belongs to. Otherwise the sample has to wait until the time in which the processor is allocated to the application. Each sample needs to be executed on the processor for a certain amount of time to generate the corresponding actuating data. This period is referred to as *execution time* of a sample. If the execution of a sample cannot be completed before the next one arrives for a given slot assignment, we encounter a DS.

Based on the fact that the sampling clock (on the dedicated hardware) is usually asynchronous with respect to the processor clock, the arrival time of a sample is non-deterministic. Depending on the arrival time of a sample, the pattern and the number of DSs can therefore vary. Then, for a given set of application and platform settings, the challenge is to obtain the absolute maximum number of DSs to verify (m, k)-firmness conditions. While the relative arrival time of a sample is non-deterministic, the relative arrival time of the successive samples can be expressed in terms of the arrival time of the first sample (in a window of k consecutive samples). In Section V we use this fact to propose a method for maximum DSs quantification.

IV. Problem Definition

Applications running under a TDMA policy can be analyzed independently. From an application point of view, a TDMA time wheel can therefore be classified into two types of intervals: *allocated intervals* and *non-allocated intervals*. An allocated interval refers to an interval in which the processor is assigned to the application. A non-allocated interval includes the intervals that are allocated to other applications or not used for processing purposes (e.g. context switching overhead [13]).

Definition 1 (TDMA Schedule)**.** *A TDMA schedule is a tuple* $p = (w, A)$ *consisting of* $w \in \mathbb{R}^+$*, the TDMA time wheel size, and a finite set* $A = \{(t_{start,i}, t_{end,i}) \in \mathbb{R}^+ \times \mathbb{R}^+ | 0 \leq t_{start,i} < t_{end,i} < w\}$ *of allocated intervals.* $t_{start,i}$ *and* $t_{end,i}$ *denote the start and end time of the allocated interval* i *(*$i \in \mathbb{N}$*).*

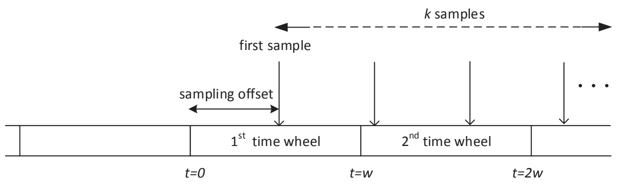

Fig. 1. Relative position of the first time wheel and the first control sample in a repetitive execution of a TDMA-scheduled processor assigned to a control application

Definition 2 (Control Application)**.** *A control application is a tuple $ca = (e, h)$ consisting of execution time e of the control task and sampling period h.*

Since from a platform's point of view the arrival times of k consecutive samples are not predictable (see Section III), satisfaction of (m, k)-firmness properties must be verified by comparing $k-m$ with the maximum potential number of DSs.

Problem definition *For a given control application $ca = (e, h)$ running on a processor with a TDMA schedule $p = (w, A)$ with a given (m, k)-firmness condition, find the maximum number of DSs in a window of k consecutive samples.*

V. DS Quantification: Finite-Point (FP) Method

A. Background

We consider a situation in which a control application $ca = (e, h)$ is running on a processor with a TDMA schedule $p = (w, A)$. This application can satisfy performance constraints under a certain (m, k)-firmness condition. We investigate the operation of a processor in a time interval which includes the arrival times of k consecutive samples. Therefore, we define a relative time based on a specific time wheel in which the first sample out of k consecutive samples arrives.

Definition 3 (First Time Wheel)**.** *Considering k consecutive control samples processed on a TDMA-scheduled processor, we refer to the time wheel at which the first sample arrives as the first time wheel.*

We consider the start time of the first time wheel as time $t = 0$. We refer to the arrival time of the first sample as the *sampling offset*. Fig. 1 illustrates first sample, sampling offset and the first time wheel in several repetitive executions of a TDMA time wheel. In this way, the relative arrival times of all samples can be determined, since the sample arrivals are separated by the sampling period h.

The computational deadline for any arriving sample is the arrival time of the next sample, which is equal to the sampling period h. The available resource for the control application is assessed in the time interval between t and $t+h$. We represent this by the *Resource Availability Function* (RAF) $g : \mathbb{R}^+ \to \mathbb{R}^+$ such that

$$g(t) = \int_t^{t+h} f(\tau)d\tau, \tag{6}$$

where $f : \mathbb{R}^+ \to \{0, 1\}$ is Allocated-Time Function (ATF) which takes the value 1 if the processor is allocated to the application at time t and 0 otherwise. That is,

$$f(t) = \begin{cases} 1 & \text{if } nw + t_{start,i} \le t \le nw + t_{end,i} \\ & \text{for some } i \in \mathbb{N}, n \in \mathbb{Z} \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

In other words, $g(t)$ is the sliding integral of $f(t)$ over t to $t + h$. Since the TDMA schedule $p = (w, A)$ repeats every w time units, we know that ATF is periodic with the same period w, i.e., $f(t + w) = f(t)$. Therefore RAF $g(t)$ is also periodic with w. Fig. 2(a) illustrates ATF of a control application wit $ca = (270\mu s, 700\mu s)$ running under a TDMA schedule $p = (550\mu s, A)$ with $A = \{(110\mu s, 210\mu s), (330\mu s, 430\mu s)\}$. RAF for this example is shown in Fig. 2(b).

B. Miss-Zone

From Section III it is inferred that a sample, arriving at time t, misses the computational deadline if $e > g(t)$ where e is the execution time of the application. The horizontal line on Fig. 2(b) shows the execution time of the application. Then a TDMA time wheel can be split into two types of intervals:

1) *miss zone* intervals in which $e > g(t)$
2) *hit zone* intervals in which $e \le g(t)$

These two intervals are characterized by Miss Zone Function (MZF), $z : \mathbb{R}^+ \to \{0, 1\}$ such that

$$z(t) = \begin{cases} 1 & \text{if } e > g(t) \\ 0 & \text{if } e \le g(t) \end{cases} \tag{8}$$

From (8) and the fact that RAF, $g(t)$, is periodic, we conclude that MZF is also periodic with period w, i.e., $z(t+w) = z(t)$.

Definition 4 (Start Times of Miss Zones)**.** *For a given MZF, $z(t)$, the set $T_m = \{t \in \mathbb{R}^+ \mid z(t + \epsilon) > z(t)\}$ contains the start times of the miss zones. ϵ denotes an infinitesimal time duration. The set $T_{mf} = T_m \cap [0, w)$ denotes the start times of miss zones in the first period.*

Note that the miss zones are left-open intervals and T_m includes their lower bounds. Fig. 2(c) shows both the miss and hit zones in the first time wheel for the example on Fig. 2(a). For this example $T_{mf} = \{140\mu s, 360\mu s\}$. The problem we address is to determine the maximum possible number of samples out of k consecutive samples that may arrive in miss zones. In the rest of this section we propose methods to address this problem.

We consider a function that represents each control sample by a Dirac delta function $\delta(t)$. We call this function Control Sample Distribution Function (CSDF) $r : \mathbb{R}^+ \to \mathbb{R}^+$ such that

$$r(t) = \sum_{n=0}^{k-1} \delta(t - nh), \tag{9}$$

where k is the number of the consecutive samples. Consider the sampling property of the δ-function, which says that for any function ψ

$$\int \delta(t - \tau)\psi(t)dt = \psi(\tau) \tag{10}$$

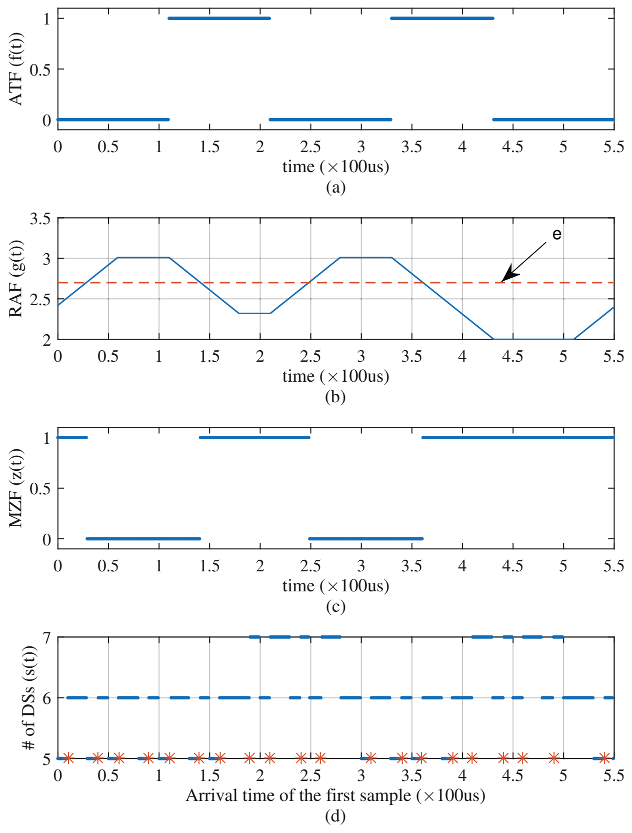

Fig. 2. A TDMA time wheel with two slices allocated to a control application.

In view of Equation (9), the number of DSs is represented by function $s : \mathbb{R}^+ \rightarrow \mathbb{N}$ such that

$$s(t) = \int_0^\infty z(\alpha)r(\alpha - t)d\alpha = \sum_{n=0}^{k-1} z(t + nh), \qquad (11)$$

where $z(t)$ is MZF, t is the sampling offset and k is the number of consecutive samples. $s(t)$ yields the number of DSs for a sampling offset of t in k consecutive samples. Fig. 2(d) shows the number of DSs for $k = 10$ and sampling period of $h = 700\mu s$ against the sampling offset t for the duration of one time wheel, i.e., between $t = 0$ and $t = 550\mu s$.

From the fact that $z(t)$ takes either value 0 or 1 and $s(t)$ is a summation of $z(t)$, we conclude that $s(t)$ only takes integer values. Moreover, from (11) and the fact that MZF is a periodic function with a period of time wheel size w, we conclude that $s(t)$ is also periodic with the same period. The absolute maximum number of DSs is therefore obtained by verification of one period of $s(t)$. That is

$$s_{max} = \max_{0 \le t < w} s(t). \qquad (12)$$

Let us take $z(t)$ as shown on Fig. 2(c). $z(t)$ is a continuous function and we need a solution for Equation (11), without trying all values t. For this example $s_{max} = 7$ as shown in the figure. However, verification of s_{max} from discrete samples of $s(t)$ may not provide a guarantee to obtain the absolute maximum value, and it can be computationally inefficient for small discretization steps. This problem is addressed by the following theorem that shows that verification of a finite number of selected points can guarantee to obtain s_{max}.

Theorem 1. *Any increase in the value of $s(t)$ happens when at least one of the k samples enters a miss-zone. That is if $s(t + \epsilon) > s(t)$ then $t + nh \in T_m$ for some $0 \le n < k$.*

Proof. For any time t, the inequality $s(t + \epsilon) > s(t)$ implies, considering Eq.11, that there is at least one $n \in \{0, 1, 2, ..., k-1\}$ such that $z(t+nh+\epsilon) > z(t+nh)$. This means, by Def. 4, that $t + nh \in T_m$. □

From the above theorem, we conclude that if there is any increase in the value of $s(t)$ then it happens for a value of t such that $t + nh$ has just entered a miss zone. Because $s(t)$ is periodic, and there are a finite number of samples, we only need to evaluate a finite number of points t.

These points are in the set T_{inc} of points t_{inc} the arrival time of the first sample, such that at least one of the samples arrives at the start time of a miss-zone. T_{inc} is obtained as follows. It includes for each $0 \le n < k$ and each (of the finite) $t \in T_{mf}$:

$$t_{inc} = \begin{cases} t - mod(n \times h, w) & t \ge mod(n \times h, w) \\ w + t - mod(n \times h, w) & t < mod(n \times h, w) \end{cases} \qquad (13)$$

where

$$mod(x, y) = y - x \cdot \left\lfloor \frac{y}{x} \right\rfloor$$

It can then be concluded that to obtain s_{max} it is sufficient to verify $s(t+\epsilon)$ for all t in (finite) T_{inc}. From (13) it follows that $|T_{inc}| = |T_{mf}| \times k$, where $|T_{mf}|$ denotes the number of the elements in T_{mf}. In the example shown in Fig. 2, $|T_{inc}| = 2 \times 10 = 20$. The elements of T_{inc} are shown by stars in Fig. 2(d). In this figure, essence of Theorem 1 is observed when any rise in the value of $s(t)$ coincides with a star.

C. Conservativeness of the bound

Due to some platform related aspects (e.g. different operational mode of the processor) execution time of the control task might not be constant value. Variation in the execution time is directly translated to a change in the level of the dashed line on Fig. 2(b). Based on the following theorem variation in the execution time has a monotonic effect on the number of DSs.

Theorem 2. *For two given applications ca_1 and ca_2 with same RAF, $g(t)$, and CSDF, $r(t)$, the inequality $e_1 < e_2$, admits that $s_{max_{ca_1}} \le s_{max_{ca_2}}$, where $s_{max_{ca_1}}$ and $s_{max_{ca_2}}$ denotes the maximum number of DSs of applications ca_1 and ca_2 respectively.*

Proof. From the fact that ATF, $f(t)$, is a combination of scaled and shifted rectangular functions, the result of the integral in (6) then yields either fixed values or ramp with a slope of 1 or -1 (see Fig. 2(b)). Any change in the value of e therefore scales the miss zone intervals equally from both sides. A bigger value of e leads to a bigger miss zone. Then, from the fact that the miss zone intervals of ca_1 is a subset of the miss zone interval of ca_2, any DS of ca_1 is also DS in ca_2. Then for any t, $s_{ca_1}(t) \leq s_{ca_2}(t)$ which concludes $s_{max_{ca_1}} \leq s_{max_{ca_2}}$. □

Considering the upper bound for execution time therefore leads to a conservative result. That is, for a control application that has different operation mode with different execution time, the maximum possible execution time should be taken into account to guarantee the maximum number of DSs. However, it should be noted that the FP method does not apply for all the cases that there is a variation in parameters, e.g., jitter in arrival time of the samples. In such a case all the possibilities for arrival time of samples should be considered to verify the (m, k)-firmness properties. In Section VI, we introduce a method based on timed automata that covers such variation in parameters and can be considered as a generalization of the FP method.

VI. DS Quantification: Timed Automata Method

DSs quantification for a case with a variation in the arrival times of samples requires an exhaustive search over infinite number of potential values for the parameter to obtain the absolute maximum number of DSs. In such a case, the FP method is no longer applicable due to the fact that relative arrival time of a sample for a certain sampling offset can take infinite number of values. In this section we use a timed automata-based model as a general method to address variation in the value of such parameters.

Timed automata are finite state automata extended with real-valued clocks [14]. This formalism provides a dense time domain and allows non-deterministic choices with respect to both discrete transitions and the progress of time. There are several tools that support exhaustive analysis of timed automata with respect to temporal logic properties. In this paper, we use the Uppaal tool which is a widely used model checking tool used by both industry and academia [15]. Uppaal supports several extensions that increase the usability, such as a C-like language to manipulate discrete variables. Furthermore, it also supports generalizations of timed automata, i.e., stopwatch automata and linear hybrid automata. These formalisms are more expressive than timed-automata, but the types of analysis that can be done is more restricted: stopwatch automata [16] only allows an analysis based on an over-approximation of the state space, and linear hybrid automata allows analysis based on statistics and simulation techniques. [16] analyzes and reasons that the prize of over-approximation in not too high though.

The fact that ATF is periodic (see Section IV) allows us to model it as a timed automata. Let us take the example shown on Fig. 2. The TDMA time wheel is divided up into five time slots–two slots allocated to the control application, as shown on Fig. 2(a). By taking the periodicity into account and by choosing a different offset (which does not change the problem), we see that the time domain is composed of consecutive intervals $[0, 100], (100, 220), [220, 320], (320, 550) \ldots$ of allocated intervals (closed intervals) and non-allocated intervals (open intervals).

Fig.3 shows the Uppaal encoding of the sequence of allocated and non-allocated intervals. We refer to this automaton as TDMA automaton. It has a local clock variable x to keep

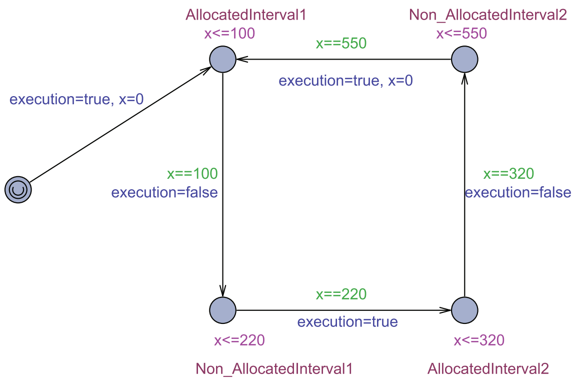

Fig. 3. TDMA automaton for modeling a TDMA time wheel.

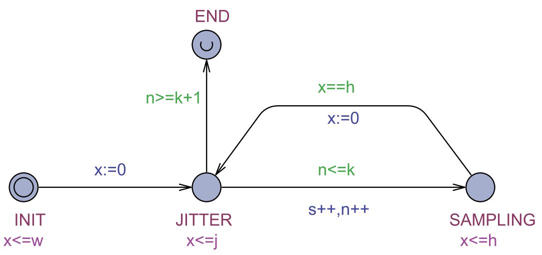

Fig. 4. Sample provider automaton to keep the track of generated samples.

track of the time. Starting from an initial location, *AllocatedInterval1*, this automaton changes the location whenever x equals to the boundaries above and updates a boolean variable $execution$ which indicates if the processor is allocated to the application in the next location. The automaton goes to the initial state once $x = w$ where w is the size of time wheel. The invariant $x \leq 100$ in combination with the guard $x == 100$ on the outgoing transition to *Non-AllocatedInterval1* ensures that the automaton can stay in *AllocatedInterval1* for exactly 100 time units before it is forced to go to *Non-AllocatedInterval1*. The other locations have the same behaviour. Note that this automaton does not exactly model the ATF, because at the edges of the intervals, the automaton can be in both current and next locations, whereas the function defines that the automaton should either be in location *Non_AllocatedInterval1* or *AllocatedInterval2*. The behavior of the function is, however, included in the behavior of the automaton, which therefore provides a *conservative abstraction*. Besides using stopwatch (as discussed earlier in Section VI), this is the second reason of over-approximation of the timed automata method.

Fig. 4 shows an automaton which keeps track of sample generation time considering jitter bound for sampling period h. This automaton is referred to as *sample provider*. It has a clock variable x, which specifies the generation time of the next sample according to a sample rate. Furthermore, it has three constant integer parameters: h, k and j. The parameter h is equal to the sampling period ($h = 700\mu s$ in the example shown on Fig. 2) and used to specify the time between generated samples. The parameter k is used to specify the number of consecutive samples that is generated. j determines the jitter bound if there is any, otherwise it takes zero. This automaton starts from the initial location *INIT*. The transition from *INIT* to *JITTER* is non-deterministic i.e. it

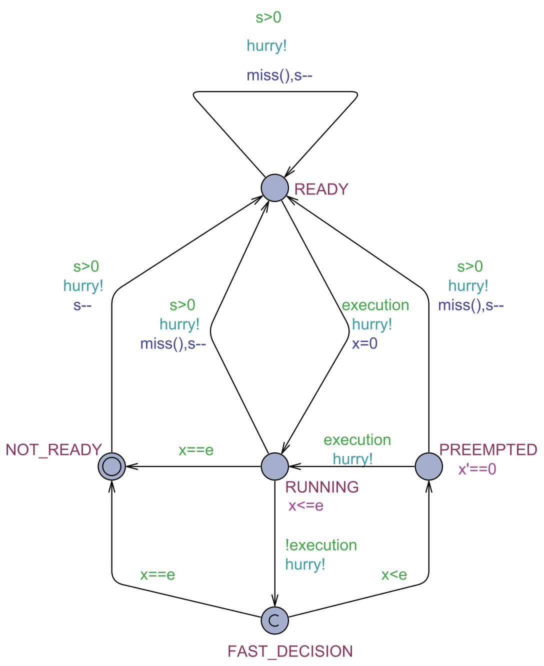

Fig. 5. CONTROL APPLICATION automaton for counting DSs.

can be taken at any value of x less than w. This captures every possible offset of sampling less than w of the sampling with respect to the TDMA time wheel. The automaton can leave the location *JITTER* at any time less than j. This allows us to model jitter [17]. This non-determinism again causes the tool to consider all possible amounts for jitter during the maximum DSs quantification. Once the automaton takes a transition towards *SAMPLING*, the value of the parameter s is incremented by one. Then it takes the transition to *SAMPLING* if we have not finished with the k samples, increments the sample counter n, and resets the clock. The automaton stays in the location *SAMPLING* until the clock equals the sampling period and then resets the clock and takes the transition back to *JITTER*. Once all the k samples have been generated, the transition to the location *END* is taken. This is an *urgent* location which must be left immediately, otherwise it causes a deadlock. This feature is used to stop the verification process in this time-automata model.

Based on the value of the parameters s and $execution$ coming from automata *SampleProvider* and *TDMA*, a third automaton, named *Control application*, decides if a generated sample has been executed properly or not. As shown on Fig. 5, this automaton is initially located on *NOT_READY*. Once a sample is generated, i.e. the value of s equals 1, the automaton takes the transition to the location *READY* and decreases the value of s by one. This shows that the generated sample is ready to be executed. Note that this transition like any other transition that has a synchronization *hurry* should be taken immediately once the guard of the transition ($s > 0$ in this case) is satisfied. The automaton stays in *READY* location until one of the following transitions are enabled: 1) The self loop is enabled if a new samples is generated. That is, the previous sample is overwritten while it had not been started to be executed. This transition therefore increases the number of the missed samples by one. 2) The transition to the location *RUNNING* is enabled if $execution$ takes true, which shows that the automaton has gone to one of the locations *AllocatedInteval1* or *AllocatedInteval2*. This transition to the location *RUNNING* also resets the clock x. In this automaton, the clock keeps track of the total time that *execution* has the value *true*, which shows that the processor is allocated to the application to execute the last sample. If *execution* takes false, i.e. the control application is preempted in real world, the automaton *ControlApplication* takes the transition to the location *fast decision*, which should be left immediately. If $x >= e$, which shows that the execution of the previous sample has been completed, the automaton takes the transition to the initial location, otherwise it goes to the location *PREEMPTED* and stops the clock. As discussed in Section VI, using stop watch causes an over-approximation of the state space which leads to have conservative results. The automaton stays in this location until either the variable $execution$ is changed to true or a new sample is generated. The maximum number of DSs can be found using the UPPAAL query `sup : m`, which exhaustively analyzes the state space for the supremum value of variable *m*. The variable `m` indicates the number of DSs that is increased by 1 whenever the function `miss()` is run. For $k = 10$, for instance, UPPAAL reports maximum 7 DSs. Due to the over-approximation of this method we conclude that real number of DSs for this case is equal or less than 7. The verification takes $46ms$. As more experiment with the same example, for $k = 50$ and $k = 100$ UPPAAL reports 33 and 64 DSs with verification times of $760ms$ and $2s$, respectively.

In the example shown on Fig. 4, assume that there is a jitter of $20\mu s$ in the sampling period for $h = 700\mu s$ (i.e. $680\mu s \le h \le 720\mu s$). All potential values for sampling period should be considered for verification of the maximum number of DSs. This situation can be specified in the automaton *SampleProvider* by defining the jitter parameter $j = 40$. That is, the location *JITTER* can be left once $x \ge 0$ and must be left before $x = 40$. Applying this change in our case, it yields 9 DSs.

VII. EVALUATION

In this section, we first explain our experimental results of applying the proposed methods on a realistic case. Next, we make some hypotheses about the scalability of the methods. Finally, by further experiments, we validate the hypotheses.

A. Case-study

For illustration of the applicability of our proposed methods, we consider a control application with sampling period of $2ms$. We consider a window of $k = 125$ consecutive samples in our experiments. Based on the specifications of the platform on which the application is running, execution time of the control task is determined. We took different sets of platform-related settings to verify the (m, k)-firmness properties of each. The FP method was implemented in MATLAB and compiled on a computer with a quad-core processor and a clock frequency of $2.6GHz$. The same system was used to verify the UPPAAL model.

Table I shows the settings and results of our experiments. In this table w indicates the size of the TDMA time wheel, e denotes the execution time of the control application, and t_{ver} shows the verification time. The last column of the table

TABLE I
DIFFERENT SETS OF PLATFORM SETTINGS AND VERIFICATION RESULTS USING FINITE-POINT AND TIMED-AUTOMATA METHODS ($k = 125$)

w	allocated interval (μs)	e	t_{ver} (FP method)	t_{ver} (Timed-Automata method)	max # of DSs
$1.3ms$	[0,80], [440,520], [870,950]	$400\mu s$	$225\mu s$	6.6s	58
$1.3ms$	[0,175], [870,1000]	$400\mu s$	$327\mu s$	3.6s	10
$700\mu s$	[0,250]	$600\mu s$	$332\mu s$	2.6s	125
$700\mu s$	[0,250]	$500\mu s$	$352\mu s$	2.1s	54

shows the maximum number of DSs for each case. No jitter in sampling period is considered in the experiments reported in the table. It is noteworthy that despite the over-approximation in the timed automata method, the results obtained by both methods proposed are the same for this example. However, as it can be seen in Table I, the verification times are quite different. The verification time of the UPPAAL model is at least one order of magnitude larger than that of the FP method. This justifies the use of the FP method for cases without jitter. Our experiments show that this difference in the verification time between the two methods increases as k increases. However, this verification time is still acceptable as a part of the design process. It is noteworthy that the timed automata model can be simplified for a control application without jitter in the sampling period. Then the verification time would be shortened, though still longer than the verification time of the FP method. However in this work we introduce the timed automata model to address the cases with jitter.

Fig. 6 depicts the maximum number of the DSs against the sampling period for the set of settings in the first row of Table I. From classical response time analysis it can be obtained that the worse-case and best-case response time for this example are $2.135ms$ and $1.77ms$. That is, a sampling period shorter than $1.77ms$ will result in all DSs while a sampling period longer than $2.135ms$ is enough to meet all the deadlines. The maximum number of DSs for sampling periods between the values above can be obtained using the method proposed in this work. The range of sampling periods between the above values gives different number of DSs as shown on Fig. 6. For a given platform settings, this analysis can be used to choose a suitable sampling period to meet an (m, k)-firmness bound (hence, to meet the QoC requirement). That is, considering (m, k)-firmness properties, we can reduce the sampling period to a value less than $2.135ms$ without allocating more processor resource to the application. Besides, we can reduce the allocated resource instead of changing the sampling period to have a resource efficient allocation. In the first set of settings in Table I, for example, considering a sampling period of $2.135ms$ which results in no DSs, we can reduce 11% of the length of each allocated slice, i.e. 33% less resources allocated, to have 25 DSs.

B. Complexity analysis

Next, we discuss the scalability of the proposed methods. The FP method applies (11) and (13) on $| T_{inc} |$ points to verify the absolute maximum number of DSs (see Section V). Since $| T_{inc} | = k \times | T_{mf} |$, where $| T_{mf} |$ is the number

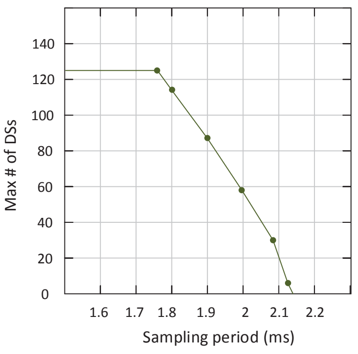

Fig. 6. Maximum number of DSs against sampling period for the case in the first row of Table I

of the miss zones in one time wheel, we expect that k and $| T_{mf} |$ affect the verification time linearly. We investigated this hypothesis by verifying different values of k and $| T_{mf} |$ for the first case shown in the Table I. Fig. 7 confirms our hypothesis by showing this linearity. The same effect was observed for $| T_{mf} |$. We believe that the verification time of the FP method will be even less if it is implemented in a language like C.

The verification time of UPPAAL highly depends on the size of the state space that the tool internally builds while it is exploring possible states of an automaton. The bigger the state space, the more time it takes to verify. We can fairly abstract each state with information about the location of each automaton and value of the clock variable. Then, the size of the state space depends on the all possible numbers of different combinations of clock values and the number of the locations in each automaton. Based on the explanation above we expect that the following platform-related and application-related settings exponentially affect the verification time: 1. The number of separated allocated intervals in the automaton *TDMA* (see Section VI). Table I confirms this hypothesis by showing that for the cases with the same settings except the number of allocated intervals, the one that has more allocated intervals takes more time to verify the result. Our results show that this dependency is almost linear. 2. k, which determines how long the state space building process continues. Fig. 8 shows the exponential dependency of the verification time on k. Note that the scale of the y-axis differs from that of Fig. 7.

Another aspect that affects the verification time of a UPPAAL verification is the non-determinism in parameters. In such a case, UPPAAL has to build up the state space for all the possible values of non-deterministic parameters which takes up a larger space in memory and longer verification time. Fig. 9 depicts the verification time against the number of consecutive samples for the same settings as Fig. 8 except a sampling period with a jitter of 5% of the sampling period h. Based on our investigation, the value of k is usually less than a few hundred. However we increased k to 500 and the verification time was still in range of a few hours which is practical (see Fig. 9).

VIII. CONCLUSION

An analytical method was proposed to quantify the maximum number of dropped samples for a given control applica-

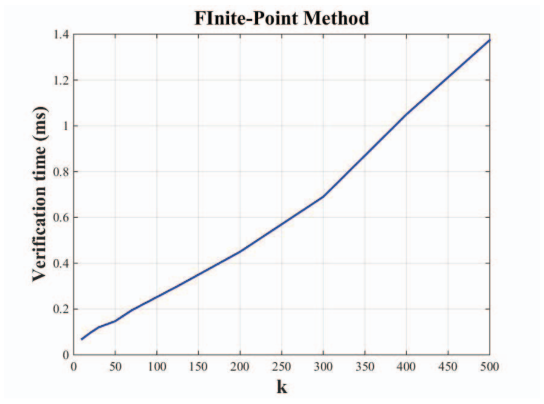

Fig. 7. Verification time of the FP method against the number of consecutive samples k

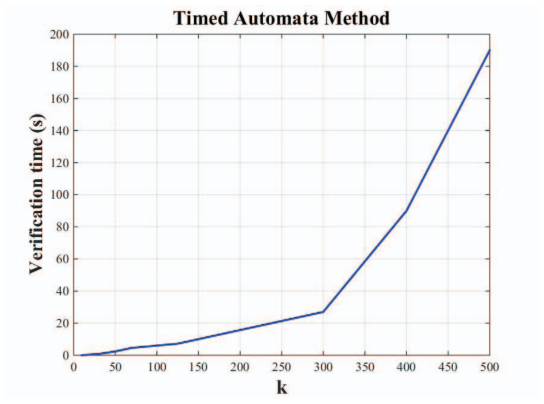

Fig. 8. Verification time of the timed automata method against the number of consecutive samples k

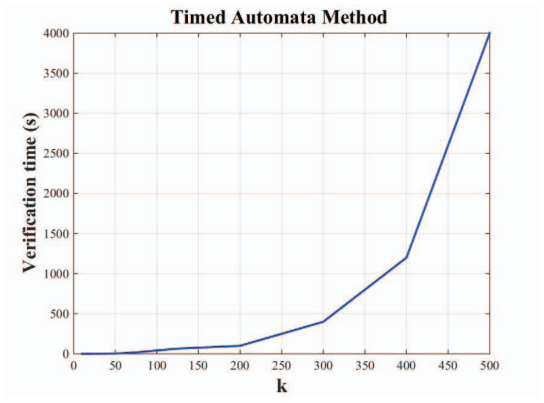

Fig. 9. Verification time of the timed automata method against the number of consecutive samples considering a jitter of 5%

tion governed by an (m, k)-firmness condition running under a TDMA–scheduled processor. We showed that the proposed FP method verifies (m, k)-firmness properties with acceptable verification time for use at design time. We extended the problem for the cases with a variation in the sampling period. To this end, a timed automata based method was used to model a TDMA time wheel and the sampling process of a control application. The timed automata based method has a longer verification time compared to the FP method for cases where both methods are applicable. However run times are still practical. The FP method provides an exact bound, whereas the timed automata method yields conservative results.

Acknowledgment

This research is supported by the ARTEMIS joint undertaking under the ALMARVI project (621439), by the Dutch Technology Foundation (STW) under grant number 12697 ”Control based on Data-Intensive Sensing,” and by the Innovational Research Incentives Scheme under the VICI grant ”Wireless control systems: A new frontier in automation” (No. 11382) awarded by NWO (The Netherlands Organisation for Scientific Research) and STW (Dutch Technology Foundation).

References

[1] G. C. Buttazzo, *Hard real-time computing systems: predictable scheduling algorithms and applications*. Springer, 2011, vol. 24.

[2] B. Akesson, A. Minaeva, P. Sucha, A. Nelson, and Z. Hanzalek, “An efficient configuration methodology for time-division multiplexed single resources,” in *Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2015, pp. 161–171.

[3] A. R. Behrouzian, D. Goswami, T. Basten, M. Geilen, and H. A. Ara, “Multi-constraint multi-processor resource allocation,” in *Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), International Conference on*. IEEE, 2015, pp. 338–346.

[4] M. Hamdaoui and P. Ramanathan, “A dynamic priority assignment technique for streams with (m, k)-firm deadlines,” *Computers, IEEE Transactions on*, vol. 44, no. 12, pp. 1443–1451, 1995.

[5] A. Schranzhofer, R. Pellizzoni, J.-J. Chen, L. Thiele, and M. Caccamo, “Worst-case response time analysis of resource access models in multi-core systems,” in *Proceedings of the 47th Design Automation Conference*. ACM, 2010, pp. 332–337.

[6] F. Felicioni, N. Jia, Y.-Q. Song, and F. Simonot-Lion, “Impact of a (m, k)-firm data dropouts policy on the quality of control,” in *6th IEEE International Workshop on Factory Communication Systems*. IEEE, 2006, pp. 353–359.

[7] N. Jia, Y.-Q. Song, and R.-Z. Lin, “Analysis of networked control system with packet drops governed by (m, k)-firm constraint,” in *6th IFAC international conference on fieldbus systems and their applications (FeT)*, 2005.

[8] N. Jia, Y.-Q. Song, and F. Simonot-Lion, “Task handler based on (m, k)-firm constraint model for managing a set of real-time controllers,” in *15th International Conference on Real-Time and Network Systems-RTNS*, 2007, pp. 183–194.

[9] F. Flavia, J. Ning, F. Simonot-Lion, and S. YeQiong, “Optimal online (m, k)-firm constraint assignment for real-time control tasks based on plant state information,” in *Emerging Technologies and Factory Automation, IEEE International Conference on*. IEEE, 2008, pp. 908–915.

[10] D. Goswami, R. Schneider, and S. Chakraborty, “Relaxing Signal Delay Constraints in Distributed Embedded Controllers,” *Control Systems Technology, IEEE Transactions on*, vol. 22, no. 6, pp. 2337–2345, 2014.

[11] W. Xu, Z. A. H. Hammadeh, A. Kröller, R. Ernst, and S. Quinton, “Improved deadline miss models for real-time systems using typical worst-case analysis,” in *27th Euromicro Conference on Real-Time Systems, ECRTS*, 2015, pp. 247–256.

[12] K. J. Åström and B. Wittenmark, *Computer-controlled systems: theory and design*. Courier Corporation, 2013.

[13] J. Valencia, D. Goswami, and K. Goossens, “Composable platform-aware embedded control systems on a multi-core architecture,” in *Digital System Design (DSD), Euromicro Conference on*. IEEE, 2015, pp. 502–509.

[14] R. Alur and D. Dill, “A theory of timed automata,” *Theoretical Computer Science*, vol. 126, no. 2, 1994.

[15] G. Behrmann, A. David, K. Larsen, J. Hakansson, P. Petterson, W. Yi, and M. Hendriks, “Uppaal 4.0,” in *Quantitative Evaluation of Systems, Third International Conference on*, 2006.

[16] F. Cassez and K. Larsen, “The impressive power of stopwatches,” in *CONCUR Concurrency Theory*. Springer, 2000, pp. 138–152.

[17] S. Perathoner, E. Wandeler, and L. Thiele, “Timed automata templates for distributed embedded system architectures,” Tech. Rep. 233, Nov 2005.