Dynamic Counters and the Efficient and Effective Online Power Management of Embedded Real-time Systems

Kai Lampka
Computer Engineering and Networks Laboratory
ETH Zurich, Switzerland
lampka@tik.ee.ethz.ch

Kai Huang
fortiss GmbH
Guerickestr. 25, 80805
Munich, Germany
khuang@fortiss.org

Jian-Jia Chen
Institute for Process Control and Robotics
Karlsruhe Institute of Technology, Germany
jian-jia.chen@kit.edu

ABSTRACT

Energy consumption has become an important issue for modern embedded systems. This is because, one does not only like to deploy high-performance systems and provide guaranteed services, but also request system deployments to last as long as possible. With online dynamic power management (DPM), one adaptively changes the system's power mode, i. e., schedules when to turn on and off on-the-fly, to achieve energy savings. This paper introduces dynamic counters and discusses their usage in the context of (online) DPM in a setting where systems have to fulfill hard real-time requirements. The proposed scheme enables one to conservatively bound the future workload and thereby to safely schedule on- and off-periods of devices. Simulation results indicate that the proposed technique is more efficient and more effective with respect to energy savings, compared to previous work.

Categories and Subject Descriptors

C.3 [**SPECIAL-PURPOSE AND APPLICATION-BASED SYSTEMS**]: Real-time and embedded systems

General Terms

Algorithms, Design

Keywords

Dynamic Counters, Real-Time Calculus, Dynamic Power Management, Real-Time Systems, Embedded Systems

1. INTRODUCTION

1.1 Motivation

For ensuring the stability, safety, and dependability of embedded systems, system designers have to provide guarantees or assurance on several non-functional properties, such as timing satisfaction, energy reduction, etc. This has triggered the researches in the area of energy-efficient scheduling for (hard) real-time embedded systems, in which the objective is to minimize the energy consumption while meeting the (hard) timing satisfaction of real-time tasks. The widely adopted techniques are as follows:

1. Dynamic frequency voltage scaling (DVFS) features a reduction in the dynamic power consumption by changing the supply voltage and frequency.

2. Dynamic power management (DPM) targets at a reduction of the static power consumption, which is realized by changing a device's power mode, e. g., sleep mode with clock or voltage gating for different power consumption characteristics, when it is idle.

To efficiently manage the power consumption of a system, online DVFS/DPM scheduling algorithms adaptively control the power modes of a device (cf. Sec. 1.2). For systems with unknown event arrivals in advance, taking advantage of the runtime information, online scheduling algorithms can generally reduce more energy consumption. However, one has to pay the price, i.e., runtime and memory overheads, for online handling due to the lack of knowing future workload. One may also note that the effectiveness of online schemes in hard real-time settings depends on the accuracy of the provided worst-case estimations, e. g., workload bounds, execution times, power consumptions, etc. Moreover, an increased variability of the estimates may significantly influence the effectiveness. However, for hard real-time systems as considered here, any violation of deadlines or buffer requirements is not acceptable. Hence, strictly considering upper bounds on the quantities can not be avoided.

For not being extremely pessimistic when deriving activation and deactivation schedules for a device online, one commonly records the streaming behavior of the environment. The obtained historic information can then be used for bounding future workload. For the performance of such *history-aware* techniques the following aspects are decisive: (a) the computational effort, i.e., runtime and memory overheads, and (b) the precision of the obtained bound. However, the above two objectives are conflicting: the amount of historical information to be stored is limited due to the memory constraint of a system. As soon as the past behavior drops out of the finite-horizon memory, the bound on future workload tends to be pessimistic. This is because deadline miss needs to be strictly avoided, and history-dependent online algorithms may therefore prefer idle over sleep modes. Moreover, the usage of historical information consumes memory and computational power. Hence the storage of more (longer) historical information for obtaining less pessimistic bounds on future event arrivals is truly limited, especially in embedded systems as their memory and computing budgets are strictly limited. Therefore, how to make use of historical information in an efficient manner and to tightly bound future workload needs more consideration.

CODES+ISSS'11, October 9–14, 2011, Taipei, Taiwan.

This paper presents an online DPM scheme based on dynamic counters. Dynamic counters feature a history-aware bounding of future workload with respect to some static upper and lower workload bounds provided to the system engineer at design time. Compared to previous work in this direction, e. g., [10], the proposed technique offers the following advantages:

- Memory efficiency can be achieved as past event arrivals do not need to be stored explicitly. Instead, the information is implicitly described by the dynamic counters, yielding a constant memory overhead.
- Computational efficiency can be achieved as the bounding of future workload is extremely simplified; previous work required complex (mathematical) operations such as the construction and (de)convolution of history-aware bounding functions.
- Energy reduction can be achieved effectively, since dynamic counters capture the stream's history back to a system's start. With previous approaches, the system explicitly records a stream's history which will be lost over a certain age, and, hence, induces pessimism into the obtained bounds.

An additional advantage of a dynamic counter-based DPM scheme arises from its simplicity which supports an easy implementation at the lowest system-levels, namely in hardware and its steering routines. This has clearly potential for making the presented scheme competitive in practice.

1.2 Related Work

Energy-efficient scheduling by applying DVFS for real-time systems has received much attention in the past decade, and has been extensively studied. Moreover, in deep sub-micron technologies, static power consumption in CMOS circuits has significant contribution of the total power consumption, in which the static power consumption is comparable to the dynamic power dissipation [13]. As a result, by considering both DPM and DVFS, Irani et al. [11], Jejurikar et al. [12, 13], and Chen and Kuo [5, 6] propose to execute tasks at certain speeds and to control the procrastination of real-time events.

When only considering DPM to change the device/system modes dynamically, Baptiste et al. [3, 4] propose polynomial-time algorithms based on dynamic programming to control when to activate and deactivate the device/system for a set of aperiodic real-time jobs with known arrival times offline. Shrivastava et al. [18] develop a framework for code transformations such that the idle time can be aggregated for energy reduction. Park et al. [16] use dynamic programming to decide where to insert instructions to put some execution units to low-power mode. By controlling shutting down and waking up system devices for energy efficiency, Swaminathan et al. [19] explore dynamic power management of real-time events. Yang et al. [22] propose to control preemption to reduce the idle energy consumption in devices. Devadas and Aydin [9] adopt device forbidden regions for preemption to prolong the idle time so that a device can be put to the sleep mode as long as possible. Moreover, Augustine et al. [2] explore systems with multiple low-power states, and design an online competitive algorithm to decide which low-power state the system should enter when there is no event in the ready queue. However, there is no procrastination in [2] to aggregate the idle time, and the energy consumption overhead because of the activating and deactivating of the device may be much larger than the saved energy.

Most of the above approaches require regular event arrivals, such as periodic real-time events [6, 9, 22], or precise information about event arrivals, such as aperiodic real-time events with known arrival times [3, 4]. However, when the events might have burst or may come to the system without regular or precise event arrival patterns, these approaches cannot be applied. The non-determinism in the timing of event arrivals comes from two main reasons: (a) An event may be triggered by the (physical) environment, which can often not be predicted accurately. (b) When a distributed system is considered, an event might be triggered by other events on different processing components, in which variable processing times, communication delays, and interferences on shared resources make the prediction of precise arrival times extremely complicated if not impossible. For coping with such settings, Real-Time Calculus (RTC) [20], which is based on Network Calculus [8, 14], can be adopted, as it features the modeling and analysis of systems with regular and irregular event arrivals. Specifically, its arrival curves define upper and lower bounds of the number of event arrivals of the workload to be handled within specified time interval lengths. Even though the number of arrivals is bounded, it is important to note that the actual event arrival times are not assumed to be known. Hence event arrival curves feature models which range from strictly periodic event arrivals, streams where event arrivals are Periodic with Jitter and a minimum Distance (i. e., PJD task model), up to sporadic event arrivals. Arrival curves together with an adequate model of the resource availability allow one to compute bounds on delays and buffer sizes and thereby allow to verify that a given system design meets its *hard* real-time requirements.

On the basis of RTC, Maxiaguine et al. [15] compute a safe frequency at periodical interval with predefined length to prevent buffer overflow of a system for DVFS-based scheduling. The authors of [17] explore the schedulability for online DVFS scheduling algorithms as proposed by Yao et al. [23]. Without using arrival curves, Irani et al. [11] explore how to perform online scheduling to decide which frequency a processor should execute and whether a processor should be activated to serve events or should stay in the sleep mode for energy reduction. However, since the online algorithm in [11] greedily stays in the sleep mode as long as possible without referring to incoming events in the near future, the resulting schedule might make an event miss its deadline (due to burst arrivals and late activation to the active mode). To cope with these concerns, Huang et al. [10] exploit RTC-based traffic and resource modelling. The presented online algorithms in [10] rely on a (finite) trace of past event arrivals and exploit expensive numerical operations such as (de)convolution of arrival curves for deriving history-aware bounds on future workload. However, the imposed memory and computation overheads turn out to be not negligible and severely affect the applicability of their proposed DPM technique. This paper presents dynamic counters, a scheme which does not suffer from these deficiencies of the approaches in [10].

1.3 Organization

The remainder of this paper is organized as follows: Sec. 2 reviews basic concepts from theory. Sec. 3 introduces the concept of dynamic counters and shows their usage for bounding future workload. In Sec. 4 we show how dynamic counters can be exploited for online DPM. For conciseness the discussion focuses on single event stream models, an extension to devices processing events of different types appears to be straightforward. A benchmarking of dynamic counters in case of online DPM is presented in Sec. 5. Sec. 6 concludes the paper.

2. BACKGROUND THEORY

2.1 Event streams

A timed event is a pair (t, τ) where $t \in \mathbb{R}_{\geq 0}$ is some non-negative time stamp and $\tau \in Act$ is some event label or type with Act as a set of event labels or event types. A (timed) trace $tr := (t_1, \tau); (t_2, \tau); \ldots$ is a sequence of timed τ-events ordered by increasing time stamps, s. t. $t_i \leq t_{i+1}$ for $i \in \mathbb{N}$ and $\tau \in Act$. Given a trace tr one may apply a filter-operation which for a given event type $e \in Act$ removes all pairs (t_i, τ) from tr where $\tau \neq e$ holds. A *stream* (of an event type) is defined as a set of filtered, typed resp., traces. In the context of analysis of (hard) real-time systems, streams are commonly characterizing by real-time traffic models. These models range from strictly periodic event arrivals over PJD-streams, i. e., traces where event arrivals are periodic with jitter and a minimum distance, up to models featuring sporadic event arrivals. In this setting events of a stream are assumed to be processed by a task running on a computing device, which consumes a number of resource units, e. g., clock cycles.

The description of the proposed dynamic counters and their application to online DPM is based on event streams and their relation to resource usage. However, for simplicity we do not transfer event arrivals into resource units to be provided by the device. Such a scaling is, e. g., in processor cycles seems straight forward, as it simply requires a scaling of a stream's bounding functions as introduced next.

2.2 Bounding streams: Event Arrival Curves

Beyond traffic models, event streams can also be abstractly characterized by so called arrival curves [20]. An arrival curve is a pair of upper and lower arrival curves , denoted by $\alpha := (\alpha^l, \alpha^u)$, with the following property:

$$\alpha^l(t-s) \leq R(t) - R(s) \leq \alpha^u(t-s),\ \forall\, 0 \leq s \leq t, \quad (1)$$

where R is the cumulative event counting function which reports the number of event arrivals for the respective stream. It is important to note that arrival curves, like the PJD-model, defines a possibly infinite set of traces, namely all traces of the respective type whose cumulative counting function R is bounded as defined above.

In the following it is assumed, that for a stream, its absolute bounding curve α is provided to the system engineer at design time. As for conciseness, we focus on single-stream scenarios in the following, but, the presented scheme can easily be adopted for multiple event streams.

2.3 Deriving a history-aware upper bound for future event arrivals

As pointed out above, a trace's cumulative counting function $R(t)$ is upper-bounded by the upper arrival curve α^u. At time t, to bound the events in the near future, we can use $\alpha^u(\Delta)$ to bound the event arrivals in time $(t, t+\Delta]$. However, α^u only provides information in the *interval domain*. For example, if many event arrivals have been observed recently, it could be foreseen that events might arrive more sparsely in the near future. Hence, a tighter bound than α^u has to consider the event arrivals in the past. Such a (history-aware) bounding function, will be denoted as $\mathcal{U} : \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \to \mathbb{N}$ in the reminder of this paper. Independently from the concrete definition of $\mathcal{U}$, at time t the following condition must hold:

$$\forall \Delta \in \mathbb{R}_{\geq 0} : \mathcal{U}(\Delta, t) \geq R(t+\Delta) - R(t).$$

In the following we briefly point out how bounding function $\mathcal{U}$ can be computed by explicitly considering past event arrivals up to time $t - \Delta^h$.

Based on the RTC framework [20, 21], the following properties can be obtained to bound the future event arrivals (as discussed in [10]): let the length of the history window for which a controller can record a stream's history be of size Δ^h, i. e., historical information for only Δ^h time units is stored. At time t a "historical event curve" can be defined as:

$$H(\Delta, t) = \begin{cases} R(t) - R(t-\Delta), & \text{if } \Delta \leq \Delta^h, \\ R(t) - R(t-\Delta^h), & \text{otherwise} \end{cases} \quad (2)$$

where by construction for all $\lambda \in \mathbb{R}_{\geq 0}$ with $\lambda \leq \Delta^h$ it holds that $H(\lambda, t) \leq R(t) - R(t-\lambda)$.

Eq. (1) states that

$$\alpha^u(\Delta) \geq R(t+\Delta) - R(t) \text{ and } \alpha^u(\lambda) \geq R(t) - R(t-\lambda)$$

must hold. This yields:

$$\alpha^u(\Delta + \lambda) \geq R(t+\Delta) - R(t-\lambda).$$

The above equation with the right-hand side extended by $-R(t) + R(t)$ together with the property of the historic event arrival curve yields:

$$\forall \lambda \in \mathbb{R}_{\geq 0} : \alpha^u(\Delta+\lambda) - H(\lambda, t) \geq R(t+\Delta) - R(t).$$

As one wants to bound $R(t+\Delta) - R(t)$ as tightly as possible, i. e., to compute the smallest number which fulfills the above equation, one needs to compute the infinum on bounding curve $\alpha^u(\Delta+\lambda) - H(\lambda, t)$. This gives us the following definition for bounding function $\mathcal{U}_{TB}$ which is based on a finite memory of past event arrivals:

$$\mathcal{U}_{TB}(\Delta, t) = \inf_{\lambda \geq 0} \left\{ \alpha^u(\Delta+\lambda) - H(\lambda, t) \right\}, \quad (3)$$

where $\inf$ is the infimum. Function $\mathcal{U}_{TB}$ bounds at time t the maximal number of future event arrivals up to time $t + \Delta$. Note that TB refers to the fact that $\mathcal{U}_{TB}$ is trace-based.

The above bound on the number of future event arrivals requires the maintenance of a stream's history back to time $t - \Delta^h$. As already pointed out in Sec. 1.1 and Sec. 1.2 this requires non-negligible memory space and computation time. The computation overhead may become significant as Eq. (3) requires the computation of function H and a deconvolution of α^u and H to obtain the trace-based bounding function $\mathcal{U}_{TB}$. Moreover, once significant parts drop out of the recorded history the bounding function $\mathcal{U}_{TB}(\Delta, t)$ becomes significantly pessimistic.

Unlike schemes making use of explicitly recorded histories, a technique based on dynamic counters as presented in Sec. 3 does not suffer from this effect.

3. DYNAMIC COUNTERS

3.1 Preliminaries

As pointed out above, the curves of α are time-invariant, i. e., they do not contain any information where in the current trace the worst- and best-case behavior will be seen. Consequently, the usage of α for bounding the worst- or best-case number of future event arrivals is extremely pessimistic. It is our aim to deduce a tighter bound for the number of future event arrivals, where we employ dynamic counters for tracking a stream's history w. r. t. some arrival curve α.

Algorithm 1 Implementing a dynamic upper counter

```
1: procedure DynamicUpperCounter(signal s)
2:     if s = event_arrival then
3:         if TOP = N^u then
4:             setTimer(CLCKUp, δ^u)
5:             k = 0
6:         end if
7:         TOP := TOP − 1
8:     end if
9:     if s = timer_elapsed then
10:        TOP := max(TOP + 1, N^u)
11:        setTimer(CLCKUp, δ^u)
12:        k = k + 1
13:    end if
14:    if TOP < 0 then
15:        Throw_Overflow_Execption
16:    end if
17: end procedure
```

At first, for illustration purpose, the discussion is restricted to the case of streams which are bounded by a single upper staircase function and a single lower staircase function, each with a constant step-width δ^u, δ^l respectively. This yields the following setting:

$$\begin{aligned} &\text{(a) Upper bound:} \quad \alpha^u(\Delta) := N^u + \lfloor \tfrac{\Delta}{\delta^u} \rfloor \\ &\text{(b) Lower bound:} \quad \alpha^l(\Delta) := -N^l + \lfloor \tfrac{\Delta}{\delta^l} \rfloor \end{aligned} \tag{4}$$

with $N^{\{u,l\}} \in \mathbb{N}$ and $\delta^{\{u,l\}} \in \mathbb{R}_{>0}$. For tracking the worst- and best-case behavior of a stream bounded by the above arrival functions this paper introduces the dynamic counters b^u and b^l. They are implemented by variables TOP and BOT of algorithm `DynamicUpperCounter` (Algo. 1) and algorithm `DynamicLowerCounter` (Algo. 2). These algorithms, discussed in detail below, need to be executed in a control loop in a time- and event-triggered fashion. Algorithm `DynamicUpperCounter` is executed every δ^u time units, whereas algorithm `DynamicLowerCounter` is executed every δ^l time units. Upon an event arrival both algorithms have to be executed. For simplification we assume the presence of a respective signal s which is passed from the control loop to the algorithms in order to invoke adequate reactions on their side.

In a nutshell, the values of counter TOP and counter BOT at mission time t together with the slope of the curves of Eq. (4), allow one to conservatively bound the number of future event arrivals. With these bounds one may schedule the on and off-periods of a device as briefly discussed in Sec. 4.

Renewal points. With dynamic counters the system may reach renewal points. Renewal points are states beyond which the past streaming behavior of the environment has no influence on the bound on the number of future event arrivals. In the algorithms this is covered by line 3-6, where the global timers, $CLCKUp$ and $CLCKLow$, and the local variable k are all reset. For the properties of a dynamic counter driven bounding of future event arrivals as derived in Sec. 3.4 the existence of renewal points is quite convenient. It allows one to limit the time horizon of the considered history, s. t. one solely needs to deal with a somehow relativized mission time t, rather than considering the whole history, since the system starts. More details follow below and in Sec. 3.3.2.

3.2 Implementing dynamic counters and the bounding of future event arrivals

3.2.1 Upper dynamic counter

Algo. 1 is used for tracking the behavior of a stream w. r. t. its maximum number of event arrivals. The algorithm operates on a global counter TOP and a global timer $CLCKUp$. The core idea of the approach is to describe the current state of the stream at time t by a single counter TOP. For achieving this TOP is decremented upon each event arrival (line 2-8) and incremented when global timer $CLCKUp$ elapses which happens every δ^u time unit (line 9-17); as pointed out above the invocation of the algorithm takes place in some external control loop.

At time t, counter TOP reports the *potential burst capacity*. According to the definition of α^u in Eq. (4), parameter N^u gives the maximum number of events which may arrive at the same instant of time. It will be denoted as absolute burst capacity in the following.

With the above functionality, Algo. 1 and its variable TOP implement the following function:

$$b^u(t) := \min(N^u + \lfloor \frac{t}{\delta^u} \rfloor - R(t), N^u), \tag{5}$$

where at current time t counter $TOP = b^u(t)$. In the above equation $R(t)$ is the cumulative counting function of the filtered trace in time $[0, t)$. Here it records the number of event arrivals up to time t. One may note, that we do not need to track this number explicitly, as TOP is incremented upon each event arrival. Furthermore, one should note that if an event arrival takes place and $TOP = N^u$ holds, Algo. 1 resets timer $CLCKUp$ and local variable k (cf. line 3-6) As already pointed out before, this refers to the reaching of a renewal point. The renewal point is due to the fact that up to now less than $\lfloor \frac{t}{\delta^u} \rfloor$ events have arrived which implies that the history is irrelevant for the future bound as the potential burst capacity is equal to the absolute burst capacity ($TOP = N^u$). The reset of timer $CLCKUp$ needs to be done as one does not want to falsely overestimate the number of future event arrivals and thereby violate the absolute the upper bound imposed by α; see also Sec. 3.3.2.

Auxiliary variable k tracks the offset between current time t and the last δ^u time step (see also line 11), i. e., local variables k is the smallest integer s. t. $k\delta^u \leq t$ holds. This functionality allows to deduce the following function:

$$q^u(\Delta, t) := \begin{cases} \lfloor \frac{\Delta + (t - k\delta^u)}{\delta^u} \rfloor & \text{if } b^u(t) < N^u \\ \lfloor \frac{\Delta}{\delta^u} \rfloor & \text{if } b^u(t) = N^u \end{cases} \tag{6}$$

It bounds the number of events arriving in addition to the potential burst capacity in the time interval $[t, t + \Delta]$. Potential burst capacity, i. e., value of TOP, together with function q defined in Eq. (6) yields:

$$\mathcal{U}(\Delta, t) := b^u(t) + q(\Delta, t). \tag{7}$$

The above function bounds future event arrivals. Unlike function $\mathcal{U}_{TB}$ of Eq. (3) it is, however, independent of a trace recording and its computation appears to be less complex; $\mathcal{U}_{TB}$ requires a deconvolution of function α^u and the historic event arrival curve H.

One may also note that $b(t') = N^u$ at time t' yields: $\mathcal{U}(\Delta, t') = \alpha(\Delta)$ (cf. to Eq. (4)(a) and Eq. (6)). This is what we called a renewal point, reached here by the system at time t'.

Finally, it needs to be pointed out that in case the evaluation of function $\mathcal{U}$ (Eq. (7)) coincidences with the elapsing of timer $CLCKUp$, i. e., δ^u is an integer remainder of t, Algo. 1 has to be executed first, This is necessary as otherwise one would underestimate the number of future event arrivals.

Algorithm 2 Implementing a dynamic lower counter

```
1: procedure DynamicLowerCounter(signal s)
2:     if s = event_arrival then
3:         if BOT = −N^l then
4:             setTimer(CLCKLow, δ^l)
5:             k = 0
6:         end if
7:         BOT := max(BOT − 1, −N^l)
8:     end if
9:     if s = timer_elapsed then
10:        if BOT = 0 then
11:            Throw_Underflow_Execption
12:        end if
13:    else
14:        BOT := BOT + 1
15:        k = k + 1
16:        setTimer(CLCKLow, δ^l)
17:    end if
18: end procedure
```

3.2.2 Lower dynamic counter

Algo. 2 works similarly to the algorithm discussed above, but tracks the number of event arrivals w. r. t. lower bound α^l of Eq. (4). In case an event arrives and $BOT = -N^l$ holds Algo. 2 resets timer $CLCKLow$ (line 3). This feature refers once again to the reaching of a renewal point, where here up to current time t sufficiently many event arrivals have been observed, s. t. the history up to the current time t is irrelevant for the lower bound on future event arrivals, see also Sec. 3.3.2.

Constant $|-N^l|$ is the fictitious maximum number of δ^l steps which may pass at most between any pair of event arrivals, i. e., the maximum distance of two events is consequently bounded by $\frac{N^l}{\delta^l}$. In a nutshell, the algorithm allows one to anticipate a lower bound on the future event arrivals:

$$\mathcal{L}(\Delta, t) := \max(b^l(t) + q^l(\Delta, t), 0), \tag{8}$$

where at time t counter BOT holds the value of $b^l(t)$; initially BOT is set to $-N^l$. Function $q^l(\Delta, t)$ bounds the minimal number of additional future event arrivals. These functions which can be computed by exploiting Algo. 2 and Eq. (4) are defined as follows:

$$\begin{aligned} b^l(t) &:= \max(-N^l + \lfloor \tfrac{t}{\delta^l} \rfloor - R(t), -N^l) \\ q^l(\Delta, t) &:= \begin{cases} \lfloor \frac{\Delta + (t - k\delta^l)}{\delta^l} \rfloor & \text{if } b^l(t) > -N^l \\ \lfloor \frac{\Delta}{\delta^l} \rfloor & \text{if } b^l(t) = -N^l. \end{cases} \end{aligned} \tag{9}$$

As before, $R(t)$ is the cumulative counting function of the observed stream. It is implicitly contained in the value of BOT, as the dynamic counters track potential and actual event arrivals. Therefore, the explicit storage of $R(t)$ is, once again, not necessary.

Analogously to the upper dynamic counter, the system has reached a renewal point once $b^l(t') = -N^l$ holds and $\mathcal{L}(\Delta, t')$ coincides with the absolute bound $\alpha^l(\Delta)$ (cf. Eq. (9), Eq. (8), and Eq. (4)(b)).

Finally one may note that in case the evaluation of function $\mathcal{L}$ of Eq. (8) coincidences with the elapsing of timer $CLCKLow$, i. e., δ^l is an integer remainder of t Algo. 2 needs to be executed before evaluating Eq. (8). This is necessary as otherwise function $\mathcal{L}$ would underestimate the number of future event arrivals.

3.3 Additional features of dynamic counters

In Sec. 3.2, we discussed details about the implementation of dynamic counters and their use for bounding future event arrivals. In the following, we will give details on the detection of bound violations, i. e., event arrivals do not comply to the bounding arrival curves.

3.3.1 Detection of over- and underflows

The dynamic counters can not only be employed for anticipating future event arrivals, but also allow to detect over- and underflows of the tracked streams. We speak of an overflow of a stream in case upper bound α^u is violated, i. e., in case we are actually facing more event arrivals as defined. We speak of an underflow of a stream in case lower bound α^l is violated, i. e., one actually sees less events as defined.

Overflow. If $TOP = 0$ and an event arrives before periodic incrementation could have been executed, an overflow has occurred, i. e., more than $N^u + \lfloor \frac{\Delta}{\delta^u} \rfloor$ events have actually arrived in interval $\Delta = [0, t]$. This overflow can easily be detected and the algorithm can indicate this to the environment, e. g., the operating system (cf. line 13-16 of Algo. 1).

Underflow. An underflow has happened if dynamic counter $BOT = 0$ and the periodic decrement is executed. This setting refers to the situation that not sufficiently many event arrivals within interval $\Delta := [0, t]$ have happened which is a violation of α^l. This is detected by algorithm `DynamicLowerCounter` which throws an exception to the environment, e. g., the operating system (cf. line 10-12 of Algo. 2).

3.3.2 Renewal points

Dynamic counters track the state of a stream and it can be observed that they might reach a state where the event arrivals of the past become irrelevant for the bound on future event arrivals. Such states are denoted as renewal points. Their occurrence can be explained as follows:

Renewal point w. r. t. the upper bound. Such a state is reached if not sufficiently many event arrivals have taken place. For the upper dynamic counter TOP this refers to the situation that it hits its maximum N^l and an event arrival occurs. This behavior is implemented by line 3-6 of Algo. 1, it has to be enforced as the potential burst capacity is equal to the absolute one and here α^u is the only valid upper bound, i. e., $\mathcal{U}(\Delta, t) = \alpha^u(\Delta)$ holds.

Renewal point w. r. t. the lower bound. Here one has observed sufficiently many event arrivals in the past, i. e., BOT is equal to $-N^l$ and an event arrival takes place. This behavior is implemented by line 3-6 of Algo. 1, it has to be enforced as sufficient events have been seen in the past and α^l is a tight lower bound, i. e., $\mathcal{L}(\Delta, t) = \alpha^l(\Delta)$ holds.

The existence of renewal points yields that besides dynamic counters, also auxiliary variables have the potential for being bounded. However, in case a system never reaches a renewal state, e. g., one deals with a stream where a burst is followed by strictly periodic event arrivals, auxiliary variable k of Algo. 1 and Algo. 2 may not be bounded. This situation can be resolved by simply restarting Algo. 1 at an arbitrary point in time which resets its mission time t and variable k to 0. This operation is safe as it solely introduces pessimism until the reaching of the next renewal point by overestimating the possible event arrivals $\mathcal{U}$ in the future. The point when to restart needs of course to be chosen carefully as it

corrupts the quality of the bound on future event arrivals. In case of Algo. 2, however, a reset needs to take place on event arrivals only; otherwise, the number of future event arrivals would be underestimated.

3.4 Tightness and safeness of bounds

In the following we only show that $\mathcal{U}$ is a tighter bound on future event arrival than α^u, and that $\mathcal{U}$ is a conservative bound, i. e., it does not underestimate the maximal number of event arrivals. Analogously it can be shown then that $\mathcal{L}$ is a tighter bound w. r. t. α^l and that it is conservative too.

We will show that $\mathcal{U}$ is tighter than α^u and present the conservative property by showing the following relation:

$$\alpha^u(\Delta) \geq \mathcal{U}(\Delta, t) \geq R(t+\Delta) - R(t),$$

where R is the cumulative counting function of the filtered trace.

3.4.1 Tightness of bound

THEOREM 1. *History-dependent function $\mathcal{U}(\Delta, t)$, at time t, provides a tighter bound for the number of future event arrivals than arrival curve $\alpha^u(\Delta)$, i. e., $\forall t, \Delta \in \mathbb{R}_+ : \mathcal{U}(\Delta, t) \leq \alpha(\Delta)$.*

PROOF. We just have to prove this theorem by showing that $\mathcal{U}(t-s, t) = b^u(t) + \lfloor \frac{\Delta + (t - k\delta^u)}{\delta^u} \rfloor \leq \alpha^u(t-s)$. If $b^u(t) = N^u$, the above statement is trivial, as $\mathcal{U}(\Delta, t) = \alpha^u(\Delta)$. If $0 \leq b^u(t) < N^u$, as $t - k\delta^u < \delta^u$ and with $b^u(t) < N^u$, it follows $\mathcal{U}(\Delta, t) \leq \alpha^u(\Delta)$. □

In the above setting, it is interesting to note that for $b^u(t) \ll N^u$, i. e., for small values of the history dependent dynamic counter, the history-dependent bound $\mathcal{U}(\Delta, t)$ may be substantially smaller than the arrival curve $\alpha^u(\Delta)$.

3.4.2 Conservative bound

THEOREM 2. *Function $\mathcal{U}(\Delta, t)$ is a conservative bound w. r. t. a cumulative counting function R of a filtered trace and all time horizons of length Δ, resp. future time durations starting at time t and ending at time $t + \Delta$.*

PROOF. Let s be the last renewal point, s. t. we are dealing with a relativized t; this holds at least for starting time $s = 0$. To prove the theorem we need now to distinguish if δ^u is an integer divider of (the relativized) t or not where for these two different cases we also have to cover the different value ranges of $R(t)$.

δ^u is an integer divider of t ($t = k\delta^u$) . The following two value ranges for $R(t)$ have to be covered.

- **Case 1** : $0 \leq R(t) \leq \lfloor \frac{t}{\delta^u} \rfloor$. This setting gives that $b^u(t) = N^u$. For constructing a contradiction, let $c > 0$ be the positive number of events that the dynamic counter driven bounding falsely ignores, i. e., we assume that $\mathcal{U}(\Delta, t)$ is not a conservative bound at time t. The assumption $R(t) \leq \lfloor \frac{t}{\delta^u} \rfloor$ yields than: $\mathcal{U}(\Delta, t) + c = N^u + \lfloor \frac{\Delta}{\delta^u} \rfloor + c$ which has to bounded by α^u by definition. In a nutshell, we therefore have: $N^u + \lfloor \frac{\Delta}{\delta^u} \rfloor + c \leq \alpha^u(\Delta)$ which is a contradiction as $c > 0$ and $\alpha^u(\Delta) = N^u + \lfloor \frac{\Delta}{\delta^u} \rfloor$.
- **Case 2:** $\lfloor \frac{t}{\delta^u} \rfloor < R(t) \leq N^u + \lfloor \frac{t}{\delta^u} \rfloor$. Function $b^u(t) = N^u + \lfloor \frac{t}{\delta^u} \rfloor - R(t)$ reports the number of events that one could have seen, but actually has not seen; please recall that $R(t)$ is the number of events which have arrived up to time t. Together with the potentially ignored events we obtain that $N^u + \lfloor \frac{t}{\delta^u} \rfloor - R(t)) + \lfloor \frac{\Delta}{\delta^u} \rfloor + R(t) + c \leq \alpha^u(t + \Delta)$ has to hold. However, this bound would trivially hold only for $c < 0$. Thus one can conclude that with dynamic counters one does not falsely ignore future event arrivals and $\mathcal{U}(\Delta, t)$ is not only tight, but also conservative too.

δ^u is not an integer divider of t ($t > k\delta^u$) . This case can be handled in the above manner. With $t - k\delta^u < \delta^u$ fraction $\lfloor \frac{\Delta + (t - k\delta^u)}{\delta^u} \rfloor$ can be partitioned accordingly.

□

3.5 Handling complex event arrival patterns

In the context of this work we are dealing with discrete numbers of arrivals and their arrival patterns. In principle any (discrete) complex arrival pattern can be bounded by an upper and lower, staircase event arrival curve, as long as the system under consideration is monotone and time-invariant. The *monotone* property means that a higher number of input events seen in an interval yields a higher number of output events in intervals of equal or larger sizes. The *time-invariant* property means that the system behavior depends on the system states only. No matter when this state is reached, the possible set of the reactions of the systems is always the same, independently upon the concrete time when the actual state is reached.

3.5.1 Approximating an upper event arrival curve

The minimum computation on a set of staircase curves of the form $\alpha_i(\Delta) := N_i^u + \left\lfloor \frac{\Delta}{\delta_i^u} \right\rfloor$ is a conservative approximation of an upper event arrival curve α if the following condition applies:

$$\forall \Delta \in \mathbb{R}_{\geq 0} : \alpha^u(\Delta) \leq \min_{i \in \mathcal{I}}(\alpha_i^u(\Delta)). \tag{10}$$

Each of the upper staircase functions α_i in the above equation is represented by its own dynamic counter TOP_i. Hence one can deduce a set of bounding functions $\mathcal{U}_i$ by applying Eq. (7). For bounding the number of future event arrivals w. r. t. a stream S one can simply take the minimum over all these functions:

$$\mathcal{U}(\Delta, t) := \min_i(\mathcal{U}_i(\Delta, t)). \tag{11}$$

3.5.2 Approximating a lower arrival curve

Analogously a set of staircase curves and maximum computation can be employed for approximating a lower curve:

$$\alpha^l(\Delta) \geq \max_{j \in \mathcal{J}}(0, \alpha_j^l(\Delta)). \tag{12}$$

Each of the individual staircase functions α_j^l in the above equation is defined according to Eq. (4), i. e., $\alpha_j^l(\Delta) := -N_j^l + \left\lfloor \frac{\Delta}{\delta_j^l} \right\rfloor$. Once again we track each of the curves with its own individual dynamic counter, here BOT_j. This allows one to deduce a bounding function $\mathcal{L}_j$ by applying Eq. (8). For bounding the number of future event arrivals w. r. t. a stream S from below we simply take the maximum over all individual staircase functions:

$$\mathcal{L}(\Delta, t) := \max_j(\mathcal{L}_j(\Delta, t)). \tag{13}$$

Example. An example for the approximation of an arrival curve by means of individual staircase curves is given in Fig. 1. α^u is given as the minimum of three staircase curves α_1^u, α_2^u and α_3^u. Lower curve α^l is the maximum of the constant 0-function and the two staircase curves α_1^l and α_2^l. The individual staircase curves are

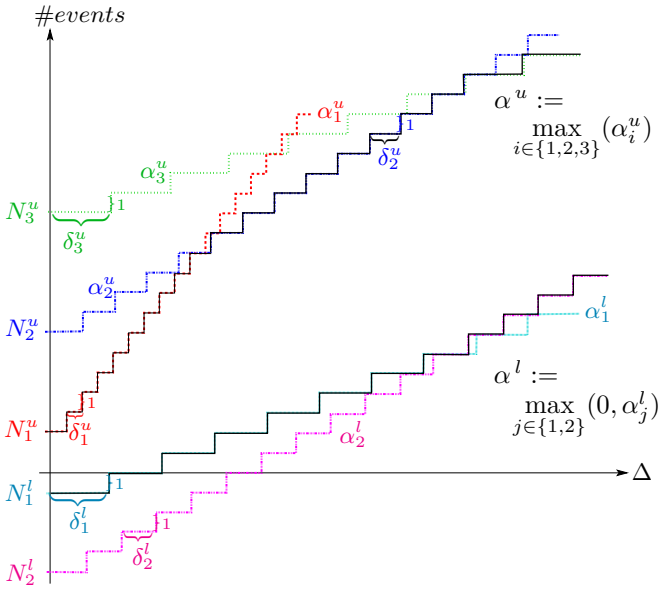


Figure 1: Arrival curves as combination of staircase functions

tracked by five corresponding dynamic counters $TOP_{\{1,2,3\}}$ and $BOT_{\{1,2\}}$. The values for instantiating the resp. algorithms can be deduced from Fig. 1.

Remark. The fact that the approximating RTC-curves of Eq. (10) and Eq. (12) are constructed by a single minimum and maximum operation yields that α^u has increasing and α^l has decreasing step widths, i. e., the distances between two jump points grows, shrinks resp., with Δ. This feature which one may denote as pseudo concaveness/convexity reduces the complexity of the approach, but is not a pre-requisite. When dealing with non-pseudo-concave/convex upper and lower bounds one may employ additional dynamic counters and include the additional functions $\mathcal{U}_i$ and $\mathcal{L}_j$ in nested minimum and maximum operations and embed this into Eq. (11) and Eq. (13) for computing function $\mathcal{U}$ and $\mathcal{L}$.

3.5.3 Handling of PJD arrival patterns

As case studies, we will consider the periodic with jitter event traffic model (PJD). This directly translates to upper and lower event arrival curves [21]. The upper bounding function α^u for such models can be represented as minimum of at most two staircase functions. A PJD stream is defined by three parameters: period p, maximum jitter j, and minimum event inter-arrival time d. For the lower bound of a PJD-based approximation one staircase function α^l is always sufficient.

The parameters of the staircase functions can be computed as follows:

- Case $d = 0 \ \vee \ d \leq p - j$:
  $N^u := \left\lceil \frac{j}{p} \right\rceil + 1; \qquad N^l := -\left\lceil \frac{j}{p} \right\rceil; \qquad \delta^u := \delta^l := p$
- Case $d > 0 \ \wedge \ d > p - j$:
  $N_1^u := 1; \qquad \delta_1^u := d; \qquad N_2^u := \left\lceil \frac{j}{p} \right\rceil + 1;$
  $N^l := -\left\lceil \frac{j}{p} \right\rceil; \qquad \delta_2^u := \delta^l := p$

Note that in fact an exact representation of a PJD model's upper curve by means of staircase functions α_1^u and α_2^u is in fact not always possible. However, in case horizontal shifting of staircase functions is excluded the above formulae guarantee a correct (i. e., conservative) approximation of a PJD-approximation. Otherwise one can achieve this by employing three staircase functions instead.

4. EXPLOITING DYNAMIC COUNTERS IN ONLINE DPM

In the following, we will use the developed dynamic counters for online DPM, by adopting the results in [10]. For the sake of completeness we briefly highlight the foundation and organization of such a scheme.

Reductions in energy consumption can be obtained by turning the device into sleep mode. Switching from/to sleep mode incurs overhead as switching to sleep mode takes $t\downarrow$ time and switching back to active mode takes $t\uparrow$ time. Hence a shutting down is economically useful, if and only if the energy savings of the sleeping phase are larger as the switching costs. Let the energy consumption for switching the device into sleep mode be $E\downarrow$ and let $E\uparrow$ be the energy consumption for switching it back into active mode. This yields a device's threshold or *break-even time*:

$$T_{BET} := max(t\downarrow + t\uparrow, \frac{E\downarrow + E\uparrow}{\rho})$$

where ρ is the energy saving rate achieved by turning the device off instead of keeping it idling in its on mode. The above equation yields the threshold below which the switching to sleep mode is not feasible, due to physical or economic reasons.

In our method, we abstract the workload by considering only the number of incoming events. Each of these events will be processed by a dedicated task running on the device under management. Given the ingredients for computing bounds on future event arrivals one may schedule now the on- and off-periods of a device online. In the following we briefly introduce a heuristic to do this, where D is the relative deadline for each event and Q is a user-defined maximum on the size of the input buffer where the incoming events are stored a priori to their processing.

4.1 Deactivation of a device

In principle it seems possible to shut the device down at any time, as long as each event meets its relative deadline, i. e., its processing terminates before $t + D$, provided that the event arrived at time t. Hence one could turn the device off even if there are events in the input buffer waiting for being processed. However, it seems to be a better strategy to deactivate the device only if the input buffer is empty, i. e., there are no backlogged events. This is because, in such cases, it may be possible to prolong the sleep interval beyond time $t + D$ when no event arrives. Provided that there are switching costs associated with a device, the above heuristic gives rise to the question, whether one should always switch the device off, i. e., as soon as the buffer is empty, or do this only if the worst-case bound on the number of future event arrivals is significantly small. If the bounding of future event arrivals reveals that there is no event arrival before time $t + T_{BET}$ the switch in to the off-mode is beneficial. How to compute the time span τ for which no event arrivals can take place will be discussed next.

By considering the relative deadline D of events and the buffer bound Q one may determine the minimal computing capability to be provided by the device such that all deadlines are met and no buffer overflow occurs as follows:

$$\mathcal{W}(\Delta, t) := \max(\mathcal{U}(\Delta, t) - Q, \mathcal{U}(\Delta - D, t)) \qquad (14)$$

The above bounding function on the the number of events that can be processed in time interval $[t, t + \Delta)$ can be related to a device's

cumulative service function $C : \mathbb{R}_{\geq 0} \rightarrow \mathbb{N}$, which book-keeps the number of resource units consumed up to time t, as follows:

$$C(t + \Delta) - C(t) \geq \mathcal{W}(\Delta, t). \quad (15)$$

As already pointed out in Sec. 2.1 we only consider service in number of events. Suppose that the device needs a worst-case execution time of ω time units to process an event. Then, the largest time period τ for which a shutdown of the device is feasible can be determined, because $\forall \lambda \geq \tau : \lambda - \tau \geq \omega \cdot \mathcal{U}(\lambda, t)$ must hold.

4.2 Activation of a device

For deciding when to turn a device back in operation the above computation is not sufficient. One does not necessarily want to turn the device on after the sleeping period as elapsed, i. e., after exactly τ time units have passed. This is because, even with events in the input buffer, prolongation of the current off period might be feasible. However, this strategy requires consideration of the deadlines of the events buffered, up to current time t. Let E denote the set of buffered events. Then, the future workload in number of events is determined by the following term:

$$W^b(\Delta, t) = |\{i \in E : d_i \leq t + \Delta\}|, \quad (16)$$

where d_i is the absolute deadline of backlogged event i. Eq. (16) along with Eq. (15) allow us to compute the prolongation of the sleeping interval as before, but with the additional workload added to the bound on potential future event arrivals. To calculate the new (maximal) τ, the following condition must hold: $\forall \lambda \geq \tau : \lambda - \tau \geq \omega \cdot (\mathcal{U}(\lambda, t) + \mathcal{U}^b(\Delta, t)$. Once τ is reasonably close to 0 the device needs to be put back to the active mode.

5. EMPIRIC EVALUATION

For empirically evaluating the proposed technique we benchmark the online DPM scheme based on dynamic counters with comparison to the approach in [10]. Both methods exploit the ideas presented in Sec. 4 for online scheduling on and off periods of devices. However, *the techniques differ in the way how function $\mathcal{U}$ for bounding future event arrivals is computed.*

The work of [10] employs a finite trace of past event arrivals, recorded for the time interval $[t - \Delta^h, t]$. This finite history is used for computing an upper bounding function $\mathcal{U}_{TB}$ by mainly following the lines of Sec. 2.2. In the following this scheme will be addressed as *trace-based* approach and the obtained performance metrics will be indexed with identifier TB accordingly. In contrast the online DPM scheme based on dynamic counters uses the identifier DNC. For the DNC-based bounding of future event arrivals we apply the approximations of Sec. 3.5 with at most two staircase functions, i. e., . two upper dynamic counters respectively. Moreover, we exploit Eq. (5) - Eq. (7) in Sec. 3.2.1.

In the following, we will give experimental evidence that the usage of dynamic counters is advantageous. The simulations have been carried out on an AMD Opteron 2.6 GHz processor with 8 GB RAM and a Linux Operating system. The simulations were executed on top of the Matlab-based RTC/RTS-toolbox [1]. For computing the individual bounding functions $\mathcal{U}_{DNC}$ and $\mathcal{U}_{TB}$, we use the functions of the Real-time Calculus toolbox. The RTC/RTS-toolbox which we use is implemented in JAVA and as part of Matlab. Therefore, the reported run-times of the different schemes have to be considered with care, they should rather demonstrate tendencies, instead of being interpreted as hard results.

For simplicity we also restricted the benchmarking to a single event stream. An extension to multiple streams running on a device is straight-forward: one simply needs to sum the individual bounds on the different event arrivals of all streams processed on the device and use the aggregated bound for scheduling the on and off periods of the device; we assume that without online DPM the system is schedulable.

	S_1	S_2	S_3	S_4	S_5
period (msec)	111	198	354	194	119
jitter (msec)	341	387	387	260	187
delay (msec)	20	48	17	32	89
wcet (msec)	7	12	11	5	6

Table 1: Event stream settings.

Device Name	P_a (W)	P_s (W)	P_σ (W)	t_{sw} (S)	E_{sw} (mJ)
Realtek Ethernet	0.19	0.125	0.085	0.01	0.8

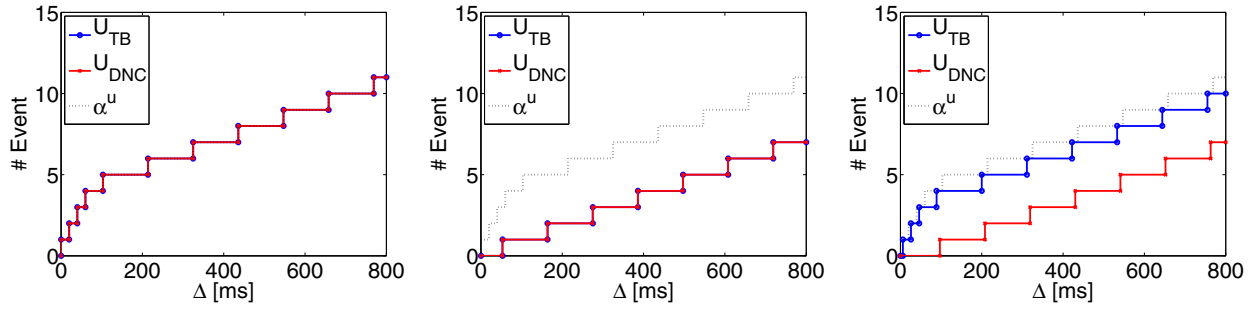
Table 2: Power profiles for a peripheral device.

5.1 Experimental Setup

In the following we consider PJD arrival patterns, uniquely defined by parameters `period`, `jitter`, and `delay`. The specifications of the used PJD streams for invoking computations in the device under management are given in Table 1. The event arrival curve bounding these streams can be bounded by at most two staircase functions of fixed step-length (cf. Sec. 3.5). The worst-case execution time (WCET) required for processing an event is also given in Table 1. The relative deadline D_i of a stream S_i is set equal to its period times a *deadline factor*. The history window Δ^h for the TB method is defined by a multiple χ of a streams period, i. e., $\Delta^h = \chi \cdot p_i$. In the following we denote χ as the *history window coefficient*. To benchmark the TB- and DNC-based DPM schemes, we simulate traces with a 100 sec time span. The individual traces for the experiments are generated by the RTS-toolbox [1] and according to the upper (PJD-based) arrival curves constructed from the data in Table 1. The user-defined backlog size, i. e., parameter Q of Eq. (14), is set to 20 events for each stream. In the simulation, we adopt the power profiles of a peripheral device, namely Realtek Ethernet [7]. The power consumption at active P_a, idle P_s, and off P_σ modes, as well as the time t_{sw} and energy E_{sw} overheads of this device are listed in Table 2.

5.2 Simulation results

Fig. 2 shows the results for functions α^u, $\mathcal{U}_{DNC}$ and $\mathcal{U}_{TB}$, for bounding future event arrivals of event stream S_1. In Fig. 2 we plotted the function at different stages, where for illustration purpose only extreme scenarios, i. e., corner cases are presented. A bound based on function α^u is time-invariant, hence (absolute) bound α^u remains the same in all three sub-figures of Fig. 2. At system start or when reaching a renewal point (cf. Sec. 3.3.2) all three bounding functions are the same which is illustrated in Fig. 2(A). This refers to the fact that either no history is available or the number of past event arrivals is insignificant w. r. t. the worst-case bound on the number of future event arrivals. However, as soon as significantly many events have arrived in the past, the TB- and DNC-based techniques allow to obtain tighter upper bounds on the number of future event arrivals; cf. Fig. 2(B). In Fig. 2(C) we illustrated another corner case. In this setting the TB method is more pessimistic. This is because, significant event arrivals, i. e., bursts, which have occurred in the past drop out of the recorded history, as the latter only covers event arrivals back to time $t - \Delta^h$. In the case of the DNC method, such an effect does not exist, the dynamic counters store a trace's history w. r. t. the number of event arrivals back to the last renewal point. (The behavior beyond such renewal



(A) Zero time or at a renewal point ($t = 0$) (B) Bursty arrivals within $t - \Delta^h$ (C) Bursty arrivals before $t - \Delta^h$

Figure 2: Bounding future event arrivals at time t: Coverage of corner cases.

points is insignificant w. r. t. the number of future event arrivals.) Event trace generation for obtaining the above corner cases appears to be straight-forward for PJD streams: one simply produces bursty arrivals followed by almost periodic event arrivals. At time $t = 0$ one obtains the graphs of Fig. 2(A), for time $0 < t \leq \Delta^h$ the ones depicted in Fig. 2(B). As soon as $t > \Delta^h$ the bursty event arrivals drop out of the recorded history and one can note the differences as depicted in Fig. 2(C). Even though we only reported the results for stream S_1, the results are very similar for other streams considered in this paper.

In the next experiment, we benchmark the computing time (in the simulations) for TB- and DNC-based online DPM schemes with a two-stream scenario, i.e., event streams S_2 and S_5 with earliest-deadline-first scheduling policy. The relative deadlines of these two event streams are set equal to their periods (deadline factors are 1). The obtained results are shown in Fig. 3: with approach TB we employed different sizes of time horizons, i. e., lengths of timed traces to be kept in the memory (Δ^h), which is steered by the history window coefficient. The abbreviation HAD denotes the history-driven device deactivations which refers to the computation of a potential sleep interval of length τ as explained in Sec. 4.1. The abbreviation WCG refers to the worst-case-greedy driven device activation strategy as introduced in Sec. 4.2.

We report the mean computing time for the activation/deactivation strategies for the 100 sec time span. As illustrated in Fig. 3, the computing time for both HAD and WCG is reduced significantly when using the counter-based approach, even though the bounding of future event arrivals is a small part of the online DPM scheme. The reason comes as follows: the DNC-based DPM methods are independent of any history recording, their effort is solely related to the length of the steps of the staircase functions employed for approximating α^u. The step lengths determine the period of the updates of the dynamic counters. In case of event arrivals the computational burden is also limited. For the DNC-based DPM methods one solely needs to decrease the dynamic counters (cf. line 2-8 Algo. 1), and to check for overflows (cf. line 14- 16 Algo. 1) when necessary. For the TB-based methods one needs to generate a time stamp, and inserts it into the history. It is easy to see that for larger values of Δ^h more past event arrivals need to be stored. This results in a more complex curve $H(\Delta, t)$ (Eq. (2)). Moreover, the convolution in $(\min, +)$ algebra for computing $\mathcal{U}_{TB}$ (Eq. (3)) demands more computing. Note that, the above results about timing overheads here are for showing the tendencies. However, it is not difficult to see that the usage of dynamic counters for deriving bounds on the number of future event arrivals appears to be generally more efficient than procedures based on calculating the convolution of the historical traces.

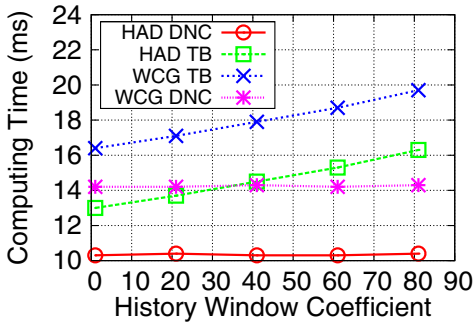


Figure 3: Computing times for TB and DNC-based DPM

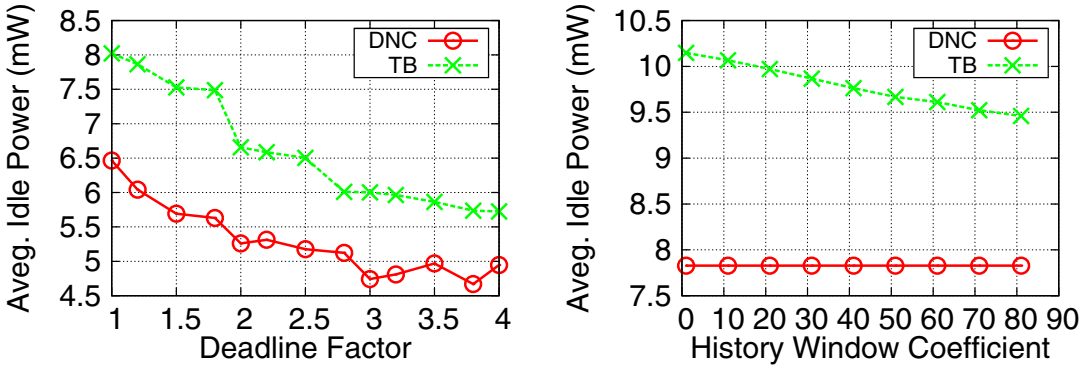
Fig. 4 demonstrates the power efficiency, i. e., the effectiveness, of the DNC-based DPM scheme, where we plotted the average idle power consumption (instead of the total power consumption). For different online DPM schemes, we use the same input traces for fair comparison. As, for a trace, the schemes only differ in the power related costs for idling and switching, where the energy consumption for processing the events remains the same for both schemes. The average idle power consumption is defined as:

$$\frac{E_{sw} \cdot C_{on,\,off} + T_{on,\,sum} \cdot \rho_{static}}{total\_time\_span}, \tag{17}$$

where E_{sw} is the energy overheads for one pair of activation and deactivation operations of the device; $C_{on,\,off}$ is the total numbers of activation/deactivation pairs; $T_{on,\,sum}$ the sum of the length of all the time intervals in which the device is on; and ρ_{static} is the power consumption saving achieved by turning the device off instead of keeping it idling in its active mode. To illustrate the power savings for different cases, we simulate two arbitrarily chosen cases: running streams S_3, S_4, and S_5 on the device with preemptive fixed-priority scheduling policy with different deadline factors, as shown in Fig. 4(A), and running streams S_2 and S_5 on the device with preemptive earliest-deadline-first scheduling policy with different history window sizes Δ^h, as shown in Fig. 4(B).

As shown in Fig. 4, DNC-based DPM can be more effective than its TB-based counterpart. However, by increasing the relative deadlines the effectiveness of TB-based DPM can be increased (cf. Fig. 4(A).) The reason is that with larger deadline factors the lengths of the computed sleeping intervals get larger (cf. Sec. 3). In turn, the device will therefore be shut down more often. A similar picture can be drawn for window coefficients: Fig. 4(B), shows that $\mathcal{U}_{DNC}$ is less pessimistic. This yields also a larger number of device shutdowns, instead of keeping the device idling.

In addition to deadline factors and window coefficients, the break-even time of the device and the distances of bursty arrivals



(A) Effectiveness and deadline factors

(B) Effectiveness and history window coefficients

Figure 4: Effectiveness of TB and DNC-based DPM

in the generated trace also influence the power-savings. This is because, with smaller values for τ (cf. Sec. 3) the TB-based DPM scheme will once again, i. e., similar to lower deadline factors and window coefficients, prefer idling over turning a device actually off. In the above experiments the parameters have been chosen as carefully as possible in order to not giving a biased picture in favor to a DNC-based DPM scheme.

6. CONCLUSION

This paper presents dynamic counters for bounding the number of future event arrivals. Furthermore it discusses the usage of such bounds in the context of online DPM, under hard real-time constraints. As demonstrated by the simulation results (Sec. 5) online DPM based on dynamic counters shows the potential for outperforming trace-based online DPM. Moreover, due to its simplicity, dynamic counters can be easily implemented as part of the hardware or lower operating system stack. This is of great value, as to the best of our knowledge so far bounding of future event arrivals has been dependent on explicit recording of past event arrivals and an application of computational expensive mathematical operations for deriving the actual bounding function from the stored traces.
It is worth mentioning that the presented technique goes beyond online DPM: similar to traffic shapers based on leaky buckets, dynamic counters support the on-the-fly detection of overflows and underflows of streams, i. e., violation of the number of event arrivals w. r. t. the bounding function defined at design time.

Acknowledgement.
This work was supported by the EU Framework Programme 7 projects EURETILE under grant number 247846.

7. REFERENCES

[1] Modular Performance Analysis Framework and Matlab Toolbox. www.mpa.ethz.ch.

[2] J. Augustine, S. Irani, and C. Swamy. Optimal power-down strategies. In *45th Symposium on Foundations of Computer Science*, pages 530–539, Oct. 2004.

[3] P. Baptiste. Scheduling unit tasks to minimize the number of idle periods: A polynomial time algorithm for offline dynamic power management. In *Proc. of the 17th annual ACM-SIAM symposium on Discrete algorithm*, pages 364–367, 2006.

[4] P. Baptiste, M. Chrobak, and C. Dürr. Polynomial time algorithms for minimum energy scheduling. In *ESA*, pages 136–150, 2007.

[5] J.-J. Chen and T.-W. Kuo. Procrastination for leakage-aware rate-monotonic scheduling on a dynamic voltage scaling processor. In *ACM SIGPLAN/SIGBED LCTES Conference*, pages 153–162, 2006.

[6] J.-J. Chen and T.-W. Kuo. Procrastination determination for periodic real-time tasks in leakage-aware dynamic voltage scaling systems. In *Int'l Conf. on Computer-Aided Design*, pages 289–294, 2007.

[7] H. Cheng and S. Goddard. Online energy-aware I/O device scheduling for hard real-time systems. In *Proc. of the 9th Design, Automation and Test in Europe*, pages 1055–1060, 2006.

[8] R. Cruz. A calculus for network delay. I. network elements in isolation. *IEEE Transactions on Information Theory*, 37(1):114–131, Jan 1991.

[9] V. Devadas and H. Aydin. Real-time dynamic power management through device forbidden regions. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 34–44, 2008.

[10] K. Huang, L. Santinelli, J.-J. Chen, L. Thiele, and G. C. Buttazzo. Applying real-time interface and calculus for dynamic power management in hard real-time systems. *Real-Time Systems*, 47(2):163–193, 2011.

[11] S. Irani, S. Shukla, and R. Gupta. Algorithms for power savings. In *Proc. of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 37–46, 2003.

[12] R. Jejurikar and R. K. Gupta. Procrastination scheduling in fixed priority real-time systems. In *Proc. of the ACM SIGPLAN/SIGBED LCTES Conference*, pages 57–66, 2004.

[13] R. Jejurikar, C. Pereira, and R. Gupta. Leakage aware dynamic voltage scaling for real-time embedded systems. In *ACM/IEEE Design Automation Conference*, pages 275–280, 2004.

[14] J. Le Boudec and P. Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Springer, 2001.

[15] A. Maxiaguine, S. Chakraborty, and L. Thiele. DVS for buffer-constrained architectures with predictable QoS-energy tradeoffs. In *the International Conference on Hardware-Software Codesign and System Synthesis*, pages 111–116, 2005.

[16] D. Park, J. Lee, N. S. Kim, and T. Kim. Optimal algorithm for profile-based power gating: A compiler technique for reducing leakage on execution units in microprocessors. In *ICCAD*, pages 361–364, 2010.

[17] S. Perathoner, J.-J. Chen, K. Lampka, N. Stoimenov, and L. Thiele. Combining optimistic and pessimistic dvs scheduling: An adaptive scheme and analysis. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 131–138, San Jose, California, USA, 2010. IEEE.

[18] A. Shrivastava, E. Earlie, N. Dutt, and A. Nicolau. Aggregating processor free time for energy reduction. In *Proc. of the 3rd IEEE/ACM/IFIP int. conf. on Hardware/software codesign and system synthesis*, pages 154–159, 2005.

[19] V. Swaminathan and K. Chakrabarty. Pruning-based, energy-optimal, deterministic I/O device scheduling for hard real-time systems. *ACM Transactions in Embedded Computing Systems*, 4(1):141–167, 2005.

[20] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *Proc. Intl. Symposium on Circuits and Systems*, volume 4, pages 101–104, 2000.

[21] E. Wandeler, L. Thiele, M. Verhoef, and P. Lieverse. System Architecture Evaluation Using Modular Performance Analysis - A Case Study. *International Journal on Software Tools for Technology Transfer (STTT)*, 8(6):649–667, Oct. 2006.

[22] C.-Y. Yang, J.-J. Chen, and T.-W. Kuo. Preemption control for energy-efficient task scheduling in systems with a dvs processor and non-dvs devices. In *RTCSA*, pages 293–300, 2007.

[23] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *Proc. of the 36th Annual Symposium on Foundations of Computer Science*, pages 374–382, 1995.