

Maximizing System Lifetime by Battery Scheduling*

Marijn Jongerden^{†¶} Boudewijn Haverkort[‡] Henrik Bohnenkamp[§] Joost-Pieter Katoen^{†§}

[†] University of Twente
Centre for Telematics and
Information Technology
7500 AE Enschede, The Netherlands
{jongerdenmr, brh}@ewi.utwente.nl
Phone/Fax: +31 53 489 8041/4524

[‡] Embedded Systems Institute
Eindhoven, The Netherlands
boudewijn.haverkort@esi.nl
Phone: +31 40 247 4720

[§] RWTH Aachen University
Software Modeling and Verification
52056 Aachen, Germany
{henrik,katoen}@cs.rwth-aachen.de
Phone/Fax: +49 241 80 21201/22217

Abstract

The use of mobile devices is limited by the battery lifetime. Some devices have the option to connect an extra battery, or to use smart battery-packs with multiple cells to extend the lifetime. In these cases, scheduling the batteries over the load to exploit recovery properties usually extends the system lifetime. Straightforward scheduling schemes, like round robin or choosing the best battery available, already provide a big improvement compared to a sequential discharge of the batteries. In this paper we compare these scheduling schemes with the optimal scheduling scheme produced with a priced-timed automaton battery model (implemented and evaluated in Uppaal Cora). We see that in some cases the results of the simple scheduling schemes are close to optimal. However, the optimal schedules also clearly show that there is still room for improving the battery lifetimes.

Keywords. Scheduling, Embedded Systems, Batteries, Lifetime Optimization, Kinetic Battery Model, Priced-Timed Automata.

1. Introduction

Mobile devices usually rely on batteries for their power supply. The capacity of the batteries is finite, and the duration with which one can use the device is limited by the battery lifetime. Lifetime, here, is the time of one discharge

period of the battery, from full to empty. While the battery lifetime depends mostly on its capacity and the level of the load applied to it, another important influence, on which we will focus in this paper, is *how the battery is used*, *i.e.*, its usage pattern [10]. When a battery is continuously discharged, a high current will cause it to provide less energy until the end of its lifetime than a lower current. This effect is termed the *rate-capacity effect*. On the other hand, during periods of low or no discharge current, the battery can recover to a certain extent. This is termed the *recovery effect*. These two effects are modeled in the *Kinetic Battery Model* (KiBaM) of Manwell and McGowan [17, 18, 19].

One approach to improve system lifetime is to connect one or more extra batteries, which are chosen following some schedule or scheduling policy. In most systems, the batteries are used sequentially, *i.e.*, only when one battery is empty the other is used. However, by switching back and forth between the two batteries one can make use of the recovery effect of the batteries and extend the overall system lifetime. Some research on battery scheduling has been done by Chiasserini *et al.* [9] and Benini *et al.* [7]. In these papers, several straightforward scheduling schemes, like round robin or choosing the best battery available, are compared. Although they do show that system lifetime can be extended by using battery scheduling, it is still unclear what the maximum possible lifetime is.

In this paper, we propose an approach to find *optimal* battery schedules for a given load. We model the behavior of batteries, based on a *discretized* version of the KiBaM, as *linear priced timed automata* (LPTA) [2, 4], and use the model checker Cora—which is a member of the UPPAAL family [22]—to generate the optimal schedules [5] using the well-developed model-checking techniques for LPTA. We show that the priced-timed automata model of the KiBaM is modeling the behavior of the rate-capacity and recovery effect faithfully. The generated optimal schedules show that

*The authors declare that the content of this paper has been cleared for publication by the authors respective institution.
This work is partially supported by EU-Project ICT-FP7-STREP 214755 (Quasimodo).

[¶]Corresponding Author.

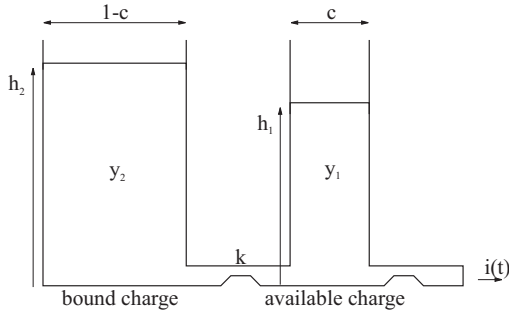


Figure 1. Schematic picture of the Kinetic Battery Model

in certain cases the round robin and best-battery scheduling policies come close to the maximal system lifetime, but are in some cases far from good.

The rest of this paper is structured as follows. In Section 2 we introduce the Kinetic Battery Model and propose a discretized version of it. In Section 3 we give background on priced-timed automata. Section 4 describes how the battery model is translated into the timed automata setting, and how the model can be used to find the optimal schedule. The battery model is validated in Section 5 in a one-battery setting. Results on multiple-battery scheduling are given in Section 6. We conclude in Section 7.

2 Kinetic battery model

2.1 Introduction

The battery model we use is the Kinetic Battery Model [17, 18, 19]. This model is very intuitive, and the simplest model that includes the two important non-linear battery properties, the rate-capacity effect and the recovery effect [16]. The rate-capacity effect is the effect that less charge can be drawn from the battery when the discharge current is increased. However, some of the charge left behind in the battery after a period with a high discharge current will be available for usage after a period with no or low current. This is the recovery effect.

In the KiBaM the battery charge is distributed over two wells: the available-charge well and the bound-charge well (*cf.* Figure 1). A fraction c of the total capacity is put in the available charge well, and a fraction $1 - c$ in the bound charge well. The available charge well supplies electrons directly to the load ($i(t)$), where t denotes the time, whereas the bound-charge well supplies electrons only to the available-charge well. The charge flows from the bound charge well to the available charge well through a “valve” with fixed conductance, k . Next to this parameter, the rate at which charge flows between the wells depends on the

height difference between the two wells. The heights of the two wells are given by: $h_1 = \frac{y_1}{c}$ and $h_2 = \frac{y_2}{1-c}$. The change of the charge in both wells is given by the following system of differential equations:

$$\begin{cases} \frac{dy_1}{dt} = -i(t) + k(h_2 - h_1), \\ \frac{dy_2}{dt} = -k(h_2 - h_1), \end{cases} \quad (1)$$

with initial conditions $y_1(0) = c \cdot C$ and $y_2(0) = (1 - c) \cdot C$, where C is the total battery capacity. The battery is considered empty when there is no charge left in the available charge well, *i.e.*, when $y_1(t) = 0$.

2.2 Coordinate transformation

Although the differential equations (1) nicely describe the discharge process of the battery, and an analytical solution can be obtained for constant discharge currents [17], the equations can be made more simple when a coordinate transform is applied. In this way even more insight can be obtained in the way the model behaves.

From (1), one can see that the height difference between the two wells ($h_2 - h_1$) plays a major role in the model. This is one of the coordinates after the transformation, the other is the total charge in the battery. So, the transformation changes the coordinates from y_1 and y_2 to $\delta = h_2 - h_1$ and $\gamma = y_1 + y_2$. This transformation changes the differential equations to:

$$\begin{cases} \frac{d\delta}{dt} = \frac{i(t)}{c} - k'\delta, \\ \frac{d\gamma}{dt} = -i(t), \end{cases} \quad (2)$$

where $k' = k/(1 - c)c$. The initial conditions change into: $\delta(0) = 0$ and $\gamma(0) = C$. In the new coordinate system the condition for the battery to be empty is:

$$\gamma(t) = (1 - c)\delta(t). \quad (3)$$

2.3 Discretization of the KiBaM

To be able to use the KiBaM in the timed automata setting, one needs to discretize the model, in particular time, charge, and height difference. We refer to the discretization of the KiBaM as the dKiBaM.

From (2) one can see that the total charge only changes when a current is drawn from the battery. The height difference changes by two processes: it increases when a current is drawn, and decreases when charge flows from the bound charge well to the available charge well. In the discretization of the model we separate these two processes.

We discretize time in steps of size T . Within a time step the discharge current is assumed constant. For a constant

current (I), the total charge will decrease linearly, cf. (2). The total charge is discretized in N parts of size $\Gamma = C/N$. It will take $\Gamma / (I \cdot T)$ time units to decrease the total charge with one charge unit at a discharge rate of I .

At the same time, the discharge with current I will increase the height difference with Γ/c . This will be the step size of the discretization of the height difference. Once some charge is drawn from the available charge well, charge will start to flow between the bound and available charge well. This is a non-linear process, described by the second part of the first equation in (2):

$$\frac{d\delta}{dt} = -k'\delta. \quad (4)$$

The solution to this differential equation is given by:

$$\delta(t) = \delta(0)e^{-k't}. \quad (5)$$

If at time point 0 we have $\delta = m \cdot \Gamma/c$, then the time t needed to decrease the height difference with one unit is

$$t = -\frac{1}{k'} \cdot \ln\left(\frac{m-1}{m}\right). \quad (6)$$

To obtain the number of time steps it takes to decrease the height difference by one unit, we divide this time t by T and round it to the nearest integer.

3 Priced-timed automata

In this section we informally describe *networks of linear priced timed automata* (NLPTA), as they are used as input to Cora[22], by means of an example. The example itself is of no relevance for battery scheduling, but illustrates nicely the important ingredients of NLPTA.

3.1 Networks of timed automata

The basic ingredients of NLPTA are *locations, switches, clocks, guards, invariants, assignments, and channels*. An NLPTA is composed of a collection of timed automata, which run in parallel and communicate via channels.

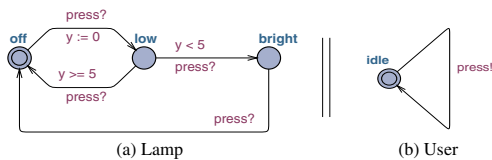


Figure 2. Model of lamp

Figure 2 (example from [3]) depicts a very simple NLPTA, comprising only two components. The network

models the behavior of a lamp. This behavior depends on when the on/off switch is pressed. In Figure 2(a) the behavior of the lamp is modeled. We see three locations **off**, **low**, and **bright**, which denote the three states the lamp can be in: in **off** the lamp is off, in **low**, it is on, giving a low light, and in **bright**, it shines brightly. Location **off** is the starting location. The switch from **off** to **low** (abbreviated **off**→**low**) is labeled with a receive operation (signified with the question mark) on channel **press**. This switch can only be taken if the timed automata in Figure 2(b) (a model of the user) executes the switch labeled with the corresponding send operation (signified with the exclamation mark) on channel **press**, i.e., both timed automata synchronize on channel **press**.

While **off**→**low** is executed, clock variable y is reset to 0 by the assignment $y := 0$. Clocks are real-valued variables which are used to measure time: clock values increase linearly with rate 1 as time progresses. Clocks are used to express enabling- and urgency-conditions depending on time. For example, the switch **low**→**off** is labeled with a *guard* $y >= 5$, which allows this switch to be taken only if clock y has a value of at least 5. In that case, if the user presses the button again, the lamp goes off. The switch **low**→**bright**, on the other hand, is guarded by expression $y < 5$, i.e., the negation of the previous guard. Thus, if the user presses the button a second time *within* 5 time units, the lamp switches to brighter light. From location **bright**, another button press will switch the lamp off again, unconditionally.

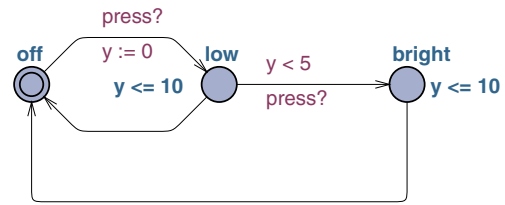


Figure 3. Model of automatic lamp

Invariants are used to express urgency conditions, i.e., unlike guards, which express when something *can* happen, they express when something *must* happen, without delay. In Figure 3 we see a slightly modified version of the lamp automaton. The lamp now switches off automatically, without user intervention. Switches **low**→**off** or **bright**→**off** are now unsynchronized. Instead, locations **low** and **bright** are now labeled with the invariant $y \leq 10$ each, which expresses that both locations can only be entered and occupied as long as y is at most 10. This means that the respective locations *must* be left 10 time units after the lamp was switched on *at the latest*, which in both cases means that the lamp goes either via **low**→**off** or **bright**→**off** to location **off**.

Using a synchronizing channel to model the communica-

tion for the automatic lamp is unrealistic in the sense that, if the lamp is in location `bright`, then the user cannot press the button (because the send operation `press!` has no corresponding receive operation). It is possible to declare a channel as a *broadcast channel*, which means that a send operation has to be synchronized with only those processes which are ready to execute a receive operation on that channel. If no process is ready to do so, the send operation can be executed anyway, with no effect on other processes. In our example, if `press` is declared a broadcast channel, the `press!` of the user can be executed even if the lamp is in state `bright`.

Switching on a lamp and letting it burn uses energy. Energy consumption can be modeled using costs. `Cora` provides the possibility to keep track of costs accumulated during the operation of the modeled system. For this purpose, there is a special variable `cost`, which can be increased explicitly during switching by an update, or implicitly by specifying a rate. Guards and invariants are, however, not allowed to refer to the cost variable. In Figure 4, `switch off → low` is labeled with update `cost += 50`, indicating that it takes 50 energy units to switch on the lamp. In locations `low` and `bright` we have the cost rates `cost' == 10` and `cost' == 20`, respectively, which indicate that the energy consumption is 10 and 20 units per time unit in the respective locations. When staying in these locations, `cost` is increasing linearly with time, with rate 10 and 20, respectively.

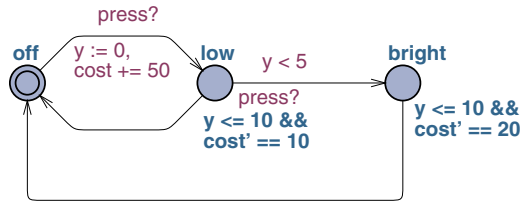


Figure 4. Model of automatic lamp with costs

In order to express prioritized execution of certain transitions, locations can be marked *Committed*. If one component is in a committed location, then the next switch to take must emanate from this committed location, and it must be taken without any delay.

It is possible to use simple data types in NLPTA like arrays and structures, based on integers, which can be declared either local to a single, or global to all automata. These data types can be manipulated in assignments, which are executed while switches are executed, and can be referenced in guards and—with some restrictions—also invariants. In our NLPTA battery model we will make extensive use of this feature.

3.2 Schedule generation

`Cora` is a model checker, *i.e.*, a tool to check whether the modeled NLPTA has certain properties which are expressed as logic formulae in a fragment of the *timed computation tree logic* [1]. Although the state space of a timed automaton is in general uncountable, precise finite abstractions of the state space exist, which make the model-checking problem decidable and feasible [1].

Timed automata models can be nondeterministic. A (timed) model checker like `Cora` can find paths—starting in the initial state—through the state space of a timed automaton to states which fulfill certain properties. These paths do thus resolve nondeterministic choices on the way to these target states. The idea of schedule generation with model checkers is to model the system to be scheduled (which is thus a combination of resources and load), but to leave the scheduling decisions open, *i.e.*, nondeterministic. If a certain scheduling objective can be formulated as a state property of this system, then the model checker can be employed to find such a state and the path leading to it; *the path is the schedule*. The scheduling objective can depend on timing properties (“look for the fastest schedule”), or, in the case of `Cora`, on the `cost` variable: optimization criterion is the minimality of the accumulated cost, *i.e.*, `Cora` tries to find the path with the lowest cost leading to the target state. This is what we will use to generate battery schedules.

4 A timed automata model for dKiBaM

In this section we introduce the network of timed automata used to model the dKiBaM of Section 2.3. We denote this model the TA-KiBaM.

4.1 Towards modeling

In the TA-KiBaM, the relevant information of the battery state is kept in arrays of integers (see also Table 1). We assume that each battery is uniquely identified by the constant *id*. We keep track of the *total remaining charge* and the *height difference between the two wells* of each battery in *numbers of charge units*. We introduce two arrays, `n_gamma` and `m_delta`, of the size equal to the number of batteries. `n_gamma[id]` is the total charge left in battery *id*, `m_delta[id]` the height difference between the two wells of battery *id*. Thus, initially `n_gamma[id] = N` and `m_delta[id] = 0`.

The recovery characteristics of a battery are described by the array `recov_times`. The contents of this array is pre-computed using (6). If the height difference of the battery is `m_delta[id]`, then `recov_times[m_delta[id]]` is the time it takes to decrease the height difference by one charge unit.

`recov_times` is independent of the load put on the battery. The load is described by three arrays of equal length.

- In array `load_time`, the times are stored when the load changes. The times are defined absolutely, counted from system start, *i.e.*, `load_time` contains a strictly increasing sequence of numbers. This array defines certain *epochs* of the battery usage period, where `load_time[y]` is the time where epoch y ends and epoch $y + 1$ starts.
- The array `cur_times` has the same size as `load_time` and gives the number of time units it takes to draw some charge units from the battery during the epochs.
- The array `cur` has the same size as `load_time` and defines together with `cur_times` the current drawn from the battery during an epoch. The array gives the number of charge units drawn from the battery during one of the periods defined in `cur_times`. The current (I_y) drawn from the battery during epoch y is given by the equation:

$$I_y = \frac{\text{cur}[y]\Gamma}{\text{cur_times}[y]T}. \quad (7)$$

The three arrays are created using an external program, and imported into the TA-KiBaM.

The criterion when a battery is empty is given by (3). The fact that we use charge units as the central measure makes it necessary to rephrase this criterion to be usable in the NLPTA model. If n is the total number of charge units left in the battery and m the height difference in number of charge units at time t , then $\gamma(t) = n\Gamma$, $\delta(t) = m\Gamma/c$, hence $\gamma(t) = (1 - c)\delta(t)$ can be transformed to $cn = (1 - c)m$. We change the equality sign to a less-than-or-equal-to sign, to account for errors due to the discretization:

$$cn \leq (1 - c)m. \quad (8)$$

4.2 Basic battery model

The discharge and recovery behavior of the dKiBaM is modeled by two timed automata, which are depicted in Figure 5. Figure 5(a) shows the *total charge* automaton, and Figure 5(b) the *height difference* automaton for battery `id`. The total charge automaton keeps track of the level of the total charge in battery `id` (`n_gamma[id]`). The automaton starts in location `idle`. When the battery is used, signaled on channel `go_on`, the automaton will switch to location `on` and the clock `c_disch` is reset. The global variable `j` (modified in another component) is the current epoch, and index to the load defining arrays `load_time`, `cur_times`, and `cur`. After `cur_times[j]` time units, a number of `cur[j]` charge units is subtracted from `n_gamma[id]`

Var	Type	Description
<code>id</code>	int	unique number for each battery
<code>n_gamma</code>	array	the current total charge for each battery, measured in charge units, initially N (<i>cf.</i> Section 2.3)
<code>m_delta</code>	array	the current height difference for each battery, initially zero
<code>load_time</code>	array	times when epoch ends (precomputed)
<code>cur_times</code>	array	times it takes to discharge <code>cur</code> charge unit (precomputed)
<code>cur</code>	array	number of charge units consumed within <code>cur_times</code> (precomputed)
<code>j</code>	int	current epoch, and thus index to arrays <code>load_time</code> , <code>cur_times</code> , and <code>cur</code>
<code>recov_time</code>	array	times it takes to recover one charge unit (precomputed)

Table 1. Important variables

and the total charge automaton synchronizes with the height difference automaton through the `use_charge` channel to increase `m_delta[id]` with `cur[j]` units.

The battery is used until it is empty. In the total charge automaton this is checked using the guard $(1000-c)m_delta[id] \geq c*n_gamma[id]$, which is exactly condition (8). When this inequality holds the battery `id` is empty, and the total charge automaton will go to location `empty`. The height difference automaton keeps track of the height difference between the two wells, `m_delta[id]`. Initially, `m_delta[id]` is zero. For every charge unit drawn from the battery, `m_delta` is increased with one unit. When `m_delta[id] > 1`, recovery starts and `m_delta[id]` is decreased by the automaton. The time it takes to recover one unit depends on `m_delta[id]`. The times are precomputed, using (6) and stored in the array `recov_times`. Thus, after `recov_times[m_delta[id]]`, `m_delta[id]` is decreased by one unit. The clock `c_recov` is used to check whether the appropriate time has passed to recover a unit.

4.3 Battery scheduling

In the load automaton (Figure 5(c)) the array `load_time` is used to determine the start and end times of the different epochs in the load. The array `cur` is used to determine whether an epoch contains a job (`cur[j] > 0`) or an idle period (`cur[j] = 0`). When the epoch contains a job

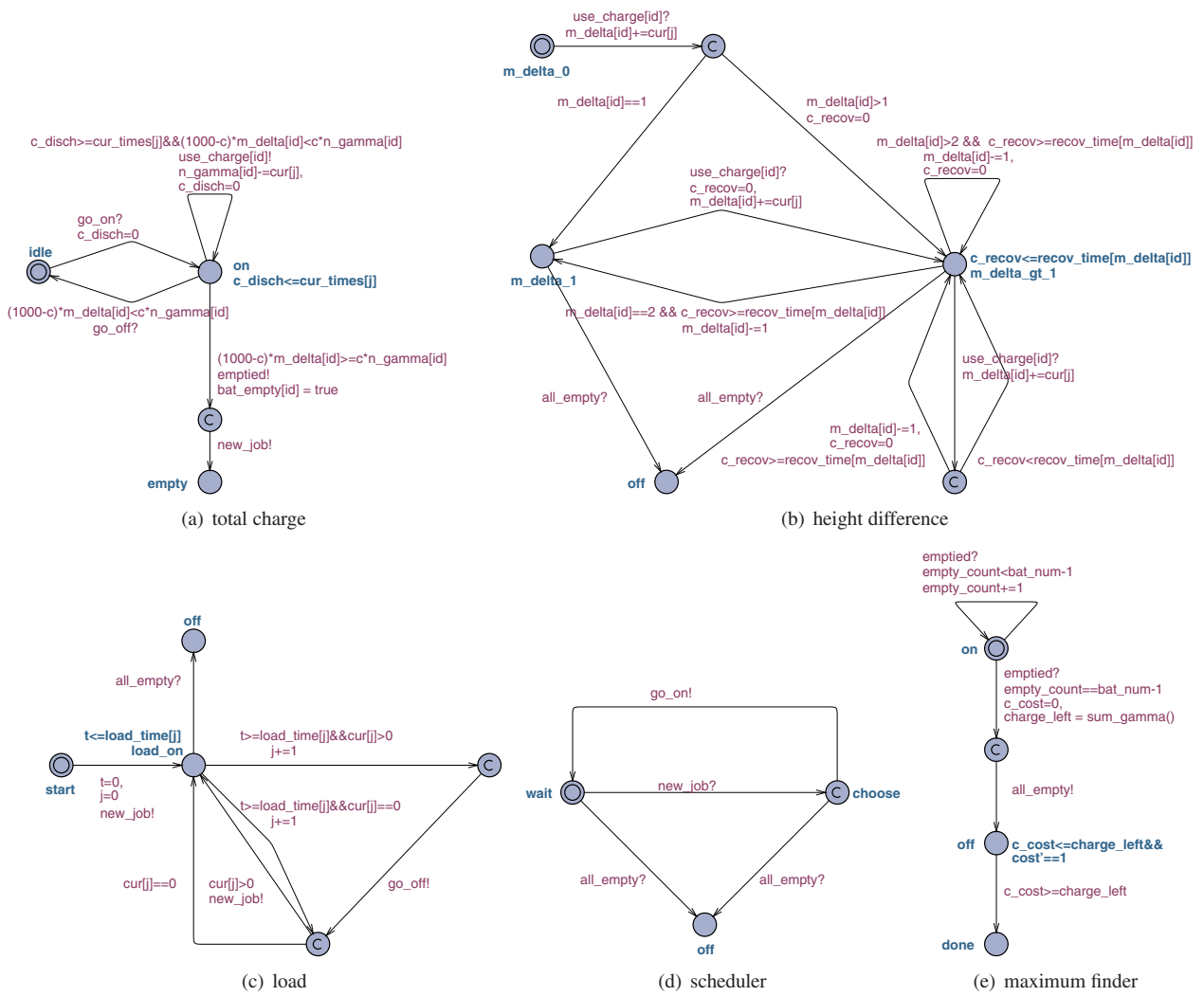


Figure 5. Timed automata for battery scheduling

the load automaton will synchronize with the scheduler automaton at the start of the epoch through the `new_job` channel to schedule the next battery. At the end of the job the load automaton synchronizes through the `go_off` channel with the assigned battery to switch it off. These two synchronizations are not needed for idle periods, since no battery is used. When the load automaton synchronizes with the scheduler automaton (Figure 5(d)) through the `new_job` channel, the scheduler will nondeterministically synchronize through the `go_on` channel with one of the batteries. When a battery is emptied during a job, the scheduler chooses another battery to continue the job at the point the other stopped. Although the battery can still recover some charge, we assume it can not be used anymore once observed empty. The optimal schedule, *i.e.*, the one that yields the longest system lifetime, is the schedule that takes the longest for both batteries to reach the location `empty`.

We want the model checker to find the longest path to this state. Unfortunately, `Cora` is not able to find the longest path leading to a target state. It is thus necessary to translate the question for the longest that leads to the maximum lifetime into a question for a minimum cost. This has been done by adding the *maximum finder* automaton (Figure 5(e)). When a battery is empty, the total charge automaton will signal the maximum finder automaton on channel `emptied`. The maximum finder automaton counts the number of empty batteries. When all batteries are empty the maximum finder automaton broadcasts on the channel `all_empty` to stop all processes in the other automata. Now, the maximum finder automaton converts the charge remaining in the bound charge well of the batteries into a cost. The path that will lead to the longest system lifetime will have used the most charge from the batteries and therefore will have the smallest amount of charge remaining in the bound

channel	sending automata	receiving automata	action
new_job	load, total charge	scheduler	call the scheduler to make a scheduling decision when a new job starts or a battery is empty
go_on	scheduler	total charge	switch the battery chosen by the scheduler to on
use_charge	total charge	height difference	increase the height difference with one unit for every charge unit drawn from the battery
emptied	total charge	maximum finder	add one to the number of batteries that are empty
all_empty	maximum finder	height difference, load, scheduler	stop all battery processes when all batteries are empty

Table 2. Overview of the synchronization channels used in the TA-KiBaM.

charge wells. In Table 2 all used synchronization channels are listed to give an overview of the interactions between the different automata.

We use thus *Cora* to check the simple TCTL property $A[] \text{ not max.done}$. This property is not satisfied, and *Cora* returns, with appropriately chosen options, a path as a counterexample which minimizes the cost and maximizes the system lifetime.

4.4 Complexity

The *complexity* of finding the optimal schedule depends exponentially on the number of scheduling decisions that has to be made, where the number of batteries (B) is the base. At every scheduling point one can choose between all B batteries. The number of scheduling points depend on the battery's capacity and the load applied.

Between the scheduling points the model is fully deterministic. The number of states in between two scheduling points will depend on the granularity of the discretization. The maximum number of state changes that can be made due to discharging is $N = C/\Gamma$, when all charge units are drained one at a time. The maximum number of state changes due to recovery is not so easy to define. However, it will be proportional to $1/\Delta$. Since in the model, $\Delta = \Gamma/c$, this maximum number of states will be proportional to $1/\Gamma$. The discretization of time will not influence the maximum number of states. Introducing smaller time steps will only increase the number of time steps it takes to change states.

5 Validation of the TA-KiBaM

To validate the TA-KiBaM, we compare the battery lifetimes computed with this model and with the original KiBaM. The KiBaM itself has been validated in [16], where it is compared to the very precise electro-chemical model Dualfoil [11, 14, 13].

test load	lifetime KiBaM (min)	lifetime TA-KiBaM (min)	difference %
CL_250	4.53	4.56	0.7
CL_500	2.02	2.04	1.0
CL_alt	2.58	2.60	0.8
ILs_250	10.80	10.84	0.4
ILs_500	4.30	4.32	0.5
ILs_alt	4.80	4.82	0.4
ILs_r1	4.72	4.74	0.4
ILs_r2	4.72	4.74	0.4
ILl_250	21.86	21.88	0.1
ILl_500	6.53	6.56	0.5

Table 3. Battery lifetimes of battery B_1 for the different loads computed with both the analytical KiBaM and the TA-KiBaM.

We consider a single-battery case, and use two different battery types, one with capacity 5.5 Amin (Ampere-minute) (B_1) and one with capacity 11 Amin (B_2). The battery parameters are the same for both batteries: $c = 0.166$ and $k' = 0.122 \text{ min}^{-1}$ [15], corresponding to the lithium-ion battery used in the Itsy pocket computer, which was also simulated by Rakhmatov *et al.* [20, 21].

The time step size is set to 0.01 min. The total charge is discretized in steps of size 0.01 Amin. This leads to the discretization step size of the height difference of $0.01/c = 0.06$ Amin.

The Itsy pocket computer operates with currents up to 700 mA. In the tests we used two types of jobs, a low current job (250 mA) and a high current job (500 mA). With these jobs, ten different test loads have been created:

- three continuous loads (CL) with no idle periods between the jobs: one load with only low current jobs

(CL₂₅₀), one with only high current jobs (CL₅₀₀), and one alternating between a low current job and a high current job (CL_{alt}).

- five intermitted loads with short idle periods of one minute between the jobs (ILs): one with only low current jobs (ILs₂₅₀), one with only high current jobs (ILs₅₀₀), one alternating between a low current job and a high current job (ILs_{alt}), and two where the job is randomly chosen (ILs_{r1} and ILs_{r2})
- two intermitted loads with long idle periods of two minutes between the jobs (ILℓ): one with only low current jobs (ILℓ₂₅₀) and one with only high current jobs (ILℓ₅₀₀).

The results of the tests for the two battery types are given in Table 3 and 4. For most loads the lifetime in the timed automaton battery model is only between 0.02 and 0.04 min longer for the analytical KiBaM. For the loads CL₂₅₀ and CL_{alt} TA-KiBaM gives a bigger difference for battery B_2 . This bigger difference is due the discretization of the height difference and the time needed to recover one height unit. When a battery is discharged, the height difference can grow up to the point that the time to increase the height difference with one unit equals the time to decrease the height difference with one unit, *i.e.*, $cur_times[x] = recov_time[y]$. The $recov_time$ is computed according to (6) and rounded to the nearest number of time steps. The rounding causes the height difference to grow less than it does in the analytical version of the model. The amount of charge that is unavailable when the battery is empty is proportional to the height difference, so the smaller height difference will give a longer lifetime. The moment the height difference does not increase anymore is only reached with the loads CL₂₅₀ and CL_{alt} applied to B_2 . In all other cases, either the battery lifetime is too short for the height difference to reach this point, or the load has idle periods in which the height difference will decrease. Note that even for the loads with the bigger differences, the relative difference is still only 1%.

Although the discretized model sometimes gives small differences in lifetime computation compared to the analytical model, we regard it perfectly usable for attacking the scheduling problem.

6 Scheduling results

We use the multiple battery timed automaton model to find the optimal way to schedule two batteries on the same test loads as used in Section 5. We use batteries of type B_1 .

Next to computing the maximum lifetime, we used the TA-KiBaM to compute the lifetime using three deterministic scheduling schemes. Hence, we compare four schedules:

test load	lifetime KiBaM (min)	lifetime TA-KiBaM (min)	difference %
CL ₂₅₀	12.16	12.28	1.0
CL ₅₀₀	4.53	4.54	0.2
CL _{alt}	6.45	6.52	1.1
ILs ₂₅₀	44.78	44.80	0.04
ILs ₅₀₀	10.80	10.84	0.4
ILs _{alt}	16.93	16.94	0.1
ILs _{r1}	22.71	22.74	0.1
ILs _{r2}	14.81	14.84	0.2
ILℓ ₂₅₀	84.90	84.92	0.02
ILℓ ₅₀₀	21.86	21.88	0.1

Table 4. Battery lifetimes of battery B_2 for the different loads computed with both the analytical KiBaM and the discretized timed automata KiBaM.

- Sequential schedule. The batteries are used sequentially, *i.e.*, the second battery is only chosen when the first one is empty.
- Round robin schedule. For every new job a new battery is chosen. The batteries are chosen in a fixed order.
- Best-of-two schedule. At the start of a job, the status of the batteries is checked. The battery with the most charge in the available charge well is chosen to supply the charge for the job.
- Optimal schedule. The schedule computed using Cora.

The computed lifetimes are given in Table 5, along with the relative difference to the lifetime of the round robin scheduler. For the test loads, one can see the order in performance of the different scheduling schemes. One can easily show, using the Cora model, that the sequential scheduling is actually the worst possible way to schedule the batteries. For the test loads the round robin and best-of-two scheme differ only in the cases of the alternating jobs. These cases are clearly very bad for the round robin scheme, since the heavy load is always put onto the same battery. This battery will be empty very fast, and then only one battery is left to handle all of the remaining load, leaving this battery with less idle time to recover. The best-of-two scheme balances the load better over the two batteries, which leads to a longer lifetime, especially in the ILs_{alt} case. In the other cases the best-of-two scheme behaves exactly like the round robin scheme. Although the round robin and best-of-two schedulers perform close to optimal in most cases, for some loads the schedules are far from optimal. The optimal scheduler yields lifetime improvements up to 32%.

test load	sequential		round robin	best-of-two		optimal	
	lifetime (min)	difference %	lifetime (min)	lifetime (min)	difference %	lifetime (min)	difference %
CL_250	9.12	-21.4	11.60	11.60	0	12.04	3.79
CL_500	4.10	-9.5	4.53	4.53	0	4.58	1.1
CL_alt	5.48	-10.2	6.10	6.12	0.3	6.48	6.2
ILs_250	22.80	-41.5	38.96	38.96	0	40.80	4.7
ILs_500	8.60	-17.9	10.48	10.48	0	10.48	0
ILs_alt	12.38	-3.4	12.82	16.30	27.2	16.91	31.9
ILs_r1	12.80	-21.28	16.26	16.26	0	20.52	26.2
ILs_r2	12.24	-15.59	14.50	14.50	0	14.54	0.3
ILl_250	45.84	-39.7	76.00	76.00	0	78.96	3.9
ILl_500	12.94	-18.9	15.96	15.96	0	18.68	17.0

Table 5. System lifetime computed for all test loads according to the four scheduling schemes. Next to the values of the lifetimes, the difference relative to the round robin scheme are given.

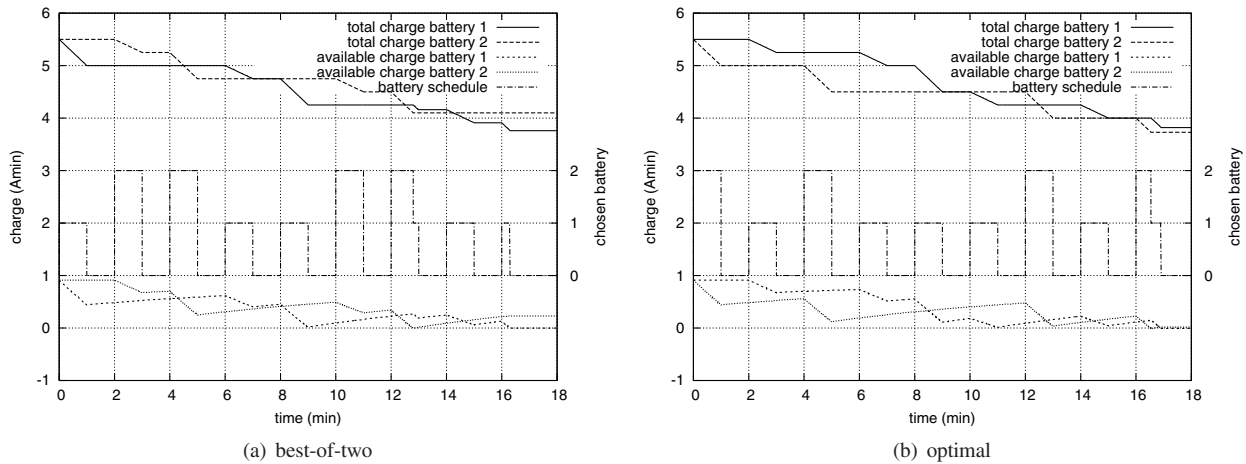


Figure 6. The schedules and the total and available charge in the batteries for the best-of-two (a) and the optimal (b) schedule for the ILs_alt load.

Besides the system lifetimes, the Cora evaluation of the timed automata battery model also provides the actual schedules which lead to these lifetimes, as well as the evolution of the charge in the battery. Figure 6 shows the evolution of the total and available charge in the two batteries (left y -axis) for both the best-of-two scheduler (Figure 6(a)) and the optimal scheduler (Figure 6(b)), in the ILl.alt case. In the figure, also the two schedules (right y -axis) are shown. When a battery is chosen, one can see the total and available charge decrease due to the load. The slope of the curves is proportional to the discharge current. When a battery is not used, one can see the available charge rise again. This is due to the recovery effect. Note that, when the batteries are empty, still a relatively large amount of charge remains in the battery (approximately 3.9 Amin, which is 70% of its

original energy).

Due to the complexity of finding the optimal schedule, it is possible to model only a limited total battery capacity. The used discharge currents are relatively high for the battery's capacity, and will drain the available charge well fast. Therefore, there will be little time for the bound charge to become available, and a large fraction will remain unused. When the battery capacity is increased this fraction will be smaller. Using the deterministic scheduling scheme, we can compute the results for larger capacities. For example, with a ten times larger capacity, the fraction of charge left behind in the batteries will be less than 10% in the case of best-of-two scheduling.

When we look at the two schedules in Figure 6, we see that the best-of-two schedule acts like a round robin sched-

uler that switches batteries after the high current jobs. The optimal schedule does not show a regular pattern; further research is needed here. The optimal schedule does depend strongly on the size of the batteries and their parameters, as well as on the load that is applied.

7 Conclusions and outlook

We have shown a new approach to maximizing the system lifetime through battery scheduling. The priced-timed automaton battery model allows us to compute the optimal way to schedule multiple batteries for a given load. The optimal schedule and lifetime can easily be compared to straightforward scheduling schemes, like round robin. For most of the tested loads the round robin and best-of-two schedulers perform close to optimal. However, the optimal schedules do show that there still is room for improvement.

The optimal scheduler can only be used in real life systems, when the load function is known in advance. However, it does help in finding a better way of using the available battery capacity. Next to the test loads as used here, realistic random loads need to be analyzed. However, Uppaal Cora does not allow for probabilities to be included in the models. For this, one does need probabilistic (priced) timed automata. However, no tools are available yet to do this, although algorithms are in development [8].

Besides multiple battery scheduling, we also want to use the timed automata battery model for another optimization problem. For a device with one battery and a given workload, we want to know how to schedule the jobs over time to optimize the battery lifetime. This could, for example, be used in nodes in sensor networks, which have simple regular workloads.

Next to the timed automata models for optimizing the battery lifetime, we want to investigate the usability of AI planning [12] to solve the scheduling problems.

References

- [1] R. Alur, C. Courcoubetis, and D. Dill. Model-checking in dense real-time. *Information and Computation*, 104(2):2–34, 1993.
- [2] R. Alur, S. L. Torre, and G. J. Pappas. Optimal paths in weighted timed automata. In Benedetto and Sangiovanni-Vincentelli [6], pages 49–62.
- [3] G. Behrmann, A. David, and K. Larsen. A tutorial on Uppaal. In *Formal Methods for the Design of Real-Time Systems*, volume 3185 of *LNCS*, pages 200–236. Springer Verlag, 2004.
- [4] G. Behrmann, A. Fehnker, T. Hune, K. G. Larsen, P. Pettersson, J. Romijn, and F. W. Vaandrager. Minimum-cost reachability for priced timed automata. In Benedetto and Sangiovanni-Vincentelli [6], pages 147–161.
- [5] G. Behrmann, K. G. Larsen, and J. I. Rasmussen. Optimal scheduling using priced timed automata. *ACM SIGMETRICS Performance Evaluation Review*, 32(4):34–40, 2005.
- [6] M. D. D. Benedetto and A. L. Sangiovanni-Vincentelli, editors. *Proc. HSCC 2001*, volume 2034 of *LNCS*. Springer, 2001.
- [7] L. Benini, G. Castelli, A. Macii, E. Macii, M. Poncino, and R. Scarsi. Extending lifetime of portable systems by battery scheduling. In *Design, Automation and Test in Europe*, pages 197–203. IEEE CS Press, 2001.
- [8] J. Berendsen, D. N. Jansen, and J.-P. Katoen. Probably on time and within budget – on reachability in priced probabilistic timed automata. In *Proc. QEST '06*. IEEE CS Press, 2006.
- [9] C. Chiasserini and R. Rao. Energy efficient battery management. *IEEE J. Sel. Areas in Com.*, 19(7):1235 – 1245, 2001.
- [10] L. Cloth, B. R. Haverkort, and M. R. Jongerden. Computing battery lifetime distributions. In *Proc. DSN 2007*, pages 780–789. IEEE CS Press, 2007.
- [11] M. Doyle, T. F. Fuller, and J. Newman. Modeling of galvanostatic charge and discharge of the lithium/polymer/insertion cell. *Journal of the Electrochemical Society*, 140(6):1526 – 1533, 1993.
- [12] M. Fox and D. Long. Modelling mixed discrete-continuous domains for planning. *Journal of AI Research*, 27:235–297, 2006.
- [13] T. F. Fuller, M. Doyle, and J. Newman. Relaxation phenomena in lithium-ion-insertion cells. *Journal of the Electrochemical Society*, 141(4):982 – 990, 1994.
- [14] T. F. Fuller, M. Doyle, and J. Newman. Simulation and optimization of the dual lithium ion insertion cell. *Journal of the Electrochemical Society*, 141(1):1 – 10, 1994.
- [15] M. R. Jongerden and B. R. Haverkort. Battery modeling. Technical Report TR-CTIT-08-01, CTIT, University of Twente, 2008.
- [16] M. R. Jongerden and B. R. Haverkort. Which battery model to use? In *Proc. UK-PEW 2008*, Tech. Report DComp., Imperial College London, pages 76–88, 2008.
- [17] J. Manwell and J. McGowan. Lead acid battery storage model for hybrid energy systems. *Solar Energy*, 50:399–405, 1993.
- [18] J. Manwell and J. McGowan. Extension of the kinetic battery model for wind/hybrid power systems. In *Proc. EWEC '94*, pages 284–289, 1994.
- [19] J. Manwell, J. McGowan, E. Baring-Gould, S. W., and A. Leotta. Evaluation of battery models for wind/hybrid power system simulation. In *Proc. EWEC '94*, pages 1182–1187, 1994.
- [20] D. Rakhmatov, S. Vrudhula, and D. A. Wallach. Battery lifetime predictions for energy-aware computing. In *Proc. ISLPED '02*, pages 154–159, 2002.
- [21] D. Rakhmatov, S. Vrudhula, and D. A. Wallach. A model for battery lifetime analysis for organizing applications on a pocket computer. *IEEE Trans. on VLSI Systems*, 11(6):1019–1030, 2003.
- [22] UPPAAL webpage. www.uppaaal.com.