Applying real-time interface and calculus for dynamic power management in hard real-time systems

Real-Time Syst (2011) 47: 163–193
DOI 10.1007/s11241-011-9115-z

Applying real-time interface and calculus for dynamic power management in hard real-time systems

Kai Huang · Luca Santinelli · Jian-Jia Chen · Lothar Thiele · Giorgio C. Buttazzo

Published online: 26 January 2011

Abstract Power dissipation has been an important design issue for a wide range of computer systems in the past decades. Dynamic power consumption due to signal switching activities and static power consumption due to leakage current are the two major sources of power consumption in a CMOS circuit. As CMOS technology advances towards deep sub-micron domain, static power dissipation is comparable to or even more than dynamic power dissipation. This article explores how to apply dynamic power management to reduce static power for hard real-time systems. We propose online algorithms that adaptively control the power mode of a system, procrastinating the processing of arrived events as late as possible. To cope with multiple event streams with different characteristics, we provide solutions for preemptive earliest-deadline-first and fixed-priority scheduling policies. By adopting a worst-case interval-based abstraction, our approach can not only tackle arbitrary event ar-

The work was partially supported by projects EURETILE (grant No. 247846), PRO3D (grant No. 248776), and PREDATOR (grant No. 216008) of European Commission FP7.

K. Huang (✉) · L. Thiele
Computer Engineering and Networks Laboratory, ETH Zurich, Zurich, Switzerland
e-mail: khuang@tik.ee.ethz.ch

L. Thiele
e-mail: thiele@tik.ee.ethz.ch

L. Santinelli · G.C. Buttazzo
Real-Time Systems Laboratory, Scuola Superiore Sant'Anna of Pisa, Pisa, Italy

L. Santinelli
e-mail: luca.santinelli@sssup.it

G.C. Buttazzo
e-mail: giorgio@sssup.it

J.-J. Chen
Institute for Process Control and Robotics, Karlsruhe Institute of Technology, Karlsruhe, Germany
e-mail: chen@kit.edu

rivals, e.g., with burstiness, but also guarantee hard real-time requirements with respect to both timing and backlog constraints. We also present extensive simulation results to demonstrate the effectiveness of our approaches.

Keywords Power management · Real-time event streams · Real-time calculus · Real-time interface

1 Introduction

Power dissipation has been an important design issue in a wide range of computer systems in the past decades. Power management with energy efficiency considerations is not only useful for mobile devices for the improvement on operating duration but also helpful for server systems to reduce power bills. It has been shown in APC (American Power Conversion) (2003) that for server systems the electricity cost remains significant even if servers do not always operate with the maximum power consumption. For mobile devices, ITRS reports the slow growth of energy density of batteries lacks far behind the tremendous increase of demands (International technology roadmap for semiconductors 2009). Because of these facts, power consumption becomes one of the first-class design concerns for modern computer systems.

Two major sources of power consumption of a CMOS circuit are dynamic power consumption due to switching activities and static power consumption mainly due to leakage current (Jejurikar et al. 2004). For micrometer-scale semiconductor technology, dynamic power dominates the power consumption of a processor. However, as modern VLSI technology is scaling down to deep sub-micron domain, the tremendous amount of transistors integrated within a chip consumes significantly more static power. The leakage current that originates in the dramatic increase in both subthreshold current and gate-oxide leakage current is projected to account for as much as 50 percentage of the total power dissipation for high-end processors in 90 nm technologies (Austin et al. 2004). The ITRS expects static power in the future will be much greater than their calculated value due to variability and temperature effects (International technology roadmap for semiconductors 2009).

This article explores how to apply dynamic power management (DPM) to reduce static power consumption while satisfying real-time constraints. We consider a system that consists of a device with `active`, `standby`, and `sleep` modes with different power consumptions and a controller that decides when to change the power modes of the device. Intuitively, the device can be switched off to the sleep mode to reduce the power consumption when it becomes idle and switched on again to `active` mode upon the arrival of an event. These switching operations, however, need more careful consideration. On the one hand, the sleep period of the device after switching-off should be long enough to recuperate mode-switch overheads. On the other hand, when to activate the device is even more involved due to the possible burstiness of future event arrivals. For every switching-on operation, sufficient time has to be reserved to serve the possible burstiness of future events in order to prevent deadline violation of events or overflow of system backlog.

To resolve these concerns, we propose online algorithms that are applicable for the controller. This article summarizes and extends the results built in Huang et al. (2009, 2010). We apply Real-Time Calculus (Thiele et al. 2000) to predict future event arrivals and Real-Time Interface (Thiele et al. 2006) for schedulability analysis. Our algorithms adaptively predict the next mode-switch moment by considering both historical and future event arrivals, and procrastinate the buffered and future events as late as possible. By adopting the worst-case interval-based abstraction in Real-Time Calculus, our algorithms can not only tackle arbitrary event arrivals, e.g. with burstiness, but also guarantee hard real-time requirements with respect to both timing and backlog constraints. To handle multiple event streams, we propose solutions for two preemptive scheduling, i.e., earliest-deadline-first (EDF) and fixed priority (FP) policies, as well as two backlog allocation schemes, i.e., individual and global backlog allocation.

The rest of this article is organized as follows: In the next section, related work in the literature is reviewed. Section 3 provides system models and problem definition. Section 4 presents a set of new routines that will be used throughout the article. Section 5 presents our algorithms for single event stream, while Sect. 6 provides solutions for multiple event streams. Simulation results are presented in Sect. 7 and Sect. 9 concludes the article.

2 Related work

The dynamic voltage scaling (DVS) technique was introduced to reduce the dynamic energy consumption by trading the performance for energy savings. For DVS processors, a higher supply voltage, generally, leads to not only a higher execution speed/frequency but also higher power consumption. As a result, DVS scheduling algorithms, e.g., Yao et al. (1995), Zhang et al. (2002), Aydin et al. (2001), tend to execute events as slowly as possible, without any violation of timing constraints. On the other hand, dynamic power management (DPM) with clock gating or voltage gating can be applied to change the device power mode, e.g., to a sleep mode, to consume less (static/leakage) power. For devices with the sleep mode, Baptiste (2006) proposes an algorithm based on dynamic programming to control when to turn on/off a device for aperiodic real-time events with the same execution time. For multiple low-power modes, Augustine et al. (2004) determine the mode that a processor should enter for aperiodic real-time events and propose a competitive algorithm for online use. Swaminathan and Chakrabarty (2005) explore dynamic power management of real-time events in controlling shutting down and waking up system devices for energy efficiency. To aggregate the idle time for energy reduction, Shrivastava et al. (2005) propose a framework for code transformations.

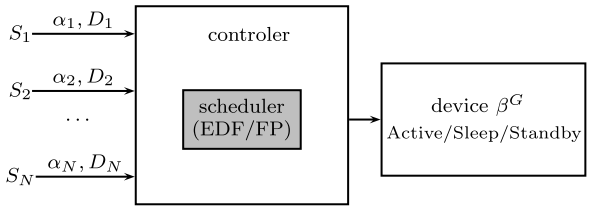
There also exists work that considers platforms with both DPM and DVS. Chen and Kuo (2007) propose to execute tasks at a certain speed (mostly at the critical speed) and to control the procrastination of real-time events. By turning the device to the sleep mode, the execution of the procrastinated real-time events is aggregated in a busy interval to reduce energy consumption. Heo et al. (2007) explore how to integrate different power management policies in a server farm. Alternatively, Devadas and Aydin (2008, 2010) consider the interplay between DVS and DPM for a system with a DVS-capable processor and multiple devices. Based on the concept of device forbidden regions, the authors propose algorithms to determine the optimal processor speed as well as transition decisions of device states to minimize overall system energy for periodic real-time tasks.

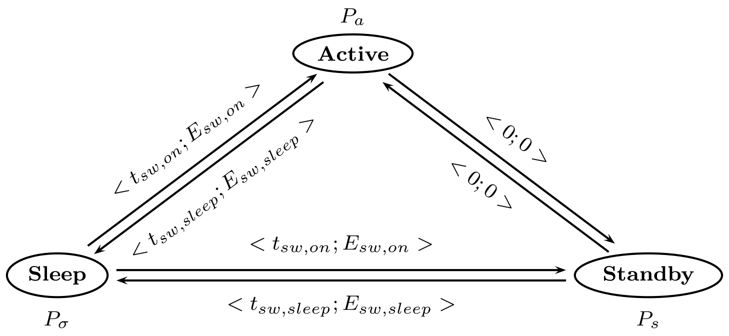
Most of the above approaches require either precise information of event arrivals, such as periodic real-time events (Chen and Kuo 2007), or aperiodic real-time events with known arrival time (Baptiste 2006; Augustine et al. 2004; Irani et al. 2003). However, in practice, the precise timing information of event arrivals might not be known in advance since the arrival time depends on many factors. When the precise timing of event arrivals is unknown, to our best knowledge, the only known approaches are to apply the online algorithms proposed by Irani et al. (2003) and Augustine et al. (2004) to control when to turn on the device. However, since the online algorithms in Augustine et al. (2004), Irani et al. (2003) greedily stay 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. Such algorithms are not applicable for hard real-time systems.

To model such irregular events, Real-Time Calculus, extended from Network Calculus (Cruz 1991), was proposed by Thiele et al. (2000), Wandeler et al. (2006) to characterize events with arrival curves. The arrival curve of an event stream describes the upper and lower bounds of the number of events arriving to the system for a specified interval. Therefore, schedulability analysis can be done based on the arrival curves of event streams. Maxiaguine et al. (2005) apply Real-Time Calculus within the DVS context and compute a safe frequency at periodical intervals with predefined length to prevent buffer overflow of a system. Chen et al. (2009) explore the schedulability for on-line DVS scheduling algorithms when the event arrivals are constrained by a given upper arrival curve. In contrast to these closest approaches, we focus on dynamic power management. In Huang et al. (2009), we propose offline algorithms to find periodic time-driven patterns to turn on/off devices for energy saving without sacrificing timing guarantees. The light run-time overhead of the periodic schemes in Huang et al. (2009) is very suitable for embedded systems that only have limited power on computation. Alternatively, we propose online algorithms in this article. The on-off decisions are dynamic and adaptively vary according to the actual arrivals of events. Furthermore, we provide solutions on multiple event-stream scenarios where event streams with different characteristics can be tackled with both earliest-deadline-first and fixed-priority scheduling policies.

Notes that researches (Agarwal et al. 2006; Roy et al. 2003; Stephan 2007) have shown that the static power of a CMOS circuit is depended on the temperature of the circuit and there are also work (Yang et al. 2010; Bao et al. 2010) on the saving of temperature-aware static power. In this work, we focus on I/O peripheral devices which are less affected by temperature. Therefore, we do not consider the temperature impact on the static power.

Fig. 1 The abstract system model of the studied problem





P_a
Active
Sleep
P_σ
Standby
P_s
$< t_{sw,on}; E_{sw,on} >$
$< t_{sw,sleep}; E_{sw,sleep} >$
$< 0; 0 >$
$< 0; 0 >$
$< t_{sw,on}; E_{sw,on} >$
$< t_{sw,sleep}; E_{sw,sleep} >$

Fig. 2 The state transit of a device, where the 2-tuple on each transit defines timing and energy overheads

3 System models and problem definition

The system we consider consists of a device and a controller. The device is responsible for event processing. The controller conducts three tasks: a) handles event arrivals, e.g., storing unprocessed events into system backlog, b) controls the power mode of the device to serve the arrived events, e.g., turning on and off the device, and c) dispatches events in the system backlog to the device with certain scheduling policy. An abstract model of our system is shown in Fig. 1, where the controller and the device could be, for instance, an operating system and an I/O peripheral device, respectively. Parameters α, D, and β^G in Fig. 1 will be introduced later on.

The device has three power modes, namely `active`, `standby`, and `sleep` modes. The power consumption in `sleep` mode is P_σ. To serve an event, the device must be in `active` mode with power consumption P_a, in which $P_a > P_\sigma$. Once there is no event to serve, the device can enter `sleep` mode. However, switching from the sleep mode to `active` mode and back takes time, denoted by $t_{sw,on}$ and $t_{sw,sleep}$, and incurs energy overhead, denoted by $E_{sw,on}$ and $E_{sw,sleep}$, respectively. To prevent the device from frequent mode switches, the device can also stay in the standby mode. The power consumption P_s in the standby mode, by definition, is no more than P_a and is more than P_σ. We assume that switching between `standby` mode and `active` mode has negligible overhead, the same assumption as in Zhuo and Chakrabarti (2005), Yang et al. (2007). Figure 2 illustrates the state chart of our power model.

Several event streams are given as inputs of the system. We denote $\mathcal{S} = \{S_1, \ldots, S_N\}$ as a stream set containing N event streams with different characteristics. To buffer incoming events of streams in $\mathcal{S}$, the controller maintains a backlog

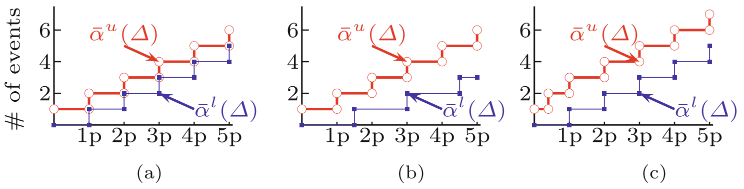


Fig. 3 Examples for arrival curves for: (**a**) periodic events with period p, (**b**) events with minimal inter-arrival distance p and maximal inter-arrival distance $p' = 1.5p$, and (**c**) events with period p, jitter $j = p$, and minimal inter-arrival distance $d = 0.4p$

of unprocessed events. Buffering more events than the size of the backlog incurs a backlog overflow and causes a controller failure. To buffer unprocessed events of different event streams, the system buffer can be shared by all event streams to efficiently utilize the system buffer or split into independent buffers, each of which is private for an event stream, to reduce runtime overhead. Therefore, we discuss two types of backlog managements: a) *global backlog* where all event streams in $\mathcal{S}$ share one buffer, and b) *individual backlog* where each event stream has its private buffer. The size of the buffer in either case is assumed given. How to decide a proper size of the system buffer is a different research problem and thus is not considered in this article.

3.1 Worst-case interval-based streaming model

To model irregular arrival of events, we adopt arrival curves $\bar{\alpha}(\Delta) = [\bar{\alpha}^u(\Delta), \bar{\alpha}^l(\Delta)]$ from Real-Time Calculus (Thiele et al. 2000), in which $\bar{\alpha}_i^u(\Delta)$ and $\bar{\alpha}_i^l(\Delta)$ are the upper and lower bounds on the number of arrival events for a stream S_i in any time interval of length Δ, respectively. The concept of arrival curves unifies many traditional timing models of real-time event streams such as sporadic, periodic, periodic with jitter, or any other arrival pattern. For example, a periodic event stream can be modeled by step functions $\bar{\alpha}^u(\Delta) = \lfloor \frac{\Delta}{p} \rfloor + 1$ and $\bar{\alpha}^l(\Delta) = \lfloor \frac{\Delta}{p} \rfloor$. For a sporadic event stream with minimal inter arrival distance p and maximal inter arrival distance p', the upper and lower arrival curve is $\bar{\alpha}^u(\Delta) = \lfloor \frac{\Delta}{p} \rfloor + 1$, $\bar{\alpha}^l(\Delta) = \lfloor \frac{\Delta}{p'} \rfloor$, respectively. Moreover, for an event stream with period p, jitter j, and minimal inter arrival distance d, the upper arrival curve is $\bar{\alpha}^u(\Delta) = \min\{\lceil \frac{\Delta+j}{p} \rceil, \lceil \frac{\Delta}{d} \rceil\}$. Figure 3 illustrates the arrival curves for the preceding cases. To be complete, we also assume $\alpha(\Delta) = 0$ for $\Delta < 0$.

Analogous to arrival curves that provide an abstract event stream model, a 2-tuple $\beta(\Delta) = [\beta^u(\Delta), \beta^l(\Delta)]$ of upper and lower service curves then provides an abstract resource model. The upper and lower service curve provides an upper and lower bound on the available resources in any time interval of length Δ. Further details are referred to Thiele et al. (2000).

Note that an arrival curve $\bar{\alpha}_i(\Delta)$ specifies the number of events of stream S_i whereas a service curve $\beta(\Delta)$ specifies the available amount of time for execution, for interval length Δ. Therefore, $\bar{\alpha}_i(\Delta)$ has to be transformed to $\alpha_i(\Delta)$ to indicate the amount of computation time required for the arrived events in intervals. Suppose that the execution time of any event in stream S_i is w_i. The transformation can be done by $\alpha_i^u = w_i \bar{\alpha}_i^u$, $\alpha_i^l = w_i \bar{\alpha}_i^l$ and back by $\bar{\alpha}_i^u = \alpha^u / w_i$, $\bar{\alpha}_i^l = \alpha^l / w_i$ thereof. For variable workloads in an event stream, our algorithms can be revised slightly by adopting the workload curves in Maxiaguine et al. (2004). Moreover, to satisfy the real-time constraint, the response time of an event in event stream S_i must be no more than its specified relative deadline D_i, where the response time of an event is its finishing time minus the arrival time of the event. On the arrival of an event $e_{i,j}$ of stream S_i at time t, the absolute deadline is $t + D_i$.

3.2 Worst-case scheduling analysis using arrival curves

Using this interval-based model, one can perform worst-case scheduling analysis by means of Real-Time Calculus (Maxiaguine et al. 2005) and Real-Time Interface (Roy et al. 2003). Within this context, a device is said to provide guaranteed output service $\beta^G(\Delta)$. For instance, if the device offers computation time Δ in any time interval of length Δ, then $\beta^G(\Delta) = \Delta$. Suppose now that an event stream S_i needs service curve $\beta^A(\Delta)$, denoted as service demand, to operate correctly, i.e., to satisfy responsiveness constraints. The device is schedulable if and only if the condition

$$\beta^G(\Delta) \geq \beta^A(\Delta), \quad \forall \Delta \geq 0 \tag{1}$$

holds.

For example, if the relative deadline D_i of each event in stream S_i is given, one can say the device has to provide at least $\beta^A(\Delta) = \alpha_i(\Delta - D_i)$ service to prevent deadline misses of events in S_i. In the rest of this article, we show how to properly construct $\beta^G(\Delta)$ and $\beta^A(\Delta)$ for a given system to not only reduce static power consumption but also guarantee hard real-time properties.

3.3 Problem definition

This article explores how to effectively minimize the energy consumption with dynamic power management schemes when serving a stream set $\mathcal{S}$ of N event streams. Intuitively, energy saving can be obtained by turning the device to `sleep` mode when no event to process and staying at `sleep` mode as no event arrives. However, switching from/to `sleep` mode incurs both timing and energy overheads. Similar to Cheng and Goddard (2006), we define a break-even time to model these overheads.

Definition 1 (Break Even Time) Suppose that switching the device to `sleep` mode takes $t_{sw,sleep}$ time and switching back to `active` mode takes $t_{sw,on}$ time, the corresponding energy consumption for the switching activities are $E_{sw,sleep}$ and $E_{sw,on}$, and the static power is $P_s - P_\delta$. The break-even time is defined as:

$$T_{BET} \overset{\text{def}}{=} \max\left\{ t_{sw,on} + t_{sw,sleep}, \frac{E_{sw,on} + E_{sw,sleep}}{P_s - P_\delta} \right\} \tag{2}$$

Consider the case that the device is switched to `sleep` mode. If the interval that the device can stay in `sleep` mode is shorter than T_{BET}, the mode-switch overheads are larger than the energy saving. Therefore, such mode switch is not worthwhile.

On the other hand, retaining the device longer in `sleep` mode might incur backlog overflow or deadline misses for burst event arrivals in the near future. The question is what is a proper time to conduct a mode switch of the device.

We say that a scheduling decision is to decide when to perform a mode-switch of the device. A scheduling decision is *feasible* if it is always possible to meet the timing and backlog constraints for any event trace constrained by an arrival curve. An algorithm is *feasible* if it always generates feasible scheduling decisions. Therefore, the problem studied in this article is to decide a feasible schedule for a) *when to turn the device to* `sleep` *mode to reduce the static power*, and b) *when to turn the device from* `sleep` *mode to* `active` *mode to serve events*.

4 Real-time calculus routines

In this section, we present basic routines to construct service guarantee and demand.

4.1 Bounded delay

A service curve $\beta(\Delta)$ can be constructed as a bounded delay function.

Definition 2 (Bounded Delay Service Curve (Le Boudec 1998)) A bounded delay service curve is defined as no service provided for at most τ units of time:

$$\boldsymbol{bdf}(\Delta, \tau) \overset{\text{def}}{=} \max\big\{0, (\Delta - \tau)\big\}, \quad \forall \Delta \geq 0 \tag{3}$$

We say that a *sleep interval* is the time for which the device retains in `sleep` mode. The τ in Definition 4 can be considered as such a sleep interval. In the case that the device provides full service, τ is 0, i.e., the device never switches to `sleep` mode.

Definition 3 (Longest Feasible Sleep Interval) The longest feasible sleep interval τ^* with respect to a given service demand $\beta^A(\Delta)$ is thereby defined as:

$$\tau^* = \max\big\{\tau : \boldsymbol{bdf}(\Delta, \tau) \geq \beta^A(\Delta),\ \forall \Delta \geq 0\big\} \tag{4}$$

Definition 4 (Deadline Service Demand (Thiele et al. 2006)) Suppose an event stream S_i with relative deadline D_i. To satisfy the required relative deadline D_i, the minimum service demand of stream S_i is

$$\beta^\flat(\Delta) \overset{\text{def}}{=} \alpha_i^u(\Delta - D_i) \tag{5}$$

Definition 5 (Backlog-size Service Demand) Suppose a system has a backlog of size Q_i to buffer un-processed events for stream S_i. To prevent backlog overflow, the minimum service demand of stream S_i is

$$\beta^\dagger(\Delta) \overset{\text{def}}{=} \big\{\alpha_i^u(\Delta) - w_i \cdot Q_i\big\}^+ \tag{6}$$

where $\{a\}^+ = \max\{a, 0\}$.

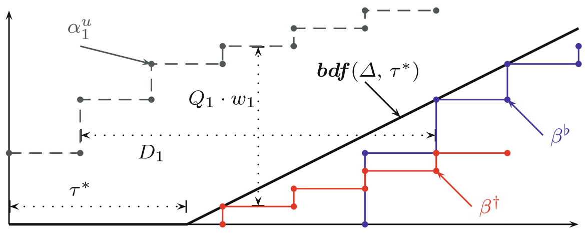


Fig. 4 An example for the bounded delay function for event stream S_1 with upper arrival curve $\alpha_1^u(\Delta)$. Only part of the arrival curve is presented for simplicity

Combining both deadline and backlog service demands, the τ^* in (4) can be refined as

$$\tau^* = \max\left\{\tau : \boldsymbol{bdf}(\Delta, \tau) \geq \max\{\beta^\flat(\Delta), \beta^\dagger(\Delta)\}\right\} \tag{7}$$

Figure 4 depicts an example for above constructions for single event stream. Based on these constructions, we state the following lemma.

Lemma 1 *Given τ^* computed from (7) which is larger than T_{BET}, at any time instant t when the device is active and no event to process, it is feasible to switch the device to* `sleep` *mode for $[t, t + \tau^*)$ interval without violating the deadline and backlog-size requirements of for any event in stream S_i.*

Proof The service demand $\beta^\flat$ in (5) is constructed as horizontally right shifting α_i^u with D_i distance, which represents the tightest bound that guarantees the deadline constraint. Similarly, the service demand $\beta^\dagger$ in (6) obtained by vertically shifting α_i^u down by $w_i \cdot Q$ distance defines the tightest bound that prevents backlog overflow. The bounded delay function $\boldsymbol{bdf}(\Delta, \tau^*)$ constructs a β^G that bounds both $\beta^\flat$ and $\beta^\dagger$, thus fulfills (1). □

4.2 Future prediction with historical information

We keep track of event arrivals in the past as a history. Because the total amount of arrived events in any time interval is constrained by the corresponding arrival curves, one can predict future event arrivals based on a certain length of historical information of event arrivals in the recent past. If a burstiness has been observed recently, for instance, it can be foreseen that sparse events will arrive in the near future. To make use of historical information, we define a history curve.

Definition 6 (History Curve) Suppose t is the current time and $R_i(t)$ is the accumulated number of events of stream S_i in interval $[0, t)$. The length of the history window of which controller can maintain is Δ^h, i.e., historical information for only Δ^h time units is recorded. At time t, a history curve for stream S_i is define as

$$H_i(\Delta, t) \overset{\text{def}}{=} \begin{cases} R_i(t) - R_i(t - \Delta), & \text{if } \Delta \leq \Delta^h, \\ R_i(t) - R_i(t - \Delta^h), & \text{otherwise} \end{cases} \tag{8}$$

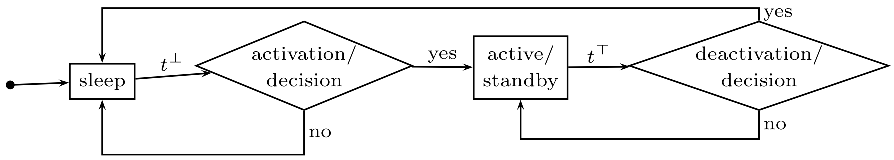


Fig. 5 The control flow of our approach

The maximal future event arrivals in the near future from time t to $t + \Delta$, denoted as $\bar{\alpha}_i^u(\Delta, t)$, is

$$\bar{\alpha}_i^u(\Delta, t) = \inf_{\lambda \geq 0} \left\{ \bar{\alpha}_i^u(\Delta + \lambda) - H_i(\lambda, t) \right\} \tag{9}$$

We use (9) to predict future event arrivals for any time instant.

4.3 Backlogged demand

Analogously, we can preciously define the service demand of those backlogged events. We denote the set of unfinished events of S_i in the backlog at time t as $\boldsymbol{E_i}(t)$. Note that although the absolute deadline $D_{i,j}$ for event $e_{i,j} \in \boldsymbol{E_i}(t)$ does not change, the relative deadline is not D_i anymore. It varies according to relative distance from t.

Definition 7 (Backlogged Demand) Suppose that events in $\boldsymbol{E_i}(t)$ are indexed as $e_{i,1}, e_{i,2}, \ldots, e_{i,|\boldsymbol{E_i}(t)|}$ from the earliest deadline to the latest. A backlog demand curve for stream S_i at time t is defined as

$$B_i(\Delta, t) \overset{\text{def}}{=} w_i \cdot \begin{cases} (j-1), & D_{i,j} - t < \Delta \leq D_{i,j+1} - t, \\ |\boldsymbol{E_i}(t)|, & \Delta > D_{i,|\boldsymbol{E_i}(t)|} - t, \end{cases} \tag{10}$$

in which $D_{i,0}$ is defined as t for brevity.

5 Basic algorithms for single stream

In general, we have to decide when to turn the device from `sleep` mode to `active` mode and turn it back to sleep mode in order to reduce static power. Therefore, we have to deal with *deactivation scheduling decisions* and *activation scheduling decisions* to switch safely and effectively. An overview of our approach is illustrated in Fig. 5.

For deactivation scheduling decisions, when the device is in `active` mode and there is no event in the backlog, we have to decide whether the device has to switch to `sleep` mode instantly or it should remain active/standby for a while for serving the next incoming event (we assume the device switches between `active` and `standby` modes automatically). For brevity, for the rest of this article, time instants for deactivation scheduling decisions are denoted by $t^{\top}$.

After the device is switched to `sleep` mode, it has to be switched active again for event processing. The activation scheduling decision is evaluated at the time instant upon the arrival of an event or expiration of the sleep interval that the controller previously set. A scheduler has to decide whether the device has to change to `active` mode instantly to serve events, or it should remain in `sleep` mode for a while to aggregate more events and prevent unnecessary mode switches. For brevity, time instants for activation scheduling decisions are denoted by $t^{\perp}$ for the rest of this article.

In this section, we present our approach to minimize static power. The assumption of our approach is that both deadline and backlog requirements can be guaranteed if the device always provides full service, i.e., the device never turns to `sleep` mode. For simplicity, we consider only single event stream case, says S_1. We first present how to deal with deactivation of scheduling decisions and then propose two methods for the scheduling decisions of the activation. The solutions to multiple event streams are presented in the subsequent two sections.

5.1 Our deactivation algorithm

The History-Aware Deactivation (HAD) algorithm analyzes whether the device should be turned to `sleep` mode from `active` mode. The principle is to switch the device only when energy saving is possible. One obvious fact is that as long as there are events in the system backlog, the device can be kept busy in `active` mode and no energy overhead for mode switching is introduced. In order to reduce static power, the deactivation decision thereby makes sense only when the device is in `active` mode while there is no new arrival of any event as well as no event in system backlog. Suppose that $t^{\top}$ is such a time instant.

Turning the device instantly at time $t^{\top}$ to `sleep` mode, however, does not always help still. The reason is that we pay overheads for mode switches. In the case there are events arriving in the very near future, the device has to be switched active again to process these events. If the energy saving obtained from a short sleep interval cannot counteract the switching overheads, this mode switch only introduces additional energy consumption. Therefore, the idea is firstly to compute the maximal possible sleep interval τ^* and check whether this τ^* is sufficient to cover the break-even time. Specifically, we calculate the arrival curve $\hat{\alpha}_1^u(\Delta, t^{\top})$ at time $t^{\top}$ by (9) and refine the service demands in (5) and (6) as

$$\beta^{\flat}(\Delta) = \alpha_1^u(\Delta - D_1, t^{\top}) \tag{11}$$

$$\beta^{\dagger}(\Delta) = \left\{\alpha_1^u(\Delta, t^{\top}) - Q_1 \cdot w_1\right\}^+ \tag{12}$$

By applying (11) and (12) to (7), the maximal sleep interval τ^* is computed. If τ^* is larger than T_{BET}, the device is switched to `sleep` mode at time $t^{\top}$. Otherwise, the device is retained in `active` mode. The pseudo code of the algorithm is depicted in Algorithm 1.

The algorithm leads to the following theorem:

Theorem 1 *Algorithm HAD guarantees a feasible scheduling upon a deactivation decision at any time $t^{\top}$ for single event-stream system, if the device provides full service from time $t^{\top} + \tau^*$ on, where τ^* is computed from Line* 1 *in the algorithm.*

Algorithm 1 HAD deactivation

```
procedure at time instant $t_w^{\perp}$:
1: compute $\tau^*$ of (7) by $\beta^{\flat}$ and $\beta^{\dagger}$ in (11) and (12);
2: if $\tau^* > T_{BET}$ then
3:     deactivate the device;
4: end if
```

Proof The theorem states that at any time instant $t^{\top}$ at which Algorithm HAD decides to deactivate the device, the latest activation time to prevent constraint violations is $t^{\top} + \tau^*$. We prove this theorem by contradiction. Suppose at time $t^{\top} + \lambda$, the deadline of an event which comes within the interval $[t^{\top}, t^{\top} + \lambda)$ is missed. We denote the number of events arrived within this interval as u. Because of the deadline missing, the service demand $u \cdot w_1$ in this interval is larger than our constructed service supply $\boldsymbol{bdf}(\lambda, \tau^*)$ which actually bounds the service demand of the maximum number of events that can arrival, i.e., $w_1 \cdot \bar{\alpha}_1^u(\lambda, t^{\top})$. The inequality $u > \bar{\alpha}_1^u(\lambda, t^{\top})$ contradicts to the definition in (9). Therefore, the theorem holds. □

5.2 Our activation algorithms

Once the device is in `sleep` mode, the controller needs to switch it back to `active` mode at a later moment for event processing. How to decide the actual switch moment needs more consideration. On the one hand, it is preferable to aggregate as many events as possible for each switch operation to not only reduce the standby period but also minimize the number of switch operations. On the other hand, the real-time constraints of the aggregated and future events need to be guaranteed. In addition, a polling mechanism is not desirable which will overload the controller. In this section, we present two algorithms, namely worst-case-greedy (WCG) algorithm and event-driven-greedy (EDG) algorithm, for the activation scheduling decisions. The differences between these two algorithms are in the following:

- Algorithm WCG is time-triggered. It conservatively assumes worst-case event arrivals and predicts the earliest switch moment. If the worst case does not occur when the predicted time comes, a new prediction is conducted and the switch decision is deferred to a later moment.
- Unlike WCG, Algorithm EDG works in an event-triggered manner. It optimistically considers the least event arrivals and predicts the latest time for mode switches. Upon the new arrival of an event before the predicted time, the decision is reevaluated and it is shifted earlier if necessary.

Therefore, the time instant $t^{\perp}$ for the evaluation of the activation scheduling decisions can be at either event arrivals or the predicted activation time. We identify these two cases by *event arrival* and *wake-up alarm arrival*, designated as $t_e^{\perp}$ and $t_w^{\perp}$, respectively.

5.2.1 Worst-case-greedy (WCG) activation

Algorithm WCG works in a time-triggered manner. It reacts to each wake-up alarm and performs two tasks: a) check whether the device has to be switched to `active` mode for the current alarm; b) if not, determine a new wake-up alarm. In the case that worst-case burst happens before the wake-up alarm $t_w^\perp$, i.e., our previous prediction is correct, the mode switch has to be carried on at the current wake-up alarm. If the actual arrivals of events are less than the worst case, switching the device at the current $t_w^\perp$ will reserve too much service than actual needs. The device can stay in `sleep` mode for a longer period.

To evaluate the activation decision and predict the next wake-up alarm, we again apply the bounded-delay function (7) to find a next possible sleep interval. To obtain tight results, the deadline and backlog service demand in (5) and (6) can be improved. At the current wake-up alarm $t_w^\perp$, the deadline service demand $\beta^\flat$ includes those events that are already stored in the system backlog, i.e., $B_1(\Delta, t_w^\perp)$ defined in (10), besides the history-refined worst-case event arrival $\alpha_1^u(\Delta - D_1, t_w^\perp)$. Similarly, the current size of the available backlog is the original size subtracted by the number of backlogged events, i.e., $|\boldsymbol{E}(t^\perp)|$ defined in Sect. 4.3. The deadline service demand $\beta^\flat$ and the backlog-size service demand $\beta^\dagger$ are thus refined as

$$\beta^\flat(\Delta) = \alpha_1^u(\Delta - D_1, t_w^\perp) + w_1 \cdot B_1(\Delta, t_w^\perp) \tag{13}$$

$$\beta^\dagger(\Delta) = \left\{\alpha_1^u(\Delta, t_w^\perp) - \left(Q_1 - |\boldsymbol{E}(t^\perp)|\right) \cdot w_1\right\}^+ \tag{14}$$

Using (13) and (14), the next sleep interval τ^* is computed. If $\tau^* > 0$, the next wake-up alarm is set to time $t_w^\perp + \tau^*$. Otherwise, the device is switched to `active` mode. The pseudo code of Algorithm WCG is depicted in Algorithm 2.

Algorithm 2 WCG activation

```
procedure event arrival at time $t_e^\perp$:
1: do nothing;
procedure wake-up alarm arrival at time $t_w^\perp$:
1: compute $\tau^*$ of (7) with $\beta^\flat$ and $\beta^\dagger$ by (13) and (14);
2: if $\tau^* > 0$ then
3:     new wake-up alarm at time $t_w^\perp \leftarrow t_w^\perp + \tau^*$;
4: else
5:     wakeup the device;
6: end if
```

The constructed $\beta^\flat$ in (13) bounds the future arrival demands from $t_w^\perp$ on and $\beta^\dagger$ in (14) guarantees the backlog constraint, leading to following theorem:

Theorem 2 *Algorithm WCG guarantees a feasible scheduling upon an activation decision at any wake-up alarm $t_w^\perp$ for single event-stream system, if the device provides full service from time $t_w^\perp$ on.*

We omit the proof due to the similarity to Theorem 1. Algorithm WCG is effective in the sense that it greedily extends the sleep period as long as a scheduling decision is feasible. It is efficient as well when actual event arrivals are close to the worst case, where the reevaluation of the wake-up alarm does not take place often. Furthermore, the number of reevaluation is bounded by $\alpha_1^u(D_1 - w_1)$.

The last question is where the first wake-up alarm is. There are two possibilities: a) at the time the first arrived event after the device is deactivated, and b) at the deactivation time instant $t^\top + \tau^*$ (τ^* is computed by (11) and (12) in the HAD algorithm). Both approaches are effective. For consistency, we adopt the second approach, such that Algorithm WCG is purely time-driven.

5.2.2 Event-driven-greedy (EDG) activation

In contrast to Algorithm WCG that predicts the earliest wake-up alarm $t_w^\perp$, Algorithm EDG predicts the latest one. It computes the latest moment by assuming the least event arrivals in the near future. Unlike Algorithm WCG where the evaluation of the activation scheduling decisions takes place upon each wake-up alarm arrives, the decision here is refined upon event arrivals.

At time $t_{e_{1,i}}^\perp$ at which event $e_{1,i}$ arrives, when is the corresponding latest wake-up alarm $t_w^\perp$ is not that obvious. One intuitive guess is $t_{e_{1,i}}^\perp + D_1 - w_1$. This time instant is however too optimistic except for the first event $e_{1,1}$ after the device is deactivated. Our EDG algorithm works in the following manner. For the first arrived event $e_{1,1}$, the wake-up alarm is set to $t_{e_{1,1}}^\perp + D_1 - w_1$. For any subsequent event $e_{1,i}$, wake-up alarm is set to the smaller value of the previous $t_w^\perp$ and $t_w^\perp - (w_1 - (t_{e_{1,i}}^\perp - t_{e_{1,i-1}}^\perp))$. This new $t_w^\perp$ is not yet always a feasible activation time instant. If τ^* computed from this time instant is not larger than 0, the activation is set to an earlier time, i.e., the earliest activation time as if worst-case event arrival happen at $t_{e_{1,1}}^\perp$.

For an event $e_{1,i}$ arrived at time $t_{e_{1,i}}^\perp$, the service demand for the newly computed wake-up alarm $t_w^\perp$ includes a) the possible burst from $t_w^\perp$ on, which is bounded by $\bar{\alpha}_1^u(\Delta, t_w^\perp)$, b) the backlog until $t_w^\perp$, and c) the estimated least event arrival between $[t_{e_{1,i}}^\perp, t_w^\perp)$, constrained by $\bar{\alpha}_1^l(\Delta)$. To compute a precise $\bar{\alpha}_1^u(\Delta, t_w^\perp)$, we first revise the historical information $H_1(\Delta, t_w^\perp)$ by advancing the time from $t_{e_{1,i}}^\perp$ to $t_w^\perp$ to include those events which definitely have to come between $[t_{e_{1,i}}^\perp, t_w^\perp)$. We denote such a trace as $H'(\Delta, t_w^\perp)$:

$$H_1'(\Delta, t_w^\perp) = \begin{cases} \bar{\alpha}_1^l(\epsilon) - \bar{\alpha}_1^l(\epsilon - \Delta), & \text{if } \Delta < \epsilon, \\ H_1(\Delta, t_{e_{1,i}}^\perp) + \bar{\alpha}_1^l(\epsilon), & \text{if } \epsilon < \Delta < \Delta^h - \epsilon, \\ H_1(\Delta^h - \epsilon, t_{e_{1,i}}^\perp) + \bar{\alpha}_1^l(\epsilon), & \text{otherwise,} \end{cases} \tag{15}$$

where $\epsilon = t_w^\perp - t_{e_{1,i}}^\perp$ for abbreviation. The curve H_1' can be considered as the concatenation of the historical information H_1 until $t_{e_{1,i}}^\perp$ and the time inversion of $\bar{\alpha}_1^l$ in the interval $[0, \epsilon)$. The worst-case arrival curve after time $t_w^\perp$ with the new historical information H_1' is

$$\bar{\alpha}_1^u(\Delta, t_w^\perp) = \inf_{\lambda \geq 0} \left\{ \bar{\alpha}_1^u(\Delta + \lambda) - H_1'(\lambda, t_w^\perp) \right\} \tag{16}$$

and $\alpha_1^u(\Delta, t_w^\perp) = w_1 \cdot \bar{\alpha}_1^u(\Delta, t_w^\perp)$.

The corresponding backlog demand curve that encapsulates the estimated least arrival events within the interval $[t^{\perp}_{e_{1,i}}, t^{\perp}_{w})$ is

$$B'_1(\Delta, t^{\perp}_w) = \begin{cases} j-1, & D_{1,j} - t^{\perp}_w < \Delta \leq D_{1,j+1} - t^{\perp}_w; \\ \mathcal{E}, & \Delta > D_{1,\mathcal{E}} - t^{\perp}_w, \end{cases} \tag{17}$$

where $\mathcal{E} = |\boldsymbol{E}_{\mathbf{1}}(t^{\perp}_{e_{1,i}})| + \bar{\alpha}^l_1(\epsilon)$, $\epsilon = t^{\perp}_w - t^{\perp}_{e_{1,i}}$, and $D_{1,0}$ is defined as $t^{\perp}_{e_{1,i}}$.

With the refined historical information and backlog demand, the two service demands $\beta^{\flat}$ and $\beta^{\dagger}$ can be refined as

$$\beta^{\flat}(\Delta) = \alpha^u_1(\Delta - D_1, t^{\perp}_w) + w_1 \cdot B'_1(\Delta, t^{\perp}_w) \tag{18}$$

$$\beta^{\dagger}(\Delta) = \left\{\alpha^u_1(\Delta, t^{\perp}_w) - \left(Q_1 - |\boldsymbol{E}(t^{\perp})| - \bar{\alpha}^l_1(\epsilon)\right) \cdot w_1\right\}^+ \tag{19}$$

By applying (18) and (19), the sleep interval τ^* in (7) is computed for event $e_{1,i}$. If $\tau^* > 0$, the wakeup alarm is valid. Otherwise, the new wakeup alarm is set to an earlier moment. The pseudo code of the algorithm is depicted in Algorithm 3.

Algorithm 3 EDG activation

```
procedure event arrival at time $t^{\perp}_{e_{1,i}}$:
  1: $t^{\perp}_w \leftarrow (t^{\perp}_{e_{1,i}} - t^{\perp}_{e_{1,i-1}} > w_1) ? t^{\perp}_w : t^{\perp}_w - (w_1 - (t^{\perp}_{e_{1,i}} - t^{\perp}_{e_{1,i-1}})$
  2: calculate $\tau^*$ at time $t^{\perp}_w$ by (15–19);
  3: if $\tau^* \leq 0$ then
  4:     $t^{\perp}_w \leftarrow t^{\perp}_{e_{1,1}} + \tau^{\perp}$, where $\tau^{\perp}$ computed from (5)–(7)
  5: end if
procedure wake-up alarm arrival at time $t^{\perp}_w$:
  1: activate the device;
```

Theorem 3 *Algorithm EDG guarantees a feasible scheduling upon an activation decision at any wakeup alarm $t^{\perp}_w$ for single event-stream system, if the device provides full service from time $t^{\perp}_w$ on.*

Proof We differentiate the mode switch decisions into two cases: decisions by hitting Line 4 of the algorithm or without. In the case of without hitting, the feasibility of the decision is guaranteed by (15)–(19) where the events actually arrived before time $t^{\perp}_w$ and the potential burstiness after are considered in each evaluation. In the case of hitting, we need to prove the arrival time $t^{\perp}_{e_{1,i}}$ of event $e_{1,i}$ is always earlier than $t^{\perp}_{e_{1,1}} + \tau^{\perp}$. For time instant $t^{\perp}_{e_{1,1}}$ at which $e_{1,1}$ arrives, the maximum number of events can be stored in the system backlog is $\min\{Q_1, \bar{\alpha}^u_1(\tau^{\perp}) - 1\}$, denoted as u. Based on the construction in Line 1, Line 4 hits only if the number of arrived events reaches this maximum. According to the subadditivity of an upper arrival curve and the linearity of the bounded delay service curve, the time interval to generate u events is bounded by $\tau^{\perp}$, i.e., the inequality $(\bar{\alpha}^u)^{-1}(\tau^{\perp}) > u$ always holds[1]. Therefore, $t^{\perp}_{e_{1,1}} + \tau^{\perp} > t^{\perp}_{e_{1,i}}$. Because $t^{\perp}_{e_{1,1}} + \tau^{\perp}$ assumes the worst-case event arrivals happen at the time instant of the arrival of the first event after the device is switched to `sleep` mode, switching the processing core to `active` mode at this time always results in a feasible scheduling decision. Since the scheduling decisions are feasible for both cases, the theorem holds. □

The activation decision in Line 4 is pessimistic. It is possible to refine $t^{\perp}_w$ when $\tau^* \leq 0$ happens, instead of setting the prediction back to $t^{\perp}_{e_{1,1}} + \tau^{\perp}$. However, such refinement demands more computation. Algorithm EDG is designed for scenarios where events come sparsely and the worst case seldom occurs. In such cases, the algorithm is effective because Line 4 will seldom be hit. The algorithm is efficient as well. The theoretical upper bound of the number of reevaluations is $\min\{Q_1, \bar{\alpha}^u_1(\tau^{\perp})\}$. In practices, the number of reevaluations is approximately equal to the number of events actually arrived. Furthermore, $\tau^{\perp}$ can be computed offline as it is a constant given the specification of the stream.

6 Our algorithms for multiple streams

To tackle multiple event streams, an essential problem is to compute the total service demand for a stream set $\mathcal{S}$. The total service demand for $\mathcal{S}$ depends on not only the service demand of individual streams but also the scheduling policy as well as the system backlog organization. The scheduling in this section refers to the policy by which events of different streams in the system backlog are chosen to feed the device. In this work, we consider two preemptive scheduling, i.e., earliest-deadline-first (EDF) and fixed-priority (FP) policies, as they are most commonly used policies in the real-time systems. With respect to the backlog organization, two different schemes, i.e., individual and global backlog, are investigated. In the case of individual backlog, each event stream S_i owns its private backlog with size Q_i. In the case of global backlog, all event streams in $\mathcal{S}$ share the same backlog.

In this section, we present solutions of how to compute the total service of $\mathcal{S}$ by applying Real-Time Interface (Wandeler and Thiele 2006) theory. Again, the basic assumption of our solutions is that the deadline and backlog requirements for all event streams can be guaranteed if the device always provides full service, i.e., the device never turns to `sleep` mode. Without loss of generality, we consider a stream set $\mathcal{S}$ with N event streams, where $N \geq 2$. We present solutions only for Algorithm EDG. The solutions for the HAD and WCG algorithms are similar to Algorithm EDG and can be easily adapted. Note that the refinements of the history curve and backlog demand in (15) and (17) can be applied to individual stream, denoted as H'_i and B'_i for briefness.

[1]Symbol $(\bar{\alpha}^u)^{-1}$ represents the inverse function of $\bar{\alpha}^u$.

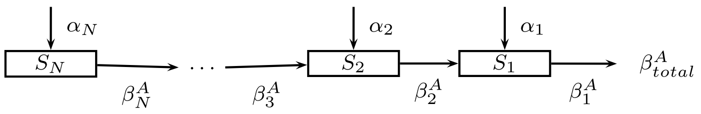


Fig. 6 The flow for computing the total service demand for the FP scheduling with distributed backlog

6.1 FP scheduling with individual backlog

Unlike a system with single event stream where a bounded delay is applied directly to the computed service demand of the event stream, we compute first the individual service demand of every stream, denoted as β_i^A, then derive the total service demand of the set $\mathcal{S}$, denoted as β_{total}^A. With the computed β_{total}^A, the bounded delay is applied to calculate the feasible sleep interval τ^*.

Without loss of generality, the event streams $S_1, S_2, \ldots, S_N$ in $\mathcal{S}$ are ordered according to their priorities. The priority of stream S_i is higher than that of S_k when $k > i$. Event streams can thereby be modeled as an ordered chain according to their priorities and a lower priority stream can only make use of the resource left from a higher priority stream. To compute the service demand of a higher priority stream, a backward approach is applied by considering the service demand from the directly lower priority stream, as shown in Fig. 6.

For the activation scheduling decision of the arrival of an event $e_{N,i}$, the service demand of stream S_N at time $t_w^\perp$ is computed as

$$\beta_N^A(\Delta, t_w^\perp) = \max\{\beta_N^\flat(\Delta, t_w^\perp), \beta_N^\dagger(\Delta, t_w^\perp)\}, \quad \text{where} \tag{20}$$

$$\beta_N^\flat(\Delta, t_w^\perp) = \alpha_N^u(\Delta - D_N, t_w^\perp) + B_N'(\Delta, t_w^\perp) \tag{21}$$

$$\beta_N^\dagger(\Delta, t_w^\perp) = \left\{\alpha_N^u(\Delta, t_w^\perp) - \left(Q_N - |\boldsymbol{E}_N(t_w^\perp)| - \bar{\alpha}_N^l(t_w^\perp - t_{e_{1,i}}^\perp)\right) \cdot w_N\right\}^+ \tag{22}$$

$$\alpha_N^u(\Delta, t_w^\perp) = w_N \cdot \left(\inf_{\lambda \geq 0}\left\{\bar{\alpha}_N^u(\Delta + \lambda) - H_N(\lambda, t_w^\perp)\right\}\right) \tag{23}$$

To derive β_1^A, we have to compute the service bounds $\beta_{N-1}^A, \beta_{N-2}^A, \ldots, \beta_2^A$, sequentially. Suppose that β_k^A has been derived, the resource constraint is that the remaining service curve should be guaranteed to be no less than β_k^A, i.e.,

$$\beta_{k-1}^\sharp(\Delta) \geq \inf\left\{\beta : \beta_k^A(\Delta, t_w^\perp) = \sup_{0 \leq \lambda \leq \Delta}\{\beta(\lambda) - \alpha_{k-1}^u(\lambda, t_w^\perp)\}\right\} \tag{24}$$

By inverting (24), we can derive $\beta_{k-1}^\sharp$ as:

$$\beta_{k-1}^\sharp(\Delta) = \beta_k^A(\Delta - \lambda) + \alpha_{k-1}^u(\Delta - \lambda, t_w^\perp)$$
$$\text{where } \lambda = \sup\left\{\tau : \beta_k^A(\Delta - \tau, t_w^\perp) = \beta_k^A(\Delta, t_w^\perp)\right\} \tag{25}$$

To guarantee the timing constraint of event stream S_{k-1}, we also know that β^A_{k-1} must be no less than its own demand. Therefore, we know that

$$\beta^A_{k-1}(\Delta) = \max\left\{\beta^\sharp_{k-1}(\Delta), \beta^\flat_{k-1}(\Delta, t^\perp_w), \beta^\dagger_{k-1}(\Delta, t^\perp_w)\right\} \tag{26}$$

where

$$\beta^\flat_{k-1}(\Delta, t^\perp_w) = \alpha^u_{k-1}(\Delta - D_{k-1}, t^\perp_w) + B'_{k-1}(\Delta, t^\perp_w) \tag{27}$$

$$\beta^\dagger_{k-1}(\Delta, t^\perp_w) = \left\{\alpha^u_{k-1}(\Delta, t^\perp_w) - \left(Q_{k-1} - |\boldsymbol{E_{k-1}}(t^\perp_w)| - \bar{\alpha}^l_{k-1}(t^\perp_w - t^\perp_{e_{1,i}})\right) \cdot w_{k-1}\right\}^+ \tag{28}$$

$$\alpha^u_{k-1}(\Delta, t^\perp_w) = w_{k-1} \cdot \left(\inf_{\lambda \geq 0}\left\{\bar{\alpha}^u_{k-1}(\Delta + \lambda) - H_{k-1}(\lambda, t^\perp_w)\right\}\right) \tag{29}$$

By applying (26) for $k = N-1, N-2, \ldots, 2$, the service demand β^A_1 of stream S_1 is derived.

Based on this approach, the computed service demand for the highest priority stream S_1 can be also seen as the total service demand β^A_{total} for stream set $\mathcal{S}$ under the fixed-priority scheduling. Therefore, the timing as well as backlog constraints for all streams in $\mathcal{S}$ can be guaranteed by the sleep interval τ^* with which $\boldsymbol{bdf}(\Delta, \tau^*)$ bounds β^A_1:

$$\tau^* = \max\left\{\tau : \boldsymbol{bdf}(\Delta, \tau) \geq \beta^A_1(\Delta), \forall \Delta \geq 0\right\} \tag{30}$$

6.2 EDF scheduling with individual backlog

For earliest-deadline-first scheduling, the total service demand β^A_{total} for all N streams can be bounded by the sum of their service demands. The β^A_{total} computed in this manner, however, is not sufficient to guarantee the backlog constraint of any stream in $\mathcal{S}$. When an event of a stream S_j is happened to have the latest deadline, events in any stream of $\mathcal{S} \setminus \{S_j\}$ will be assigned a higher priority. S_j will suffer from backlog overflow.

To compute a correct service demand which satisfies the backlog constraint for stream S_j, S_j has to be considered as the lowest priority. Similar back-forward approach is applied, as shown in Fig. 7. Instead of tracing back stepwise, the service demand needed for higher-priority streams is the sum of all streams from $\mathcal{S} \setminus \{S_j\}$. Again, we present the revision of the EDG algorithm as an example. The service $\beta^\sharp_j$ to guarantee the lowest priority stream S_j should be more than the demand β^A_j of S_j, i.e.,

$$\beta^\sharp_j(\Delta) \geq \inf\left\{\beta : \beta^A_j(\Delta, t^\perp_w) = \sup_{0 \leq \lambda \leq \Delta}\left\{\beta(\lambda) - \sum_{i \neq j}^{N} \alpha^u_i(\lambda, t^\perp_w)\right\}\right\} \tag{31}$$

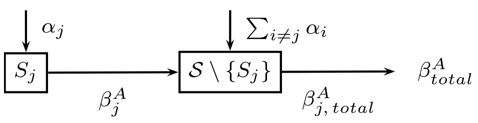


Fig. 7 The computation flow for the total service demand for the EDF scheduling with distributed backlog

By inverting (31), we can derive $\beta_j^{\sharp}(\Delta)$ as:

$$\beta_j^{\sharp}(\Delta) = \beta_j^{A}(\Delta - \lambda, t_w^{\perp}) + \sum_{i \neq j}^{N} \alpha_i^{u}(\Delta - \lambda, t_w^{\perp}) \tag{32}$$

$$\text{where } \lambda = \sup\left\{\tau : \beta_j^{A}(\Delta - \tau, t_w^{\perp}) = \beta_j^{A}(\Delta, t_w^{\perp})\right\}, \quad \text{and}$$

$$\beta_j^{A}(\Delta, t_w^{\perp}) = \max\{\beta_j^{\flat}(\Delta, t_w^{\perp}), \beta_j^{\dagger}(\Delta, t_w^{\perp})\} \tag{33}$$

where $\beta_j^{\flat}$ and $\beta_j^{\dagger}$ are from (27) and (29). To guarantee the timing constraint of all higher-priority streams, we also know that $\beta_{j,total}^{A}$ must be no less than the demand of $\mathcal{S} \setminus \{S_j\}$ as well. Therefore, we know that at time $t_w^{\perp}$,

$$\beta_{j,total}^{A}(\Delta) = \max\left\{\beta_j^{\sharp}(\Delta), \sum_{i \neq j}^{N} \beta_i^{A}(\Delta, t_w^{\perp})\right\} \tag{34}$$

Applying (34) to each steam in $\mathcal{S}$, the service demand for each steam is computed. Because each stream could be the lowest priority in the worst case, only the maximum of them can be seen as the total service demand for stream set $\mathcal{S}$. Therefore, the timing and backlog constraints for $\mathcal{S}$ can be guaranteed by τ^* with which $\boldsymbol{bdf}(\Delta, \tau^*)$ bounds the maximum of individual streams:

$$\tau^* = \max\left\{\tau : \boldsymbol{bdf}(\Delta, \tau) \geq \max_{i \in N}\{\beta_{i,total}^{A}(\Delta)\}, \forall \Delta \geq 0\right\} \tag{35}$$

6.3 EDF scheduling with global backlog

The approach to get the total service demand of $\mathcal{S}$ for global backlog is different to the approach for individual backlog. Without loss of generality, we assume that a backlog with size Q is shared by all event streams of $\mathcal{S}$.

From Wandeler and Thiele (2006), the total service demand for all N streams with respect to EDF scheduling can be bounded by the sum of their arrival curves:

$$\beta^{A} \geq \sum_{i=1}^{N} \alpha_i^{u}(\Delta - D_i) \tag{36}$$

Based on this result, we refine our algorithms.

For the HAD algorithm, because there is no backlog for each evaluation, the related deadline for each event $e_{i,j}$ in every stream S_i remains D_i. Therefore, the service demand to guarantee deadline requirement of all streams is

$$\beta^{\flat}(\Delta) = \sum_{i=1}^{N} \alpha_i^{u}(\Delta - D_i, t^{\top}) \tag{37}$$

In the case of (13) of the WCG algorithm, the backlogs of different streams needs to be considered. We apply the backlog demands fro all streams thereof:

$$\beta^{\flat}(\Delta) = \sum_{i=1}^{N} \left(\alpha_i(\Delta - D_i, t^{\perp}) + w_i \cdot B_i(\Delta, t^{\perp})\right) \tag{38}$$

The same applies to (18) of the EDG algorithm at time $t_w^{\perp}$.

Now we consider the backlog-size constraint. Besides the sum of all arrival curves, the constraint in (12) additionally needs to consider events with the longest execution time, i.e., $\max_{i \in N}\{w_i\}$. Therefore, it is revised as

$$\beta^{\dagger}(\Delta) = \left\{\sum_{i=1}^{N} \alpha_i^u(\Delta) - Q \cdot \max_{i \in N}\{w_i\}\right\}^{+} \tag{39}$$

The backlog constraint in (14) is more complex, because the backlog is not empty and contains events from different streams. The remaining capacity of the backlog is

$$\max_{i \in N}\{w_i\} \cdot Q - \sum_{j=1}^{|E(t^{\perp})|} \sum_{i=1}^{N} x_{i,j} \cdot w_i \tag{40}$$

where $x_{i,j} = 1, \forall j$ for Stream S_i, otherwise 0. Therefore, it is revised as

$$\beta^{\dagger}(\Delta) = \left\{\sum_{i=1}^{N} \alpha_i^u(\Delta, t^{\perp}) - \left(\max_{i \in N}\{w_i\} \cdot Q - \sum_{j=1}^{|E(t^{\perp})|} \sum_{i=1}^{N} x_{i,j} \cdot w_i\right)\right\}^{+} \tag{41}$$

The last revision is (19) of the EDG algorithm, where the estimated future events of all streams need to be counted. Therefore it is revised as

$$\beta^{\dagger}(\Delta) = \left\{\sum_{i=1}^{N} \alpha_i^u(\Delta, t_w^{\perp}) - \left(\max_{i \in N}\{w_i\} \cdot Q - \sum_{j=1}^{|E(t_w^{\perp})|} \sum_{i=1}^{N} x_{i,j} \cdot w_i - \sum_{i=1}^{N} \alpha_i^l(\epsilon)\right)\right\}^{+} \tag{42}$$

7 Performance evaluations

This section provides simulation results for the proposed adaptive dynamic power management schemes. All the results are obtained from an AMD Opteron 2.6 GHz processor with 8 GB RAM. The simulator is implemented in MATLAB by applying MPA and RTS tools from Wandeler and Thiele (2006).

7.1 Simulation setup

We take the event streams studied in Huang et al. (2009), Hamann and Ernst (2005) for our case studies. The specifications of these streams are depicted in Table 1. Parameters `period`, `jitter`, and `delay` are used for generating arrival curves defined

Table 1 Event stream setting according to Huang et al. (2009), Hamann and Ernst (2005)

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}
Period (msec)	198	102	283	354	239	194	148	114	313	119
Jitter (msec)	387	70	269	387	222	260	91	13	302	187
Deday (msec)	48	45	58	17	65	32	78	–	86	89
Wcet (msec)	12	7	7	11	8	5	13	14	5	6

Table 2 Power profiles for devices according to Cheng and Goddard (2006)

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
Maxstream	0.75	0.1	0.05	0.04	7.6
IBM Microdrive	1.3	0.5	0.1	0.012	9.6
SST Flash	0.125	0.05	0.001	0.001	0.098

in Sect. 3.1 and `wcet` represents the worst-case execution time of an event. The relative deadline D_i of a stream S_i is defined by a *deadline factor*, denoted as χ, of its period, i.e., $D_i = \chi * p_i$. To compare the impact of different algorithms, we simulate traces with a 10sec time span. The traces are generated by the RTS tools (Wandeler and Thiele 2006) and conformed to the arrival curve specifications. The history window Δ^h is set to five times of the longest period in a stream set. In our simulations, we adopt the power profiles for four different devices in Cheng and Goddard (2006), depicted in Table 2.

In this work, we evaluate two PPM schemes, i.e., switching to sleep with the HAD algorithm and switching back with the WCG or EDG algorithm, denoted as WCG-HAD and EDG-HAD. To show the effects of our scheme, we report the average idle power that is computed as the total idle energy consumption divided by the time span of the simulation:

$$\frac{E_{sw} \cdot number\_switches + \sum on\_time \cdot P_\sigma}{total\_time\_span} \tag{43}$$

For comparison, two other power management schemes described in Huang et al. (2009) are measured as well, i.e., a periodic scheme (PS) and a naive event-driven scheme (ED). The PS scheme is a periodic power management (PPM) scheme which controls the device with a fixed on-off period, as proposed in Huang et al. (2009). The lengths of the on and off periods are optimally computed with respect to the average idle power by an offline algorithm. The ED scheme turns on the device whenever an event arrives and turns off when the device becomes idle.

We simulate different cases of single and multiple streams. For multiple streams, we only report results for random subsets of the 10-stream set due to space limitation. $\mathcal{S}(3,4)$, for instance, represents a case considering only streams S_3 and S_4 in Table 1. For FP scheduling policy of a multiple-stream set, the stream index defines the priority of a stream. Considering again stream set $\mathcal{S}(3,4)$, for instance, S_3 has higher priority than S_4.

7.2 Simulation results

In the following section, we report simulation results for single-stream cases, multiple-stream cases, and computational expenses.

7.2.1 Single stream

First, we show the effectiveness of the proposed WCG-HAD and EDG-HAD schemes comparing to the PS and ED schemes for single-stream cases. Because PS does not consider the size of the system backlog, for fair comparison, we smooth out the backlog-size effect by setting backlog size to a relative large number, i.e., 60 events for this experiment. Figure 8 shows the *normalized* values of average idle power with respect to PS for streams in Table 1 individually running on all four devices in Table 2. As depicted in the figure, both our proposed schemes in this article outperform the pure event-driven scheme as well as the optimal PPM scheme for all cases. On average, 25% of the average idle power is saved with respect to PS for the deadline factor $\chi = 1.6$.

In general, larger ratio of `jitter` and smaller ratio of `wcet` with respect to a given period result in better energy saving, for instance, the cases for streams S_1 and S_6. On the contrary, the chance to reduce static power is less with larger `wcet` ratio, as in the case of streams S_2, S_7, and S_8. S_8 has the biggest `wcet-period` ratio, thereby conducting the least energy saving for all four devices. Another observation is that the overhead caused by the break-even time does not really affect the optimality of the average idle power. As shown in Fig. 8, the normalized values for a given stream does not change significantly for different devices, although the break-even time is considerably different for the four devices, e.g., 18.2 ms for the SST Flash and 152 ms for the Maxstream.

We also outline how the average idle power changes when the relative deadline of a stream varies. Figure 9 compares the four schemes by varying the deadline factor χ for streams S_8 and S_4. As shown in Fig. 9, our online schemes again outperform the other two. Another observation is that PS can achieve good results only when the relative deadline is large. For the cases of small relative deadlines, it is even worse than ED. Our online schemes, on the contrary, can tackle different deadlines smoothly. The reason is that our online schemes consider the actual arrivals of event, resulting in a more precise analysis of the scheduling decision. Note that ideally our two online schemes, i.e., WCG-HAD and EDG-HAD, should produce identical results, because the WCG and EDG algorithms should theoretically converge to the same mode-switch moment, given a same trace. The slight deviation depicted in these two figures is due to the pessimistic activation decision in Line 4 of the EDG algorithm. This deviation is expected to become larger for multiple stream cases.

7.2.2 Multiple streams

We also present results for multiple-stream cases. Figures 10 and 11 depict simulation results for stream set $\mathcal{S}(6, 9, 10)$ running on Realtek Ethernet with individual and global backlog allocation schemes, respectively. Note that the smallest backlog

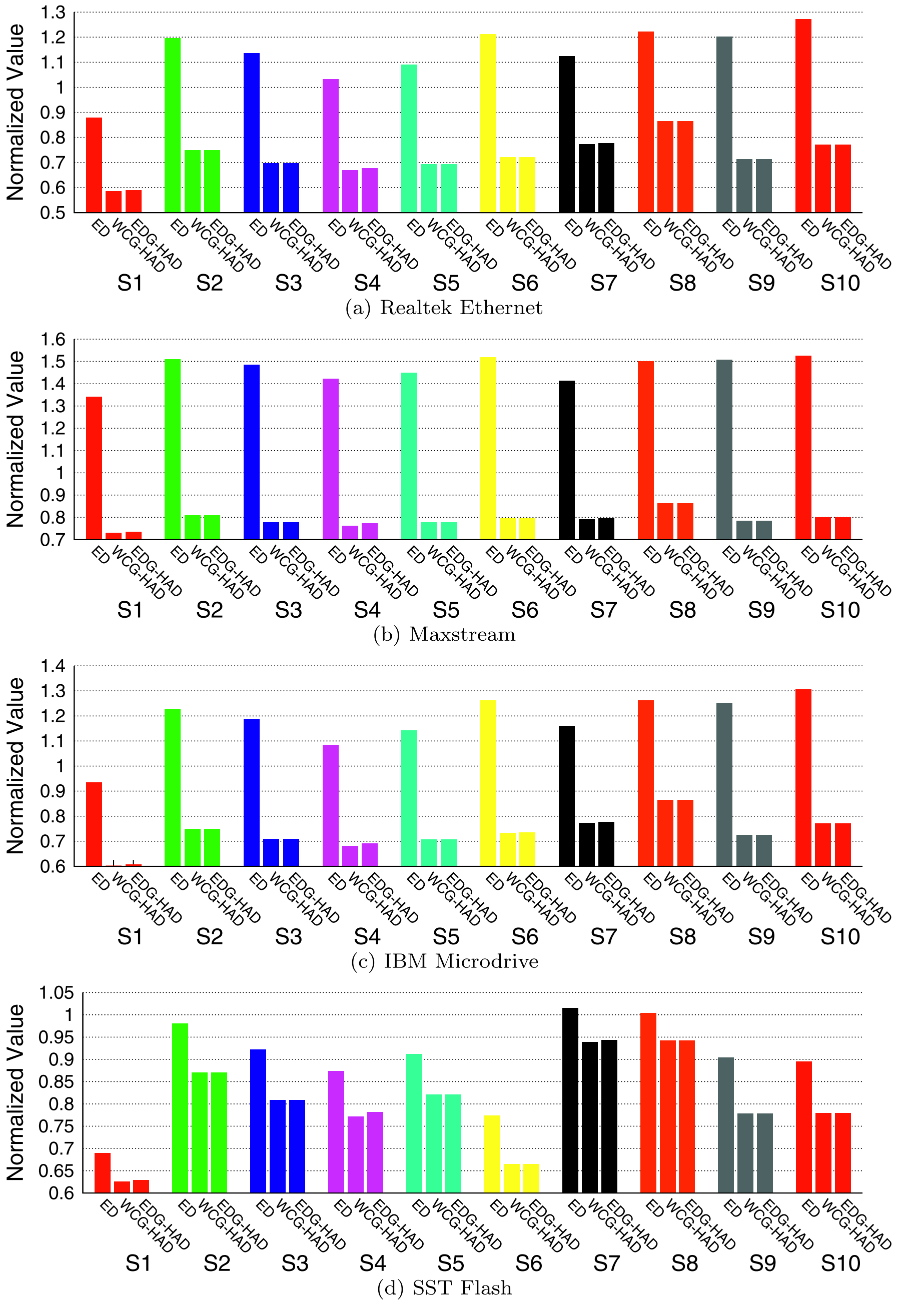


Fig. 8 Normalized average idle power consumption with respect to the PS scheme for single stream cases individually running on four different devices with deadline factor $\chi = 1.6$

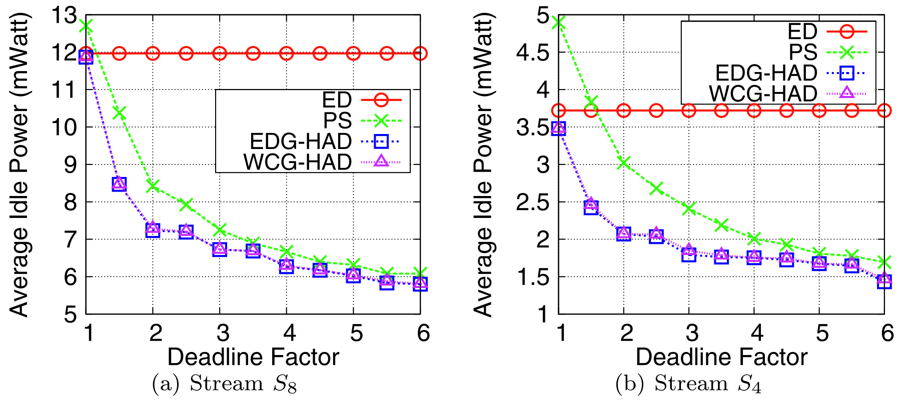


Fig. 9 Average idle power consumption of different deadline settings on Realtek Ethernet

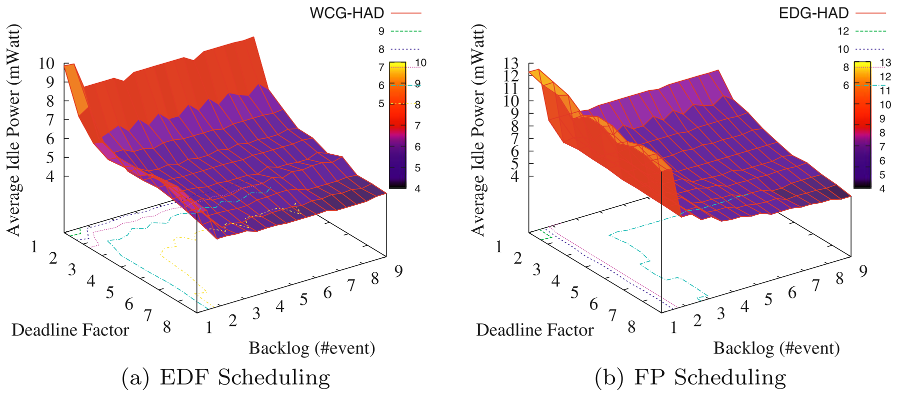


Fig. 10 Average idle power consumption with respect to different deadline and backlog settings on Realtek Ethernet for stream set $\mathcal{S}(6, 9, 10)$ under individual backlog allocation

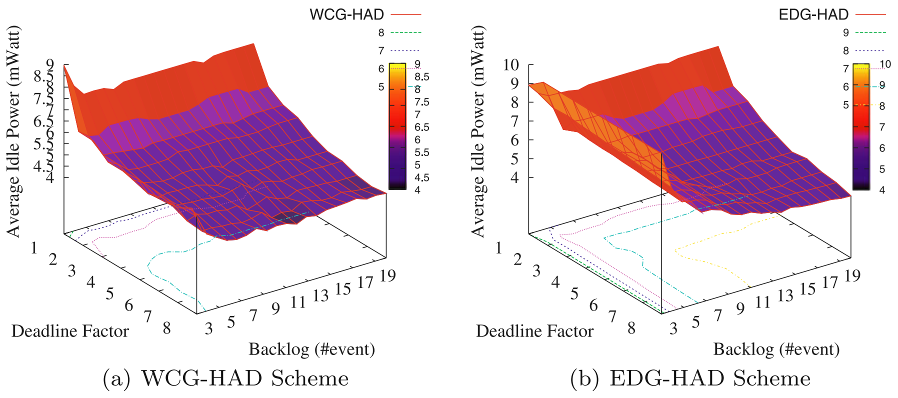


Fig. 11 Average idle power consumption with respect to different deadline and backlog settings for stream set $\mathcal{S}(6, 9, 10)$ running on Realtek Ethernet under global backlog allocation and EDF scheduling policy

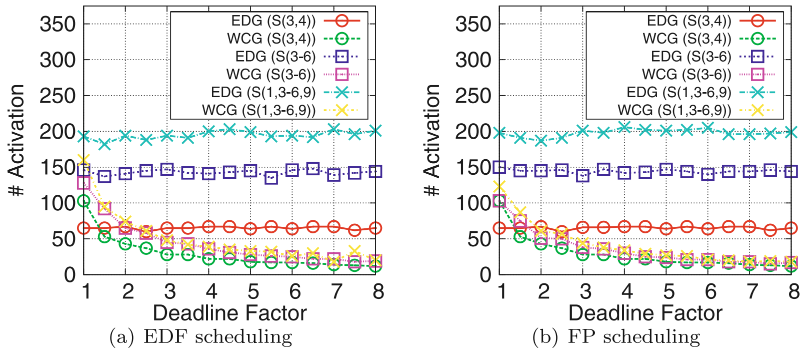


Fig. 12 Numbers of activations for different deadline settings of stream sets $\mathcal{S}(3,4)$, $\mathcal{S}(3\text{–}6)$, and $\mathcal{S}(1,3\text{–}6,9)$ running on Realtek Ethernet under individual backlog allocation with individual backlog size of 10 events

size of Fig. 11 is set to three events, because the stream set used in this experiment contains exactly three event streams. A backlog size smaller than three will results in no sleep interval for any relative deadline setting.

The results from Figs. 10 and 11 demonstrate the effectiveness of our solutions for multiple streams. These results confirm as well as following statements: a) When the relative deadline and backlog size are small, the average idle power is large, where the chances to turn off the device are slim. b) Increasing the relative deadline or backlog size individually helps reduce the idle power for only a certain degree. More increments do not conduct further improvements. c) Increasing both relative deadline and backlog size can effectively reduce the idle power, where more arrived events can be procrastinated and accumulated for each activation of the device.

Another observation is that EDG is more sensitive than WCG for small backlog sizes on both global and individual backlog schemes. As Figs. 10(b) and 11(b) shown, when the backlog sizes increase from 1 to 2 and from 3 to 4 for individual and global backlog schemes, respectively, the idle power drops significantly. The reason is the pessimistic activation decision in Line 4 of Algorithm EDG.

7.2.3 Computation expense

We also demonstrate the efficiency of our schemes by reporting the computational expenses. Figure 12 shows the numbers of activations of our algorithms and Fig. 13 depicts the worst, best, and average case computational expenses for an activation.

From Fig. 12, we can find out that, given a same stream set, the numbers of activations for the EDG algorithm do not vary often as the relative deadline changes, which confirms to the principle of the algorithm. The fluctuations are caused by event arrivals when the device is at `active` mode. Such events do not activate the algorithm. The second observation is that the numbers of activations for EDG are depended on the numbers of streams running on the device while the numbers of activations for WCG quickly converge even more streams are involved. The reason is that the activations of WCG are determined by the predicted turn-on moments that are depended on the backlog size and relative deadline. When the backlog size and relative deadline

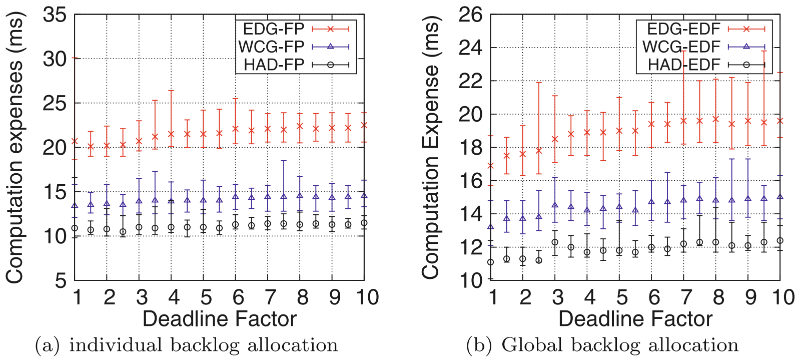


Fig. 13 Worst, best, and average case computation expenses of an activation of the proposed algorithms with respect to different deadline factors for three different 4-stream sets $\mathcal{S}(1–4)$, $\mathcal{S}(3–6)$, and $\mathcal{S}(2, 4, 6, 8)$ individually running on Realtek Ethernet

are large enough, the number of activations is one, no matter how many streams are added to the system.

From above facts, one might conclude that WCG is better. EDG is, however, meaningful in the case when event arrivals are sparse. In such cases, the number of activations of EDG will be less than that of WCG. The results shown in the figure are caused by the dense-event trace generated by the RTS tools.

Figure 13 presents the worst, best, and average case computational expenses of an activation of the proposed algorithms with respect to different deadline factors for stream sets $\mathcal{S}$(1–4), $\mathcal{S}$(3–6), and $\mathcal{S}$(2, 4, 6, 8) individually running on Realtek Ethernet. Results for FP scheduling coupled with individual backlog scheme and EDF scheduling coupled with global backlog scheme are shown in Figs. 13(a) and 13(b), respectively. We neglect the results for EDF scheduling with individual backlog scheme due to the similarity to the FP case.

From the figure, we can conclude that both our algorithms are efficient. The worst/best/average case computation expenses of each activation are within the range of millisecond and are acceptable to the stream set in Table 1. With the new construction of the bounded delay function, the computation time are retained almost constant even with large relative deadlines. In general, EDG is more expensive than WCG and HAD, which are confirmed with the definition in Sect. 5. The last observation is that the computation expenses are not negligible. There are also means to tackle this problem, for instance, setting the computation overhead as a safe margin for the computed sleep period or putting the activation itself as the highest priority events of the system. We do not elaborate them here, since they are not the focus of this article.

8 Complexity analysis

The computational overhead related to our algorithms can be attributed to three parts, i.e., computing the service demand β_i^A for each event stream S_i, the total service demand β^A_{total} for stream set $\mathcal{S}$ in the case of multiple event streams, and the longest feasible sleep interval τ^*.

To compute β^A_i for an event stream S_i, the most computational intensive part consists of numerical operations on curves, for instance, the max-plus de-convolution used in the derivation of the history-aware arrival curve in formula (9). If two curves are periodic, the computational expense depends on the least common multiple of the two periods. If they are aperiodic, the computational expense depends on the number of aperiodic segments.

The timing complexity to compute the total service demand β^A_{total} for event stream set $\mathcal{S}$ is different for EDF and FP scheduling policies. In the case of FP scheduling, $O(N)$ operations on curves are needed because of the backward approach where the service demand of a higher priority stream is derived from the directly lower priority stream. In the case of EDF scheduling, $O(N^2)$ operations on curves are needed since each event stream could be the lowest priority during runtime.

To compute τ^*, a binary search can be applied since $\max\{\beta^\flat(\Delta), \beta^\dagger(\Delta)\}$ in (8) is monotonically increasing. Therefore, assuming the possible number of τ^* is m, computing τ^* needs $O(\log m)$ operations on curves.

9 Conclusions and future work

This article explores how to apply dynamic power management to reduce the static power consumption for hard real-time embedded systems pertaining to both timing and backlog constraints. We propose algorithms to adaptively control the power mode of a device (system) based on the actual arrival of events, tackling multiple event streams with irregular event arrival patterns under both earliest deadline first and fixed priority preemptive scheduling. Proof-of-concept simulation results demonstrate the effectiveness of approaches.

Note that the computational expenses of our approach are not negligible, especially for Algorithm EDG. These computation expenses are caused by the expensive numerical (de)convolution for curve operations, for instance, the derivation of history-aware arrival curves. We are currently working on new methods, trying to replace the expensive numerical operations with light-weight analytical approach for curve predictions.

Another fact is that although our algorithms are designed to reduce the energy consumption while satisfying the timing constraints, the energy reduction of our algorithms are, however, not proved to be competitive. For future work, we would like to explore the competitiveness of our algorithms for the static power minimization problem to provide worst-case guarantees and hopefully also average case improvements.

Furthermore, another interesting future work would be to support non-preemptive scheduling. Methods to support non-preemptive fix-priority scheduling in the context of Real-Time Calculus have been developed in Haid and Thiele (2007), the idea of which could be used to compute service demands for a set of event streams under non-preemptive scheduling. In addition, supporting multiple devices is also an interesting topic, which requires methods to support the analysis of global scheduling for a set of devices. We are also investigating new methods in these directions.

References

Agarwal A, Mukhopadhyay S, Raychowdhury A, Roy K, Kim C (2006) Leakage power analysis and reduction for nanoscale circuits. IEEE MICRO 26(2):68–80

APC (American Power Conversion) (2003) Determining total cost of ownership for data center and network room infrastructure. http://www.apcmedia.com/salestools/CMRP-5T9PQG_R2_EN.pdf

Augustine J, Irani S, Swamy C (2004) Optimal power-down strategies. In: 45th symposium on foundations of computer science (FOCS), pp 530–539

Austin T, Blaauw D, Mahlke S, Mudge T, Chakrabarti C, Wolf W (2004) Mobile supercomputers. IEEE Comput 37:81–83

Aydin H, Melhem R, Mossé D, Mejía-Alvarez P (2001) Dynamic and aggressive scheduling techniques for power-aware real-time systems. In: Proceedings of the 22nd IEEE real-time systems symposium (RTSS), pp 95–105

Bao M, Andrei A, Eles P, Peng Z (2010) Temperature-aware idle time distribution for energy optimization with dynamic voltage scaling. In: Proceedings of the 13th design, automation and test in Europe (DATE), pp 21–26

Baptiste P (2006) Scheduling unit tasks to minimize the number of idle periods: a polynomial time algorithm for offline dynamic power management. In: Proceedings of the 17th annual ACM-SIAM symposium on discrete algorithm (SODA), pp 364–367

Chen J-J, Kuo T-W (2007) Procrastination determination for periodic real-time tasks in leakage-aware dynamic voltage scaling systems. In: Int'l conf. on computer-aided design (ICCAD), pp 289–294

Chen J-J, Stoimenov N, Thiele L (2009) Feasibility analysis of on-line DVS algorithms for scheduling arbitrary event streams. In: Proceedings of the 30th IEEE real-time systems symposium (RTSS).

Cheng H, Goddard S (2006) Online energy-aware I/O device scheduling for hard real-time systems. In: Proceedings of the 9th design, automation and test in Europe (DATE), pp 1055–1060

Cruz R (1991) A calculus for network delay. I. Network elements in isolation. IEEE Trans Inf Theory 37(1):114–131

Devadas V, Aydin H (2008) On the interplay of dynamic voltage scaling and dynamic power management in real-time embedded applications. In: Proceedings of ACM international conference on embedded software (EMSOFT), pp 99–108

Devadas V, Aydin H (2010) DFR-EDF: A unified energy management framework for real-time systems. In: IEEE real-time and embedded technology and applications symposium (RTAS), pp 121–130

Haid W, Thiele L (2007) Complex task activation schemes in system level performance analysis. In: Proc. int'l conf. on HW/SW codesign and system synthesis (CODES/ISSS). Salzburg, Austria, pp 173–178

Hamann A, Ernst R (2005) Tdma time slot and turn optimization with evolutionary search techniques. In: Proceedings of the 8th design, automation and test in Europe (DATE), pp 312–317

Heo J, Henriksson D, Liu X, Abdelzaher T (2007) Integrating adaptive components: an emerging challenge in performance-adaptive systems and a server farm case-study. In: Proceedings of the 28th IEEE real-time systems symposium (RTSS), pp 227–238

Huang K, Santinelli L, Chen J-J, Thiele L, Buttazzo GC (2009) Adaptive dynamic power management for hard real-time systems. In: The 30th IEEE real-time systems symposium (RTSS). Washington DC, pp 23–32

Huang K, Santinelli L, Chen J-J, Thiele L, Buttazzo GC (2009) Periodic power management schemes for real-time event streams. In: The 48th IEEE conf. on decision and control (CDC). Shanghai, China, pp 6224–6231

Huang K, Santinelli L, Chen J-J, Thiele L, Buttazzo GC (2010) Adaptive power management for real-time event streams. In: The 15th IEEE conf. on Asia and South Pacific design automation conference (ASP-DAC), pp 7–12

Irani S, Shukla S, Gupta R (2003) Algorithms for power savings. In: Proceedings of the 14th annual ACM-SIAM symposium on discrete algorithms (SODA), pp 37–46

International technology roadmap for semiconductors 2009 edition: System drivers. http://www.itrs.net/Links/2009ITRS/2009Chapters_2009Tables/2009_SysDrivers.pdf

Jejurikar R, Pereira C, Gupta R (2004) Leakage aware dynamic voltage scaling for real-time embedded systems. In: ACM/IEEE design automation conference (DAC), pp 275–280

Le Boudec J-Y (1998) Application of network calculus to guaranteed service networks. IEEE Trans Inf Theory 44(3):1087–1096

Maxiaguine A, Chakraborty S, Thiele L (2005) DVS for buffer-constrained architectures with predictable QoS-energy tradeoffs. In: International conference on hardware-software codesign and system synthesis (CODES+ISSS), pp 111–116

Maxiaguine A, Künzli S, Thiele L (2004) Workload characterization model for tasks with variable execution demand. In: Proceedings of the 7th design, automation and test in Europe (DATE)
Roy K, Mukhopadhyay S, Mahmoodi-Meimand H (2003) Leakage current mechanisms and leakage reduction techniques in deep-submicrometer CMOS circuits. Proc IEEE 91(2):305–327
Shrivastava A, Earlie E, Dutt N, Nicolau A (2005) Aggregating processor free time for energy reduction. In: Proceedings of the 3rd IEEE/ACM/IFIP international conference on hardware/software codesign and system synthesis (CODES+ISSS), pp 154–159
Stephan H (2007) Power management of digital circuits in deep sub-micron CMOS technologies. Springer, Berlin
Swaminathan V, Chakrabarty K (2005) Pruning-based, energy-optimal, deterministic I/O device scheduling for hard real-time systems. ACM Trans Embed Comput Syst 4(1):141–167
Thiele L, Chakraborty S, Naedele M (2000) Real-time calculus for scheduling hard real-time systems. IEEE Int. Symp. Circuits Syst. Proc. 4:101–104
Thiele L, Wandeler E, Stoimenov N (2006) Real-time interfaces for composing real-time systems. In: International conference on embedded software (EMSOFT), pp 34–43
Wandeler E, Thiele L (2006) Interface-based design of real-time systems with hierarchical scheduling. In: 12th IEEE real-time and embedded technology and applications symposium, pp 243–252
Wandeler E, Thiele L (2006) Real-Time Calculus (RTC) Toolbox. http://www.mpa.ethz.ch/Rtctoolbox
Wandeler E, Thiele L, Verhoef M, Lieverse P (2006) System architecture evaluation using modular performance analysis—a case study. Int J Softw Tools Technol Transf 8(6):649–667
Yang C-Y, Chen J-J, Hung C-M, Kuo T-W (2007) System-level energy-efficiency for real-time tasks. In: The 10th IEEE international symposium on object/component/service-oriented real-time distributed computing (ISORC), pp 266–273
Yang C-Y, Chen J-J, Thiele L, Kuo T-W (2010) Energy-efficient real-time task scheduling with temperature-dependent leakage. In: Proceedings of the 13th design, automation and test in Europe (DATE), pp 9–14
Yao F, Demers A, Shenker S (1995) A scheduling model for reduced CPU energy. In: Proceedings of the 36th annual symposium on foundations of computer science (FOCS), pp 374–382
Zhang Y, Hu X, Chen DZ (2002) Task scheduling and voltage selection for energy minimization. In: Proceedings of the 39th ACM/IEEE design automation conference (DAC), pp 183–188
Zhuo J, Chakrabarti C (2005) System-level energy-efficient dynamic task scheduling. In: Proceedings of the 42nd ACM/IEEE design automation conference (DAC), pp 628–631

Kai Huang received his Ph.D. degree in the Computer Engineering and Networks Laboratory of ETH Zurich, Switzerland, in 2010. He received B.Sc. degree in computer science at Fudan University, China, in 1999 and M.Sc. degree in computer science at Leiden University, The Netherlands, in 2005. His research interests include methods and tools for high performance embedded computing. He received Best Paper Awards from Int'l Symposium on Systems, Architectures, Modeling and Simulation (SAMOS) and General Chairs' Recognition Award For Interactive Papers in the IEEE Conf. on Decision and Control (CDC) in 2009.

Luca Santinelli is a postdoc at the INRIA Nancy Grand Est. He received the Ph.D. in Computer Engineering from the Scuola Superiore Sant'Anna in December 2010. His research interests cover scheduling algorithms, real-time operating systems, resource reservation mechanisms and probabilistic real-time analysis.

Jian-Jia Chen joined Department of Informatics at Karlsruhe Institute of Technology (KIT) in Germany as a Juniorprofessor for Institute for Process Control and Robotics (IPR) in 2010. He received his Ph.D. degree from Department of Computer Science and Information Engineering, National Taiwan University, Taiwan in 2006. He received his B.S. degree from the Department of Chemistry at National Taiwan University 2001. After finishing the compulsory civil service in Dec. 2007, between Jan. 2008 and April 2010, he was a postdoc researcher at Computer Engineering and Networks Laboratory (TIK) in ETH Zurich, Switzerland. His research interests include real-time systems, embedded systems, energy-efficient scheduling, power-aware designs, temperature-aware scheduling, and distributed computing. He received Best Paper Awards from ACM Symposium on Applied Computing (SAC) in 2009 and IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA) in 2005.

Lothar Thiele joined ETH Zurich, Switzerland, as a full professor of computer engineering in 1994, where he currently leads the Computer Engineering and Networks Laboratory. He received his Diplom-Ingenieur and Dr.-Ing. degrees in Electrical Engineering from the Technical University of Munich in 1981 and 1985 respectively. His research interests include models, methods and software tools for the design of embedded systems, embedded software and bioinspired optimization techniques. Lothar Thiele is associate editor of IEEE Transaction on Industrial Informatics, IEEE Transactions on Evolutionary Computation, Journal of Real-Time Systems, Journal of Signal Processing Systems, Journal of Systems Architecture, and INTEGRATION, the VLSI Journal. In 1986 he received the "Dissertation Award" of the Technical University of Munich, in 1987, the "Outstanding Young Author Award" of the IEEE Circuits and Systems Society, in 1988, the Browder J. Thompson Memorial Award of the IEEE, and in 2000–2001, the "IBM Faculty Partnership Award". In 2004, he joined the German Academy of Sciences Leopoldina. In 2005, he was the recipient of the Honorary Blaise Pascal Chair of University Leiden, The Netherlands. Since 2009 he is a member of the Foundation Board of Hasler Foundation, Switzerland. Since 2010, he is a member of the Academia Europaea.

Giorgio C. Buttazzo is Full Professor of Computer Engineering at the Scuola Superiore Sant'Anna of Pisa. He graduated in Electronic Engineering at the University of Pisa in 1985, received a Master in Computer Science at the University of Pennsylvania in 1987, and a Ph.D. in Computer Engineering at the Scuola Superiore Sant'Anna of Pisa in 1991. From 1987 to 1988, he worked on active perception and real-time control at the G.R.A.S.P. Laboratory of the University of Pennsylvania, Philadelphia. Prof. Buttazzo has been Program Chair and General Chair of the major international conferences on real-time systems. He is Chair of the IEEE Technical Committee on Real-Time Systems and member of the Euromicro Executive Board on Real-Time Systems. He has authored 6 books on realtime systems and over 200 papers in the field of real-time systems, robotics, and neural networks. He is Senior Member of IEEE.