Adaptive Dynamic Power Management for Hard Real-Time Systems

Kai Huang, Luca Santinelli*, Jian-Jia Chen, Lothar Thiele, Giorgio C. Buttazzo*
Computer Engineering and Networks Laboratory, ETH Zurich, Switzerland
Email: {firstname.lastname}@tik.ee.ethz.ch
* Real-Time Systems Laboratory, Scuola Superiore Sant'Anna of Pisa, Italy
* Email:{luca.santinelli,giorgio}@sssup.it

***Abstract*—Power dissipation has constrained the performance boosting of modern computer systems in the past decade. Dynamic power management has been widely applied to change the system (or device) state dynamically to reduce the power consumption. This paper explores how to effectively reduce the energy consumption to handle event streams with hard real-time guarantees. We adopt Real-Time Calculus to describe the event arrival and resource service by arrival curves and service curves in the interval domain, respectively. We develop online algorithms to adaptively control the power mode of the device, postponing the processing of arrival events as late as possible. Profited from the worst-case interval-based abstraction, our algorithms can on one hand tackle arbitrary event arrivals (even with burstiness) and on the other hand guarantee hard real-time requirements in terms of both timing and backlog constraints. We also present simulation results to demonstrate the effectiveness of our algorithms.**

Keywords: Power Management, Real-Time Event Streams, Real-Time Calculus, Real-Time Interface.

I. Introduction

Power dissipation has been an important design issue in a wide range of computer systems in the past decade. 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 for the reduction of power bills. Dynamic power consumption due to switching activities and static power consumption due to the leakage current are two major sources of power consumption of a CMOS circuit [12]. For micrometer-scale semiconductor technology, the dynamic power dominates the power consumption of a processor. However, for technology in the deep sub-micron (DSM) domain, the leakage power consumption is comparable to or even more than the dynamic power dissipation.

This paper explores how to apply dynamic power management (DPM) to reduce the energy consumption while satisfying the real-time constraints. We consider devices with *active*, *standby*, and *sleep* modes with different power consumptions, in which a controller decides when to change the power modes of devices. Intuitively, the device can be switched to the sleep mode to reduce the power consumption when it is idle. This switching operation, however, has two concerns. On one hand, the sleep period should be long enough to recuperate the mode-switch overhead. On the other hand, to cope with the burstiness of event arrivals, the reserved time for serving the burst events must be sufficient to prevent deadline violation of events and overflow of the system backlog when activating the device again later on.

To resolve these two concerns, we propose online algorithms that are applicable for the controller. We apply Real-Time Calculus [20] to predict future event arrival and Real-Time Interface [21] for the schedulability analysis. Specifically, we try to be optimistic to handle events only when they really arrive. Our algorithms adaptively predict the next moment for mode switch by considering both historical and future event arrivals, and procrastinate the buffered and future events as late as possible without violating the timing and backlog constraints for the given event streams. To demonstrate the performance of the proposed approach, several case studies are explored, in which the results reveal the effectiveness of our approach.

The rest of this paper is organized as follows: In the next section, we review the related work in the literature. Section III provides system models. Section IV presents a set of new routines which will be used later on. Section V presents our proposed algorithms for one event stream, while Section VI copes with multiple event streams. Simulations results are presented in Section VII. Section VIII concludes this paper.

II. Related Work

Dynamic power management (DPM) can be applied to control the change of system mode to consume less leakage power, e.g., to a sleep mode. For systems with the sleep mode, Baptiste [2] proposes an algorithm based on dynamic programming to control when to turn on/off the system for aperiodic real-time events with the same execution time. For multiple low-power modes, Augustine et al. [1] determine the mode that the processor should enter for aperiodic real-time events and propose a competitive algorithm for online use. Swaminathan et al. [18, 19] 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. [17] propose a framework for code transformations. Leakage-aware scheduling has also been recently explored

on dynamic voltage scaling (DVS) platforms, such as [3, 4, 11, 12, 13]. The basic idea behind the above results is to execute at some speed (mostly at the critical speed) and control the procrastination of the real-time events as long as possible so that the idle interval is long enough to reduce the energy consumption.

Most of the above approaches require either precise information of event arrivals, such as systems with periodic real-time events [3, 4, 11, 12, 13] or aperiodic real-time events with known arrival time [1, 2, 10]. However, in practice, the precise information of event arrival time might not be known in advance since the arrival time depends on many factors. When the precise information of event arrivals is unknown, to our best knowledge, the only known approach is to apply the online algorithms proposed by Irani et al. [10] and Augustine et al. [1] to control when to turn on the system. However, since the online algorithms in [1, 10] greedily stay in the sleep mode as long as possible without referring to incoming events in the near future, the resulting schedule might cause an event to miss its deadline.

To model such irregular events, Real-Time Calculus, extended from Network Calculus [7], was proposed by Thiele et al. [20, 25] 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. In [15], Maxiaguine et al. 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. [5] 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, this paper focuses on DPM. In a recent work [9], we study periodic DPM, finding the best on-off period with respect to energy saving by offline algorithms. In this paper, we propose online algorithms. The adaptation points are dynamic and adaptively vary according to the actual arrivals of events. Furthermore, we provide solution on multiple event-stream scenarios where event streams with different periods can be tackled with earliest-deadline-first scheduling.

III. System Models and Problem Definition

Fig. 1 illustrates a block diagram of our system where a device is managed by a controller. The controller could be the operating system where the device is an I/O peripheral device, for instance. Events of different streams, e.g. S_i, arrives to the controller and must be served on the device. We assume that the controller has a global backlog to buffer events before they are processed. When an event is buffered, it is put to the backlog until it is fetched to the device for processing. The backlog size is Q, hence buffering

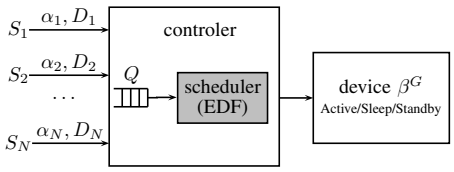


Figure 1: The abstract model of our DPM problem.

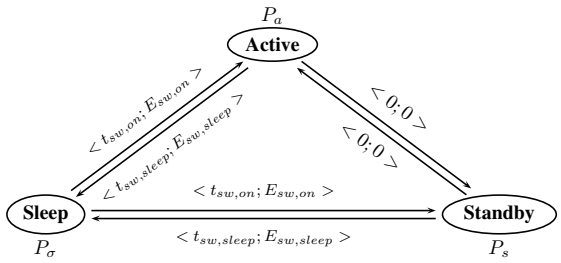


Figure 2: Example for the state transit of a device, where the tuple one each transit is the timing overhead and energy overhead.

more than Q events incurs a backlog overflow causing a controller failure. To schedule events of different streams in the backlog, we consider the earliest-deadline-first (EDF) [14], scheduling by which the device executes the event with the earliest deadline among all events in the backlog. Parameters α, D, and β^G will be introduced next.

A. Power Model

The device has three power consumption modes, including *active, standby,* and *sleep* modes. The power consumption in the sleep mode is P_σ. To serve an event, the device must be in the active mode with power consumption P_a, in which $P_a > P_\sigma$. Once there is no event to serve, the device can enter the sleep mode. However, switching from the sleep mode to the active mode takes time, denoted by $t_{sw,on}$, and requires additional energy overhead, denoted by $E_{sw,on}$. 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 less than P_a and is more than P_σ. In this paper, we assume that switching between the standby mode and the active mode has negligible overhead, compared to the other switches, which is the same as the assumption in [26, 27]. Moreover, switching from the active (also standby) mode to the sleep mode takes time, denoted by $t_{sw,sleep}$, and requires additional energy overhead, denoted by $E_{sw,sleep}$. Fig. 2 illustrates the state diagram of these three modes.

B. Event Model

This paper focuses on events that arrive to the controller irregularly. To model such events, we adopt the arrival curves based on Real-Time Calculus [20]. Specifically, a trace of

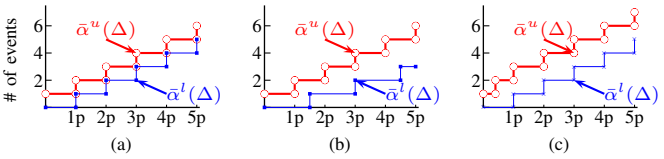


Figure 3: Examples for arrival curves for: (a) periodic events with period p, (b) events with minimal inter-arrival distance p and maximal inter-arrival distance $p' = 1.5p$, and (c) events with period p, jitter $j = p$, and minimal inter-arrival distance $d = 0.4p$.

an event stream can conveniently be described by means of a cumulative function $R(t)$, defined as the number of events seen on the event stream in the time interval $[0, t)$. While any R always describes one concrete trace of an event stream, a tuple $\bar{\alpha}(\Delta) = [\bar{\alpha}^u(\Delta), \bar{\alpha}^l(\Delta)]$ of upper and lower arrival curves provides an abstract event stream model, providing bounds on admissible traces of an event stream. The upper arrival curve $\bar{\alpha}^u(\Delta)$ provides an upper bound on the number of events that are seen in any time interval of length Δ, while the lower arrival curve $\bar{\alpha}^l(\Delta)$ analogously provides a lower bound. Please refer to [20] for a detailed discussion.

The concept of arrival curves unifies many other common timing models of event streams. For example, a periodic event stream can be modeled by step functions $\bar{\alpha}^u(\Delta) = \left\lfloor \frac{\Delta}{p} \right\rfloor + 1$ and $\bar{\alpha}^l(\Delta) = \left\lfloor \frac{\Delta}{p} \right\rfloor$. For a sporadic event stream with minimal inter arrival distance p and maximal inter arrival distance p', the upper and lower arrival curve is $\bar{\alpha}^u(\Delta) = \left\lfloor \frac{\Delta}{p} \right\rfloor + 1$, $\bar{\alpha}^l(\Delta) = \left\lfloor \frac{\Delta}{p'} \right\rfloor$, respectively. Moreover, for an event stream with period p, jitter j, and minimal inter arrival distance d, the upper arrival curve is $\bar{\alpha}^u(\Delta) = \min\{\left\lfloor \frac{\Delta+j}{p} \right\rfloor, \lceil \frac{\Delta}{d} \rceil\}$. Fig. 3 illustrates arrival curves for the above cases. For details, please refer to [20].

Analogously to the cumulative function $R(t)$, the concrete availability of the device can be described by a cumulative function $C(t)$, that is defined as the number of available resources, e.g., processor cycles or bus capacity, in the time interval $[0, t)$. Analogous to arrival curves that provide an abstract event stream model, a 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.

Note that an arrival curve $\bar{\alpha}_i(\Delta)$ specifies the (upper-bounded or lower-bounded) number of events of event stream S_i for every time interval Δ, while a service curve $\beta(\Delta)$ specifies the (upper-bounded or lower-bounded) available amount of time for execution for every time interval Δ. Therefore, the arrival curve $\bar{\alpha}_i^u(\Delta)$ (respectively, $\bar{\alpha}_i^l(\Delta)$) has to be transformed to the arrival curve $\alpha_i^u(\Delta)$ (respectively, $\alpha_i^l(\Delta)$) to indicate the amount of computation time required for the arrived events in intervals. Suppose that the execution time of an event of event stream S_i is w_i. Then, 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 $\bar{\alpha}_i^u = \alpha^u / w_i$, $\bar{\alpha}_i^l = \alpha^l / w_i$. In the case of variable workloads, workload curves [16] can be applied.

Moreover, to satisfy the real-time constraint of event stream S_i, the response time of an event in event stream S_i must be no larger than its specified relative deadline D_i, where the response time of an event is its finishing time minus its arrival time. The arrival time of an event $e_{i,j}$ of event stream S_i is characterized by its arrival time $A_{i,j}$. The execution time of $e_{i,j}$ is w_i and its absolute deadline $D_{i,j}$ is $A_{i,j} + D_i$.

C. Historical Information

As dynamic power management is a dynamic process depending on event arrival, scheduling decisions are made on-line. To predict the number of events in the *near future*, we consider a past history of event arrivals. For example, if we have already observed burstiness recently, one can predict that in the near future only sparse events will arrive due to the constraint on the arrival curve. To have a more precise prediction, it would be good to have an infinitely long history, but this is not practical for system implementation. In this paper, we assume that we have only a fixed-size history window, in which the history length is at most Δ^h. That is, at time instant t', we can only refer to the events arrived in $[t' - \Delta^h, t']$.

We use $H_i(\Delta, t')$ to denote the accumulated number of events of stream S_i arriving in time interval $[t' - \Delta, t')$. Since the history window is Δ^h, by definition, $H_i(\Delta, t')$ is $H_i(\Delta^h, t')$ for any $\Delta > \Delta^h$. Therefore, if $R_i(t)$ is the accumulated number of events of stream S_i arriving in $[0, t)$, we have:

$$H_i(\Delta, t') = \begin{cases} R_i(t') - R_i(t - \Delta), & \text{if } \Delta \leq \Delta^h; \\ R_i(t') - R_i(t' - \Delta^h), & \text{otherwise.} \end{cases} \quad (1)$$

D. Problem Definition and Approach Overview

This paper explores how to effectively minimize the energy consumption to serve a set $\mathcal{S}$ of event streams under real-time and backlog constraints. Clearly a device should not be turned to sleep if there are events in the backlog for execution. To decide whether the device should be turned to the sleep mode, we need to know whether the sleeping period is long enough. If the sleeping period is too short, the mode switching overhead $E_{sw} = E_{sw,on} + E_{sw,sleep}$ might be more than the energy saved by turning the device to the sleep mode. Therefore, we know that there is a break-even time $\frac{E_{sw}}{P_s - P_\delta}$ such that if the sleeping time is shorter than this break-even time, turning the device to the sleep mode is not worthwhile. Moreover, the timing overhead for mode switches must be considered. Therefore, for brevity, the

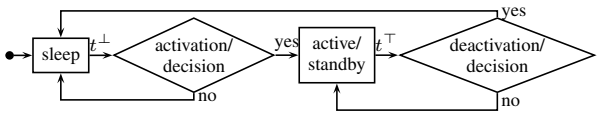


Figure 4: The control flow of our approach.

following break-even time T_{BET} will be used for deciding whether we should switch to the sleep mode:

$$T_{BET} = \max\left\{t_{sw,on} + t_{sw,sleep}, \frac{E_{sw}}{P_s - P_\delta}\right\}.$$

In general, we have to decide when to turn a device to the active mode to serve events from a sleep mode, and when to turn it back to the sleep mode for reducing energy consumption. Therefore, we have to deal with *deactivation scheduling decisions* and *activation scheduling decisions* to switch safely and effectively. The overview of our approach is illustrated in Figure 4.

For deactivation scheduling decisions, when the device is in the active mode and there is no event in the backlog, we have to decide whether the device has to change to the sleep mode instantly or it should keep in the on mode 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 paper, time instants for deactivation scheduling decisions are denoted by $t^{\top}$.

On the other hand, for activation scheduling decisions, when the device is in sleep mode and there is an event arriving or the sleep interval set by the controller expires, we have 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 for processing, so preventing unnecessary mode switches. For brevity, for the rest of this paper, time instants for deactivation scheduling decisions are denoted by $t^{\perp}$.

We say that the scheduling decision is *feasible* if it is always possible to meet the timing and backlog constraints for any event traces constrained by the arrival curve. An algorithm is *feasible* if it always generates feasible scheduling decisions.

IV. Analysis Based on Real-Time Calculus and Real-Time Interface

In this section, we review the known results of Real-Time Calculus and Real-Time Interface, based on which we derive new results of bounded delay backlog constraint and future prediction with history information.

A. Bounded-Delay Function

For event streams described by Real-Time Calculus, one can apply Real-Time Interface [21] to verify whether a system can provide guarantee output service $\beta^G(\Delta)$. Correspondingly, to guarantee that all events in an event stream,

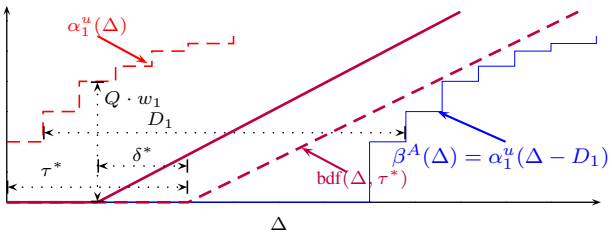


Figure 5: An example for the bounded delay function for event stream S_1, in which only part of the upper arrival curve $\alpha_1^u(\Delta)$ is presented for simplicity.

e.g. S_i, can be processed while respecting all timing constraints, the event stream demands a service bound $\beta^A(\Delta)$. To satisfy the required relative deadline D_i, $\beta^A(\Delta)$ can be computed as $\beta^A(\Delta) = \alpha_i^u(\Delta - D_i)$. To check the schedulability of event stream S_i in the system, the following predicate has to be true:

$$\beta^G(\Delta) \geq \beta^A(\Delta), \qquad \forall \Delta \geq 0. \tag{2}$$

To guarantee feasibility, we could apply the *bounded delay function* for constructing the β^G. A bounded-delay function $\boldsymbol{bdf}(\Delta, \tau)$ is constrained by suffering at most τ unit of time without service providing, in which

$$\boldsymbol{bdf}(\Delta, \tau) = \max\{0, (\Delta - \tau)\}, \text{ for } \Delta \geq 0. \tag{3}$$

Moreover, the longest delay τ^* for providing service guarantee $\beta^A(\Delta)$ is as follows:

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

If the backlog has a finite length Q, directly applying (4) will cause the backlog overflow in the case that $Q \cdot w_i < \sup_\Delta\{\alpha_i^u(\Delta) - \boldsymbol{bdf}(\Delta, \tau^*)\}$. Therefore, to prevent the backlog overflow, we have to reduce the longest bounded delay τ^* by δ^* which is defined as

$$\delta^* = \max\left\{0, \min\{\delta : \alpha_i^u(\Delta) - \boldsymbol{bdf}(\Delta, \tau^* - \delta) \leq Q \cdot w_i, \forall \Delta\}\right\}. \tag{5}$$

Based on the above analysis, we state the following lemma.

***Lemma** 1: Given τ^* and δ^* from (4) and (5) with $\tau^* - \delta^* \geq T_{BET}$, at any moment $t^{\top}$ for making deactivation scheduling decisions when the device is in the active mode without any event in the backlog, it is feasible to deactivate the device from time $t^{\top}$ to $t^{\top} + (\tau^* - \delta^*)$, such that no event of the stream S_1 will miss its deadline.*

Proof: Based on the known results from Real-Time Interface [21], we sketch the proof here. The service demand β^A of S_i can be computed as $\alpha_i^u(\Delta - D_i)$ and $\boldsymbol{bdf}(\Delta, \tau^*)$ can be seen as the minimal β^G that fulfills (2). To satisfy the backlog constraint, $\tau^* - \delta^*$ can be seen as a right shift of the guarantee output service β^G, which provides additional amount of service to prevent the backlog overflow. ■

Figure 5 depicts an example for the above analysis for the case of a single stream S_1. We For multiple event streams, we present results for EDF scheduling in Section VI. Note that to compensate the time overhead for a mode-switch, the device has to be activated $t_{sw,on}$ earlier. For brevity, we omit this by intention for the rest of the paper.

B. Future Prediction with Historical Information

As shown in Section III, historical information can help improving the prediction of future event arrival. To know how events will arrive in the future, $H_i(\Delta, t')$ defined in (1) can be applied to $\alpha_i(\Delta)$ to calculate the maximal number of events of event stream S_i that will arrive in the future. Specifically, suppose that at time t', the maximal number of events of stream S_i that will arrive in time interval $[t', t' + \Delta)$, denoted as $\bar{\alpha}_i^u(\Delta, t')$, can be bounded by

$$\bar{\alpha}_i^u(\Delta, t') \leq \inf_{\lambda \geq 0} \{\bar{\alpha}_i^u(\Delta + \lambda) - H_i(\lambda, t')\}. \tag{6}$$

***Lemma** 2: Equation (6) holds for any time instant t'.*

Proof: This comes directly from Equation (6). ■

The amount of computation time $\alpha_i^u(\Delta, t')$ requested by stream S_i based on the historical information is thereby

$$\alpha_i^u(\Delta, t') \leq w_i \cdot \bar{\alpha}_i^u(\Delta, t'). \tag{7}$$

V. Adaptive Algorithm for One Event Stream

In this section, we present our holistic approach to minimize energy consumption. The assumption of our approach is that the scheduling decision is always feasible when a device receives full service, i.e., when the device never turns to sleep mode. For simplicity, we consider only one event stream, says S_1. We first present how to deal with deactivation of scheduling decisions and then propose two methods for the activation of scheduling decisions. The extension to multiple event streams under EDF scheduling is presented in the next section.

A. Deactivation Scheduling Decisions

The History-Aware Deactivation (HAD) algorithm analyzes whether the device should be turned to the sleep mode from an active mode. The principle is to deactivate the device only when energy saving is possible. As we have discussed in Section III-D, deactivate scheduling decisions are only needed while the device is in the active or standby mode and there is no new arrival of events, as well as no event in the backlog. Suppose that $t^\top$ is such a time instant.

Turning the device instantly at time $t^\top$ to the sleep mode does not always help. The reason is that we pay overheads for mode switching. In the case there are events arriving in the very near future, the device has to be activated again to process these events. If the energy saving obtained from the short sleep period cannot counteract this switching overhead, the mode switching only introduces additional energy consumption. Therefore, the first step for the HAD algorithm is to identify whether there will be definitely an event coming in the near future by speculating the upper arrival curve, as well as the history. The earliest time t_ϵ after time $t^\top$ for the next event arrival can be calculated by the following equation:

$$t_\epsilon \leftarrow \min_{t > t^\top} t \text{ such that } \bar{\alpha}_1^u(t - t^\top, t^\top) > 0, \tag{8}$$

where the derivation of $\bar{\alpha}_1^u(t_\epsilon - t^\top, t^\top)$ is defined in (6). Equation (8) can be implemented by applying the inverse function of arrival curve α_1^u in Real-Time Calculus Toolbox [24].

If $t_\epsilon - t^\top$ is larger than the break-even time T_{BET}, we could greedily turn the device to the dormant mode. Otherwise, we have to be a little bit careful, because switching the device on when the first future event arrives has counter effect. In this case, energy saving can only be obtained by procrastinating the process of these future events after $t^\top + T_{BET}$. To guarantee that the procrastination will not cause any deadline missing, we apply the bounded-delay function in Section IV to calculate the maximally postponed period. Specifically, we first calculate the worst-case arrival curve $\hat{\alpha}_1^u(\Delta, t_\epsilon)$ after time t_ϵ by applying (7). Therefore, by applying the bounded delay function similarly in (4) and (5), we know that

$$\tau^\top = \max\{\tau : \textbf{\textit{bdf}}(\Delta, \tau) \geq \alpha_1^u(\Delta - D_1, t_\epsilon)\}, \tag{9}$$

$$\delta^\top = \max\{0, \min\{\delta : \alpha_1^u(\Delta, t_\epsilon) - \textbf{\textit{bdf}}(\Delta, \tau^\top - \delta) \leq Q \cdot w_1, \forall \Delta\}\}. \tag{10}$$

Based on this construction, turning the device to the sleep mode at time $t^\top$ and activating it again at time $t_\epsilon + \tau^\top - \delta^\top$ will not cause constraint violations. If $\tau^\top - \delta^\top + t_\epsilon - t^\top$ is larger than T_{BET}, we turn the device to the sleep mode at time $t^\top$; otherwise, we do not turn the device to the sleep mode. The above analysis leads to following theorem:

***Theorem** 1: The HAD algorithm guarantee a feasible scheduling upon a deactivation decision at any time $t^\top$ for one event-stream system, if the device provides full service from time $t^\epsilon + \tau^\top - \delta^\top$ on.*

Proof: We prove this theorem by contradiction. At any time instance $t^\top$ at which the HAD algorithm decides to deactivate the device, the latest activation time to prevent constraint violations is $t^\epsilon + \tau^\top - \delta^\top$. Suppose at a later 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 Φ. Because of the deadline missing, the service demand $\Phi \cdot w_1$ in this interval is larger than our constructed service supply $\textbf{\textit{bdf}}(\lambda, \tau^\top - \delta^\top)$ 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 $\Phi > \bar{\alpha}_1^u(\lambda, t^\top)$ contradicts to the definition in (6) and Lemma 2. Therefore, the theorem holds. For the case of backlog constraint, similar proof can be constructed. ■

Algorithm 1 HAD deactivation

Input: $\alpha_1()$, $H_1()$, Δ^h, Q, T_{BET}, and $t^\top$
Output: decision to deactivate the device

```
find t_ε by (8);
if t_ε − t^⊤ > T_BET then
    return true;
else
    compute τ^⊤ and δ^⊤ by (9) and (10);
    if τ^⊤ − δ^⊤ + t_ε − t^⊤ > T_BET then
        return true;
    end if
end if
return false;
```

As a result, the HAD algorithm reduces the number of mode switches to save energy consumption. The pseudo code of the HAD algorithm is depicted in Algorithm 1.

B. History-Aware Activation

When to activate a device needs more intelligence. The idea is to aggregate as many arrived events as possible for each activation not only to reduce the standby period, but also to minimize the number of mode switches. In this paper, we develop two algorithms, called worst-case-greedy (WCG) algorithm and event-driven-greedy (EDG) algorithm, for deciding when to activate the device to the active mode for event processing. The WCG algorithm decides the earliest time when the device should change to the active mode for event processing, such that all the events will meet their timing and backlog constraints. The EDG algorithm computes the latest time that the device must be activated to satisfy the timing constraint. The difference between these two algorithms is the following:

- Algorithm WCG conservatively assumes worst-case event arrivals and decides the earliest time to activate the device. If the worst case does not occur, the predicted time is too early, hence the device is kept in the sleep mode for a longer period, and the next activation moment is recomputed.
- Algorithm EDG optimistically considers the least event arrivals and decides the latest time to activate the device. Upon the arrival of each new event before the predicted time, the algorithm reevaluates the scheduling condition and decides whether the predicted activation time needs to shift earlier.

Once the controller decides to keep the device in the sleep mode, it has to setup an *alarm* to wake up the device.

Therefore, there are two cases to be considered for activation. The activation scheduling decision must be done either at time instants of event arrivals or alarm arrivals. We identify these two cases by *event arrival* and *alarm arrival* at time instant $t^\perp$, in which the device is in the sleep mode.

For brevity, we denote the set of unfinished events in the backlog at time $t^\perp$ as $\boldsymbol{E_1}(t^\perp)$, which is maintained by the controller. Suppose that the events in $\boldsymbol{E_1}(t^\perp)$ are indexed as $e_{1,1}, e_{1,2}, \ldots, e_{1,|\boldsymbol{E}(t^\perp)|}$ from the earliest deadline to the latest deadline, where $|\boldsymbol{E_1}(t^\perp)|$ is the number of events. Note that although the absolute deadline $D_{1,j}$ for every event in the backlog does not change, the related deadline is not D_1 anymore. It varies according to related distance from $t^\perp$. To accurate model the service demand for those events in the backlog, we define backlog demand curve for time $t^\perp$ as

$$B_1(\Delta, t^\perp) = \begin{cases} j-1, & D_{1,j} - t^\perp < \Delta \le D_{1,j+1} - t^\perp; \\ |\boldsymbol{E_1}(t^\perp)|, & \Delta > D_{1,|\boldsymbol{E_1}(t^\perp)|} - t^\perp \end{cases} \quad (11)$$

where $D_{1,0}$ is defined as $t^\perp$.

1) Worst-Case-Greedy (WCG) Activation: When an event comes at time $t^\perp$, we have two cases depending on the backlog status. If there already exist some events in the backlog, it is not necessary to switch the device to the active mode at $t^\perp$. This is because keeping the device in the sleep mode does not violate the timing and backlog constraints before the previously computed alarm arrives. In the case of no event in the backlog, we have to setup the first alarm to decide the earliest time we have to activate the device by:

$$\beta^A(\Delta) = \alpha_1^u(\Delta - D_1, t^\perp), \quad (12)$$

$$\tau^\perp = \max\{\tau : \boldsymbol{bdf}(\Delta, \tau) \ge \beta^A(\Delta)\}, \quad (13)$$

$$\delta^\perp = \max\{0, \min\{\delta : \alpha_1^u(\Delta, t^\perp) - \boldsymbol{bdf}(\Delta, \tau^\perp - \delta) \le Q \cdot w_1, \forall\Delta\}\}. \quad (14)$$

Therefore, the WCG algorithm simply sets the wakeup alarm at time $\tau^\perp - \delta^\perp$ if $\tau^\perp - \delta^\perp > 0$; otherwise, we have to wake up the device right at time $t^\perp$.

Then, the next question is whether the controller has to wakeup the device at time $t^\perp$ when the wakeup alarm arrives. Indeed, if the worst-case burst arrives, we have to activate the device immediately; otherwise, the device can still be kept in the sleep mode. We again apply the bounded-delay function to evaluate how long we can still keep the device in the sleep mode as follows:

$$\beta^A(\Delta) = \alpha_1^u(\Delta - D_1, t^\perp) + w_1 \cdot B_1(\Delta, t^\perp), \quad (15)$$

$$\tau^\perp = \max\{\tau : \boldsymbol{bdf}(\Delta, \tau) \ge \beta^A(\Delta)\}, \quad (16)$$

$$\delta^\perp = \max\Big\{0, \min\{\delta : \alpha_1^u(\Delta, t^\perp) - \boldsymbol{bdf}(\Delta, \tau^\perp - \delta) \le \big(Q - |\boldsymbol{E}(t^\perp)|\big) \cdot w_1, \forall\Delta\}\Big\}. \quad (17)$$

The constructed β^A in (12) and (15) bounds the future arrival events and the derived δ from (14) and (17) guarantees the backlog constraint, which lead to following theorem:

***Theorem** 2: The WCG algorithm guarantees a feasible scheduling upon an activation decision at any wakeup alarm $t^\perp$ for one event-stream system, if the device provides full service from time $t^\perp$ on.*

We omit the proof due to the similarity to Theorem 1. The WCG algorithm is effective in the sense that it greedily

Algorithm 2 WCG activation

procedure *event arrival at time* $t^\perp$*:*
1: **if** the arrival event is the only one in the backlog **then**
2: compute $\tau^\perp$ and $\delta^\perp$ by (13) and (14);
3: **if** $\tau^\perp - \delta^\perp > 0$ **then**
4: set the wakeup alarm at time $t^\perp + \tau^\perp - \delta^\perp$;
5: **else**
6: wakeup the device;
7: **end if**
8: **end if**
procedure *alarm arrival at time* $t^\perp$*:*
1: compute $\tau^\perp$ and $\delta^\perp$ by (16) and (17);
2: **if** $\tau^\perp - \delta^\perp > 0$ **then**
3: set the wakeup alarm at time $t^\perp + \tau^\perp - \delta^\perp$;
4: **else**
5: wakeup the device;
6: **end if**

extends the sleep period as long as the device is schedulable. It is efficient as well when the event arrival pattern is close to the worst case scenario, where the reevaluation of the wakeup alarm does not take place often. The number of reevaluation is bounded by $\alpha_1^u(D_1 - w_1)$.

2) Event-Driven-Greedy (EDG) Activation: In contrast to the WCG algorithm which predicts the earliest activation time, the EDG algorithm predicts the latest one. The idea of the EDG algorithm is to compute the latest moment by assuming the least case event arrival scenario in the future. This decision is refined upon the event arrivals, unlike the WCG algorithm where reevaluation takes place when each wakeup alarm arrives.

On the arrival of an event $e_{1,j}$ at time $t^\perp$, we choose the latest possible moment $t' = t^\perp + D_1 - w_1$ for processing as the reference point to compute the alarm time for $e_{1,j}$. At this time t', the service demand includes a) the possible burst from t' on, which is bounded by $\bar{\alpha}_1^u(\Delta,\, t')$, b) the backlog until $t^\perp$, and c) the estimated least event arrival between $[t^\perp,\, t')$, constrained by $\bar{\alpha}_1^l(\Delta)$.

To compute a precise $\bar{\alpha}_1^u(\Delta,\, t')$, we first revise the historical information $H_1(\Delta,\, t')$ by advancing the time from $t^\perp$ to t' such that least events come after $t^\perp$. We denote such a trace by $H'(\Delta, t')$, in which

$$H_1'(\Delta, t') = \begin{cases} \bar{\alpha}_1^l(\epsilon) - \bar{\alpha}_1^l(\epsilon - \Delta), & \text{if } \Delta < \epsilon, \\ H_1(\Delta, t^\perp) + \bar{\alpha}_1^l(\epsilon), & \text{if } \epsilon < \Delta < \Delta^h - \epsilon, \\ H_1(\Delta^h - \epsilon, t^\perp) + \bar{\alpha}_1^l(\epsilon), & \text{otherwise,} \end{cases} \quad (18)$$

where $\epsilon = t' - t^\perp$ for abbreviation. The H_1' can be seen as the concatenation of the historical information H_1 until $t^\perp$ and the time inversion of $\bar{\alpha}_1^l$ in the interval $[0, \epsilon)$. The worst-case arrival curve after time t' with the new historical information H_1' is

$$\bar{\alpha}_i^u(\Delta, t') \le \inf_{\lambda \ge 0} \left\{ \bar{\alpha}_i^u(\Delta + \lambda) - H_i'(\lambda, t') \right\}. \quad (19)$$

Algorithm 3 EDG activation

procedure *event arrival at time* $t^\perp$*:*
1: calculate $\tau^\perp$ and $\delta^\perp$ at time t' by (18–23);
2: **if** $\tau^\perp - \delta^\perp > 0$ **then**
3: new wakeup alarm $t_{new}^\perp \leftarrow t'$;
4: **else**
5: new wakeup alarm $t_{new}^\perp \leftarrow t' - (\tau^\perp - \delta^\perp)$;
6: **end if**
7: new wakeup alarm $t_{new}^\perp \leftarrow \min\{t_{new}^\perp,\, t_{old}^\perp\}$
procedure *alarm arrival at time* $t^\perp$*:*
1: wakeup the device;

Similarly, $\alpha_1^u(\Delta, t') = w_1 \cdot \bar{\alpha}_1^u(\Delta, t')$.

The corresponding backlog demand curve that encapsulates the estimated least arrival events within the interval $[t^\perp,\, t')$ is

$$B_1'(\Delta,\, t') = \begin{cases} j - 1, & D_{1,\,j} - t' < \Delta \le D_{1,\,j+1} - t'; \\ \mathcal{E}, & \Delta > D_{1,\,\mathcal{E}} - t', \end{cases} \quad (20)$$

where $\mathcal{E} = |\boldsymbol{E_1}(t^\perp)| + \bar{\alpha}_1^l(\epsilon)$, $\epsilon = t' - t^\perp$, and $D_{1,\,0}$ is defined as $t^\perp$.

With the refined historical information and backlog demand, we can again apply the bounded-delay function to find the next wakeup alarm for event $e_{1,\,j}$:

$$\beta^A(\Delta) = \alpha_1^u(\Delta - D_1,\, t') + w_1 \cdot B_1'(\Delta,\, t'), \quad (21)$$

$$\tau^\perp = \max\{\tau : \boldsymbol{bdf}(\Delta, \tau) \ge \beta^A(\Delta)\}, \quad (22)$$

$$\delta^\perp = \max\Big\{0,\, \min\{\delta : \alpha_1^u(\Delta, t') - \boldsymbol{bdf}(\Delta, \tau^\perp - \delta) \le \big(Q - |\boldsymbol{E}(t^\perp)| - \bar{\alpha}_1^l(\epsilon)\big) \cdot w_1,\, \forall \Delta\}\Big\}. \quad (23)$$

If $\tau^\perp - \delta^\perp > 0$, the wakeup alarm remain unchanged. Otherwise, the new wakeup alarm is set to the minimum of the old one and $t' + \tau^\perp - \delta^\perp$. Note that at the first execution of the EDG algorithm, the wakeup alarm can be simply set to $t^\perp + D_1 - w_1$. In this way, the new wakeup alarm computed by (18)–(23) bounds all the worst cases, which leads to the following theorem:

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

We omit the proof here due to the similarity to Theorem 1. The EDG algorithm is effective in the sense that it greedily refines the latest alarm arrival time and preserves the longest possible sleep period. The EDG algorithm is efficient as well when the worst-case event arrival seldom occurs, because the number of reevaluation is the number of events actually arrived. The pseudo code of the EDG algorithm is depicted in Algorithm 3.

VI. Multiple Event Streams

This section presents how to extend our algorithms to cope with multiple event streams. We focus on EDF scheduling [14] in this work. Note the refined history curve (18), the backlog curve (11) and its refinement (20) can be applied to individual streams. In the rest of this section, we denote them as H'_i, B_i, and B'_i for event stream S_i, respectively.

Suppose that there are N event streams where $N \geq 2$. From [22], the total service demand for all N streams can be bounded by the sum of their arrival curves:

$$\beta^A \geq \sum_{i=1}^{N} \alpha_i^u(\Delta - D_i). \tag{24}$$

Based on this result, we refine our algorithms.

First we investigate the evaluation of the bounded delay τ. 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, (9) can directly apply the total demand services from (24):

$$\tau^\top = \max\{\tau : \boldsymbol{bdf}(\Delta, \tau) \geq \sum_{i=1}^{N} \alpha_i^u(\Delta - D_i, t_\epsilon)\} \tag{25}$$

The same refinement applies to (12) and (13) of the WCG algorithm.

In the case of (15) and (16) of the WCG algorithm, the backlogs of different streams needs to be considered. We apply the backlog demands fro all streams thereof:

$$\beta^A(\Delta) = \sum_{i=1}^{N} \big(\alpha_i(\Delta - D_i,\, t^\perp) + w_i \cdot B_i(\Delta,\, t^\perp)\big). \tag{26}$$

The same applies to (21) and (22) of the EDG algorithm.

Now we consider the backlog constraint δ. Besides the sum of all arrival curves, the constraint in (10) additionally needs to consider events with the longest execution time, i.e., $\max_{i\in N}\{w_i\}$. Therefore, it is revised as

$$\delta^\top = \max\Big\{0,\, \min\{\delta : \sum_{i=1}^{N} \alpha_i^u(\Delta) - \boldsymbol{bdf}(\Delta, \tau^\top - \delta) \leq Q \cdot \max_{i\in N}\{w_i\},\, \forall\Delta\}\Big\}. \tag{27}$$

The same revision applies to (14) of the WCG algorithm.

The backlog constraint in (17) is more complex, because the backlog is not empty and contains events from different streams. The remaining capacity of the backlog is

$$\max_{i\in N}\{w_i\} \cdot Q - \sum_{j=1}^{|E(t^\perp)|} \sum_{i=1}^{N} x_{i,j} \cdot w_i \tag{28}$$

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

$$\delta^\perp = \max\Big\{0,\, \min\{\delta : \sum_{i=1}^{N} \alpha_i^u(\Delta, t^\perp) - \boldsymbol{bdf}(\Delta, \tau^\perp - \delta) \leq \max_{i\in N}\{w_i\} \cdot Q - \sum_{j=1}^{|E(t^\perp)|} \sum_{i=1}^{N} x_{i,j} \cdot w_i\}\Big\}. \tag{29}$$

Table I: Event stream setting according to [8, 9, 23].

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

Table II: Power profiles for devices according to [6].

Device Name	P_a (W)	P_s (W)	P_σ (W)	t_{sw} (S)	E_{sw} (mJ)
Realtec Ethenet	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

The last revision is (23) of the EDG algorithm, where the estimated future events of all streams need to be counted. Therefore it is revised as

$$\delta^\perp = \max\Big\{0,\, \min\{\delta : \sum_{i=1}^{N} \alpha_i^u(\Delta, t') - \boldsymbol{bdf}(\Delta, \tau^\perp - \delta) \leq \max_{i\in N}\{w_i\} \cdot Q - \sum_{j=1}^{|E(t^\perp)|} \sum_{i=1}^{N} x_{i,j} \cdot w_i - \sum_{i=1}^{N} \alpha_i^l(\epsilon)\}\Big\}. \tag{30}$$

VII. Performance Evaluations

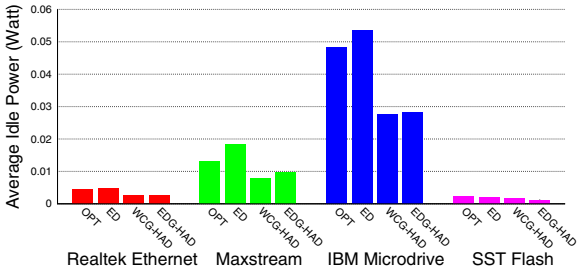
This section provides simulation results for the proposed adaptive dynamic power management schemes. All the results are obtained from a simulation host with Intel 1.6 GHz processor and 1 GB RAM. The simulator is implemented in MATLAB and applies MPA and RTS tools from [24].

A. Simulation Setup

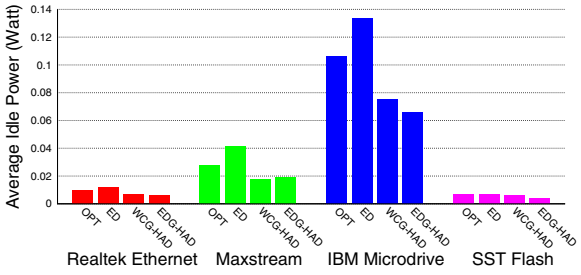
We take the event streams studied in [8, 9, 23] for our case studies. Table I presents the parameters for generating the arrival curves defined in Section III-B, where w is the worst-case execution time of an event. The relative deadline D_i of an event stream S_i is defined by a *deadline factor* χ, 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 [24] and conformed to the arrival curve specifications. The history window Δ^h is 200msec and the backlog size Q is 5. In our simulations, we adopt the power profiles for four different devices in [6], as presented in Table II.

In this work, we evaluate two schemes, i.e., WCG-HAD and EDG-HAD. For comparison, two other power management schemes presented in [9], i.e., a periodic scheme (OPT) and a naive event-driven scheme (ED), are depicted as well. The OPT scheme repeatedly applies an on-off period with fixed length which is optimally computed off-line. The ED scheme turns on the device whenever an event arrives and turns off when the device becomes idle.

We simulate different scenarios with one event stream and multiple event streams. Due to space limitation, we only



(a)



(b)

Figure 6: Average idle power consumption for stream S_5 (a) and S_8 (b) with $\chi = 1.6$.

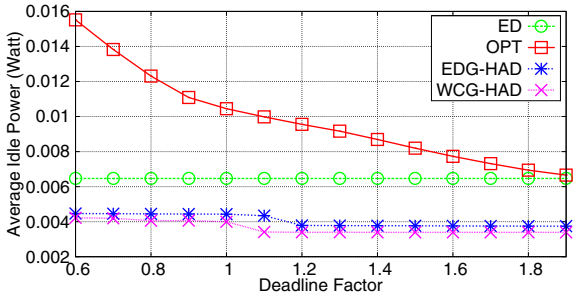


Figure 7: Average idle power consumption of different deadline settings for event stream S_1 on Realtek Ethernet.

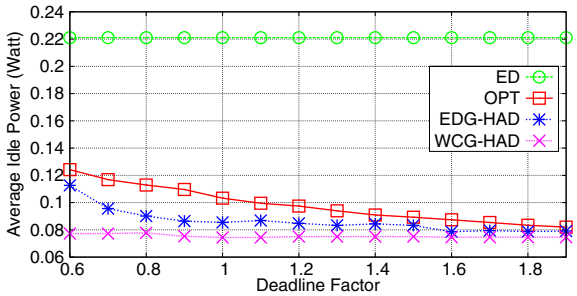


Figure 8: Average idle power consumption of the 10-stream scenario for Realtek Ethernet with EDF scheduling.

report the results for considering (a) one stream and (b) 10 streams where all the 10 event streams in Table I are scheduled with EDF scheduling. Since all the schemes have the same energy consumption for event processing, we only report the average idle power which is computed as the total idle energy consumption divided by the time span of the simulated trace.

B. *Simulation Results*

Firstly, we show the effectiveness of the proposed WCG-HAD and EDG-HAD schemes comparing to the OPT and the ED schemes. Fig. 6 shows the average idle power for streams S_5 and S_8 in Table I, subjected to the four devices specified in Table II. As shown in the figure, both our proposed schemes outperform the offline-computed periodic OPT scheme as well as the pure event-driven ED scheme for all cases. On average, 35% of the average idle power is saved with respect to the OPT scheme.

We also outline how the average idle power changes when the relative deadlines of events vary. Fig. 7 and Fig. 8 compare the four schemes by varying the deadline factor χ. In Fig. 7, a system with one event stream (S_1 in Table I) is simulated. Fig. 8 considers scenarios with all the 10 event streams in Table I with EDF scheduling. As these two figures shown, our online schemes again outperform the other two. Another observation is that the OPT scheme can achieve good results only when the relative deadline is large or event arrival is dense. On the contrary, our online schemes can tackle cases for both short relative deadline and sparse event arrival. 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 converge to the same mode-switch moment, given a same trace. The deviation depicted in these two figures is due to the refinement of the relative deadlines for the events in the backlog.

Secondly, we demonstrate the efficiency of the proposed schemes by reporting the computation time for each iteration. We depict the computation time for activation and deactivation separately: WCG-HAD$_{\text{on}}$ and WCG-HAD$_{\text{off}}$ represent the computation time for the WCG-HAD scheme to compute the turning on and the next turning off instants, respectively, while EDG-HAD$_{\text{on}}$ and EDG-HAD$_{\text{off}}$ represent the computation time that the EDG-HAD scheme requires to compute the turning on and the next turning off instant, respectively. Fig. 9 shows the relation of the computation time and the deadline factor for the scenario of the 10 event streams case in Fig. 8. As shown in the figure, both schemes require a small computation time and the increment for the case of a longer relative deadline is considerably small, which makes our algorithms applicable online.

VIII. Conclusion

This paper discussed how to apply dynamic power management to reduce energy consumption under hard real-time

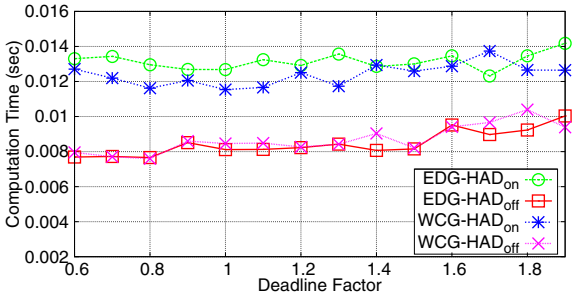


Figure 9: Computation time of WCG-HAD, and EDG-HAD for Realtek Ethernet with EDF scheduling.

constraints. We considered systems (or devices) with active, standby, and sleep modes with different power consumptions. For scheduling one event stream under hard real-time constraints, we presented online algorithms to adaptively control the on/off of a device. Additions to multiple events streams under EDF scheduling were also presented by extending the Modular Performance Analysis [25]. To demonstrate the performance of the proposed schemes, several case studies were explored, in which the results reveal the effectiveness of our approaches.

ACKNOWLEDGMENT

The work was partially supported by European Integrated Project SHAPES (grant no. 26825) under IST FET – Advanced Computing Architecture (ACA) and the European Community's Seventh Framework Programme FP7/2007-2013 project Predator (grant no. 216008).

REFERENCES

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

[2] P. Baptiste. 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)*, pages 364–367, 2006.

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

[4] J.-J. Chen and T.-W. Kuo. Procrastination determination for periodic real-time tasks in leakage-aware dynamic voltage scaling systems. In *International Conference on Computer-Aided Design (ICCAD)*, pages 289–294, 2007.

[5] J.-J. Chen, N. Stoimenov, and L. Thiele. Feasibility analysis of online dvs algorithms for scheduling arbitrary event streams. In *IEEE Real-Time Systems Symposium (RTSS)*, 2009.

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

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

[8] A. Hamann and R. Ernst. Tdma time slot and turn optimization with evolutionary search techniques. In *Proceedings of the 8th Design, Automation and Test in Europe (DATE)*, pages 312–317, 2005.

[9] K. Huang, L. Santinelli, J.-J. Chen, L. Thiele, and G. C. Buttazzo. Periodic power management schemes for real-time event streams. In *Proceedings of the 48th IEEE Conference on Decision and Control (CDC)*, Dec. 2009.

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

[11] R. Jejurikar and R. K. Gupta. Procrastination scheduling in fixed priority real-time systems. In *Proceedings of the ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, pages 57–66, 2004.

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

[13] Y.-H. Lee, K. P. Reddy, and C. M. Krishna. Scheduling techniques for reducing leakage power in hard real-time systems. In *15th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 105–112, 2003.

[14] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the Association for Computing Machinery*, 20(1):46–61, Jan. 1973.

[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 (CODES+ISSS)*, pages 111–116, 2005.

[16] A. Maxiaguine, S. Künzli, and L. Thiele. Workload characterization model for tasks with variable execution demand. In *Proceedings of the 7th Design, Automation and Test in Europe (DATE)*, 2004.

[17] A. Shrivastava, E. Earlie, N. Dutt, and A. Nicolau. 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)*, pages 154–159, 2005.

[18] V. Swaminathan and C. Chakrabarti. Energy-conscious, deterministic I/O device scheduling in hard real-time systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(7):847–858, 2003.

[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. *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, 4:101–104, 2000.

[21] L. Thiele, E. Wandeler, and N. Stoimenov. Real-time interfaces for composing real-time systems. In *International Conference On Embedded Software (EMSOFT)*, pages 34–43, 2006.

[22] E. Wandeler and L. Thiele. Interface-based design of real-time systems with hierarchical scheduling. In *12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 243–252, Apr. 2006.

[23] E. Wandeler and L. Thiele. Optimal TDMA time slot and cycle length allocation for hard real-time systems. In *11th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 479–484, 2006.

[24] E. Wandeler and L. Thiele. Real-Time Calculus (RTC) Toolbox. http://www.mpa.ethz.ch/Rtctoolbox, 2006.

[25] 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.

[26] C.-Y. Yang, J.-J. Chen, C.-M. Hung, and T.-W. Kuo. System-level energy-efficiency for real-time tasks. In *the 10th IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC)*, pages 266–273, 2007.

[27] J. Zhuo and C. Chakrabarti. System-level energy-efficient dynamic task scheduling. In *Proceedings of the 42nd ACM/IEEE Design Automation Conference(DAC)*, pages 628–631, 2005.