Power management in energy harvesting embedded systems

Diss. ETH No. 18247

Power Management in Energy Harvesting Embedded Systems

A dissertation submitted to the

SWISS FEDERAL INSTITUTE OF TECHNOLOGY
ZURICH

for the degree of

Doctor of Sciences

presented by

CLEMENS MOSER
Dipl.-Ing. B.Sc. TU Munich

born 13.02.1979
citizen of
Switzerland and Germany

accepted on the recommendation of

Prof. Dr. Lothar Thiele, examiner
Prof. Dr. Rolf Ernst, co-examiner

2009

Institut für Technische Informatik und Kommunikationsnetze
Computer Engineering and Networks Laboratory

TIK-SCHRIFTENREIHE NR. 103

Clemens Moser

Power Management in Energy Harvesting Embedded Systems

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

A dissertation submitted to the
Swiss Federal Institute of Technology (ETH) Zürich
for the degree of Doctor of Sciences

Diss. ETH No. 18247

Prof. Dr. Lothar Thiele, examiner
Prof. Dr. Rolf Ernst, co-examiner

Examination date: 13. March, 2009

Abstract

Energy harvesting (also known as energy scavenging) is the process of generating electrical energy from environmental energy sources. There exists a variety of different energy sources such as solar energy, kinetic energy, or thermal energy. The term has been frequently applied in the context of small autonomous devices such as wireless sensor nodes.

This thesis addresses power management in energy harvesting embedded systems. As an example scenario, we focus on wireless sensor nodes which are powered by solar cells. We demonstrate that classical power management solutions have to be reconceived and/or new problems arise if perpetual operation of the system is required. In particular, we provide a set of algorithms and methods for different application scenarios, including real-time scheduling, application rate control as well as reward maximization. Goal is to optimize the performance of the application subject to given energy constraints. Compared to state-of-the-art approaches, our methods optimize the system performance or achieve the same performance as state-of-the-art approaches requiring, e.g., smaller solar cells and smaller batteries. Furthermore, we show how to dimension important system parameters like the minimum battery capacity or a sufficient prediction horizon. Our theoretical results are supported by simulations using long-term measurements of solar energy in an outdoor environment. Furthermore, to demonstrate the practical relevance of our approaches, we measured the implementation overhead of our algorithms on real sensor nodes.

Zusammenfassung

Als Energy Harvesting (oder Energy Scavenging) bezeichnet man die Erzeugung elektrischer Energie aus z.B. Sonnenenergie, Vibrationen oder Umgebungstemperatur. Der Begriff wird häufig im Zusammenhang mit miniaturisierten, eingebetteten System verwendet. Entsprechende Energiewandler machen besonders dann Sinn, falls drahtlose Technologien zum Einsatz kommen oder Batterien als Energiequellen keine ausreichende Betriebsdauer garantieren.

Die vorliegende Arbeit leistet einen Beitrag zur Optimierung eingebetteter Systeme welche ihre Energie mittels Energy Harvesting aus der Umgebung entnehmen. Typischerweise werden Batterien in solchen Systemen nicht mehr als primäre Energiequellen, sondern nur noch als Zwischenspeicher verwendet. Am Beispiel von solargetriebenen, drahtlosen Sensorknoten wird gezeigt, wie geeignete Algorithmen die Energieversorgung langfristig optimieren können. Ziel dieser Optimierung ist es, unterbrechungsfreien Betrieb des Systems zu gewährleisten und gleichzeitig die verfügbare Energie möglichst sinnvoll zu nutzen. Zu diesem Zweck werden in dieser Arbeit verschiedene Anwendungsszenarien untersucht. Darüber hinaus werden Methoden zur Berechnung der notwendigen Speicherkapazität der Batterie vorgestellt, welche für einen unterbrechungsfreien Betrieb notwendig ist. Die entworfenen Algorithmen wurden teilweise auf Sensorknoten implementiert und getestet. Zudem haben Simulationen gezeigt, dass unsere Algorithmen die langfristige Energieversorgung im Vergleich zu State-of-the-Art Lösungen deutlich verbessern. Die durchgeführten Simulationen basieren auf Messdaten solarer Energie, die in Langzeitmessungen von einer Photovoltaikanlage aufgezeichnet wurden. Auf der Grundlage dieser Messdaten wurden unsere Algorithmen ausgiebig getestet.

Acknowledgement

This thesis would not have been possible without the help and support of many people to whom I would like to express my gratefulness. In the first place, I would like to thank my advisor Professor Dr. Lothar Thiele. Your constant support and patience was always very motivating, and the many fruitful discussions inspired much of the work presented in this thesis. I have benefited a lot from your experience and I am very glad that I got the chance to join your research group.

I would also like to express my gratitude to Professor Dr. Rolf Ernst for being my co-examiner. Thanks for investing the time to read through the thesis and for your positive comments.

Furthermore, I thank Dr. Jian-Jia Chen. Your enthusiasm and vast knowledge has always been a great source of inspiration. Thank you for the very fruitful research cooperation and for your friendship!

I thank all my current and former colleagues of the TEC group for their company and support. In particular, I would like to thank Kai Huang for the great time we had while sharing our office in the last years.

Last but not least, my dearest thanks go to my parents Maria and Rainer, my brother Vincent, and my girlfriend Karin, for their love and support throughout all these years of my education.

The work presented in this thesis was supported by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005- 67322. In addition, this research has been supported by grants from the European Network of Excellence Artist2 and ArtistDesign. This support is gratefully acknowledged.

Contents

1

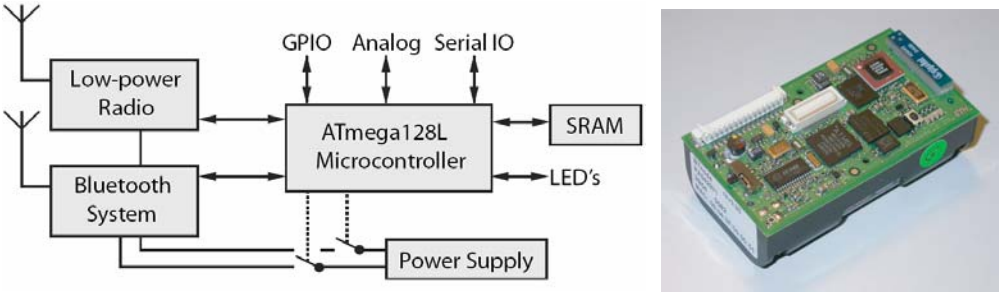
Introduction

This thesis presents a set of novel power management solutions for energy harvesting embedded systems. In particular, in the field of wireless sensor networks, energy harvesting techniques are currently gaining momentum. Therefore, we will focus on wireless sensor nodes as a primary application scenario throughout this thesis. Nevertheless, our methods generally apply to all kinds of embedded systems which take advantage of regenerative energy sources to achieve long-term operation.

In Section 1.1, we start with a short survey of application scenarios and the design space of wireless sensor networks. In Section 1.2, we discuss the benefits of energy harvesting and elaborate on the new challenges which have to be mastered. A discussion of various environmental energy sources is provided in Section 1.3. Finally, we give the outline and summarize the contributions of this thesis in Section 1.4.

1.1 Wireless Sensor Networks

Wireless sensor networks (WSN) have opened up an exciting field of research that is increasingly becoming popular nowadays. A WSN can be seen as a system of self-powered, wireless sensors which are able to detect and transmit events to a base station. WSNs are deployed wherever it is not possible or practical to maintain a wired network infrastructure. Main applications of WSNs are, e.g., the monitoring of environmental physical quantities such as temperature, humidity or vibrations as well as physiological monitoring, smart spaces or factory instrumentation.



(a) BTnode rev3 hardware overview. (b) The BTnode sensor node.

Fig. 1: A state of the art sensor node: The BTnode.

In the recent years, a great number of prototype sensor networks have been deployed, including networks for volcano monitoring [WALJ+06], habitat monitoring [SPMC04] or glacial movement monitoring [MOH04]. For most of these deployments, unattended operation of the network for long periods is highly desirable.

In the last decade, a large number of sensor network platforms both from academia and from industry have emerged, including, e.g., the BTnode [BDH+04], the Moteiv Tmote Sky [PSC05], the Crossbow Mica2 [HHKK04] and the Intel Imote [NKA+05]. In Figure 1, the BTnode rev3 sensor node is displayed. It is introduced here since it can be seen as a state of the art sensor network platform and it has been us for the experiments in this thesis. The BTnode system core consists of an Atmel ATmega128l microcontroller, two radios for wireless communication and SRAM memory. As standard power supply, 2-cell AA batteries are used.

As for many other battery-operated embedded systems, a sensor's operating time is a crucial design parameter. As electronic systems continue to shrink, however, less energy is storable on-board. Research continues to develop higher energy-density batteries and supercapacitors, but the amount of energy available still severely limits the system's lifespan. As a result, size as well as weight of most existing sensor nodes are largely dominated by their batteries.

A comprehensive overview of the design space of sensor networks is given in [RM04]. For many application scenarios, the sensor nodes are anticipated to be small and inexpensive devices which can be unobtrusively embedded in their environment. Thus, a sensor node's hardware is stringently limited in terms of computation, memory, communication as well as storable energy (e.g. batteries). These hardware constraints also limit the complexity of the software executed on a sensor node.

1.2 Energy Harvesting in Sensor Networks

One of the main advantages of WSNs is their independence of pre-established infrastructure. That is, in most common scenarios, recharging or replacing nodes' batteries is not practical due to (a) inaccessibility and/or (b) sheer number of the sensor nodes. In order for sensor networks to become a ubiquitous part of our environment, alternative power sources should be employed. Therefore, environmental *energy harvesting* is deemed a promising approach: If nodes are equipped with energy transducers like, e.g., solar cells, the generated energy may increase the autonomy of the nodes significantly.

Benefits

The past several years have seen an increasing interest in wireless sensor nodes which are scavenging energy from their environment. Specifically, techniques to harvest energy via photovoltaic cells have attracted the interest of the sensor network community [RKH+05]. Solar energy is certainly one of the most promising energy sources and typical environmental monitoring applications have access to solar energy. Due to the progress in low-power design, the energy generated by small solar panels suffices to execute most common data gathering applications. Equipped with photovoltaic cells, *perpetual operation* becomes possible without frequent recharging and replacement of the batteries. Ideally, sensor nodes once deployed in a harsh environment benefit from a drastically increased operating time and become virtually immortal. Since batteries are solely used as energy buffers and not as primary energy sources, the cost, weight and size of batteries can be reduced significantly.

Challenges

The overhead of the energy harvesting *hardware* as well as demand for corresponding *software* control can be seen as disadvantages compared to battery driven systems. Concerning the hardware, an energy harvesting device like a solar panel or a piezoelectric element has to be integrated into the system. Besides the actual harvesting device, dedicated power supply circuits are required to efficiently charge the battery. This thesis, however, focuses on the design of *adaptive software solutions* for energy harvesting systems. Here, carefully designed power management algorithms must be applied in order to avoid wasting precious energy. The energy required for sophisticated control algorithms may introduce a high control overhead for low-power applications. For sensor nodes which periodically sense and transmit data, but spend most of the time in power-saving sleep modes, simple, low-complexity solutions are needed.

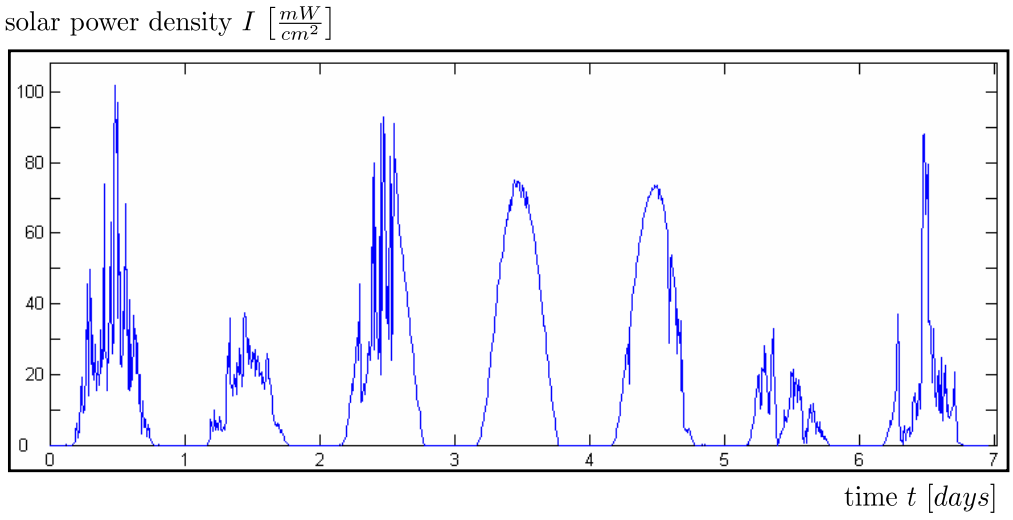


Fig. 2: Recording of solar light intensity during one week [Sun06].

From a networking perspective, classical sensor network protocols cannot harness the full potential provided by the harvesting technology. Here, some attempts have been made to make routing or clustering decisions within the network *harvesting aware* [LSR05, VRS+04]. Based on the knowledge of the currently generated power at the single nodes, the network lifetime can be optimized by shifting the communication and computation load. In the example of a solar powered network, nodes which are directly exposed to sunlight have to disburden nodes who are harvesting less energy due to shadowing effects.

In contrast, this thesis focuses on the *temporal* variations of the energy source experienced by a single node instead of *spatial* variations between several nodes. The obtained results can, e.g., be applied to networks, whose nodes are independently from each other transmitting data to a base station. The temporal variations of solar energy harvested by a sensor node in an outdoor environment is exemplified in Figure 2. The figure displays recordings of solar light intensity during one week.

It becomes evident that solar energy is an *infinite energy* source, compared to conventional batteries, which are *finite energy* sources. On the other hand, the harvested *power* can only be regarded as constant on average in a *long-term* perspective. On a *short-term* perspective, however, the harvested power is highly unstable, as shown in Figure 2. While the forth and fifth day turn out to be quite sunny, for instance, only little energy is harvested at the second and sixth day.

A prerequisite for any reasonable power management is an *energy prediction* of the energy harvested in the future. For the example of solar energy in Figure 2, the wheather forecast may be of help. In the course of this thesis, we will explore different methods for energy prediction.

Subsequently, the actual power management decides on the future use of energy based on the input of the energy prediction.

Sensor nodes executing a given application may frequently run out of energy in times with insufficient illumination. In the example of Figure 2, this may happen during the cloudy days number 2 and 6 or during the nights. If one strives for predictable, continuous operation of a sensor node, common power management techniques have to be reconceived. In addition to perform classical power saving techniques, the sensor node has to adapt to the stochastic nature of the energy source and has to decide *when to use* this energy. Goal of this adaptation is to maximize the utility of the application in a long-term perspective. The resulting mode of operation is sometimes called *energy neutral* operation [KPS04]: The performance of the application is not predetermined a priori, but adjusted in a best effort manner during runtime and ultimately dictated by the power source. Therefore, storage devices like batteries are solely used as energy buffers to compensate the variations of the underlying energy source.

Simply stated, this thesis addresses questions like the following:

- When should a sensor node use energy to sample and transmit data, and when should the sensor node idle and recharge the energy storage?
- If the energy storage is empty, the application has to be suspended and the sensor node has to be shut down. How can such energy underflows be avoided?
- If the energy storage is full, harvested energy cannot be stored and is wasted. How can such energy overflows be avoided?
- In real-time systems where a deadline violation may lead to severe performance degradation of the system, in which order shall we process tasks in order to respect all deadlines?
- How should we dimension the energy storage to optimally playout the variations of the underlying energy source?
- The hardware of a sensor node is stringently limited in terms of computation, memory and communication. How can we design low complexity power management solutions which respect those hardware constraints?

1.3 Environmental Energy Sources for Sensor Networks

In this section, we make an effort to overview potential energy sources for wireless sensor networks. In [RSF+04, Yea04, RWR03], current state of

the art, ongoing research as well as theoretical limits for many potential energy sources have been discussed. In comparison to the latter works, the overview in this thesis is not meant to be exhaustive. Rather, we want to provide the reader with some basic understanding of conversion principles and want to document the substantial interest that this topic has attracted recently, both in academia as well as in industry. For this purpose, we will also provide examples of commercially available products which have appeared on the market.

We will start with an brief overview in Section 1.3.1 and take a closer look at photovoltaic energy harvesting techniques in Section 1.3.2.

1.3.1 Overview

In general, the source and amount of energy that can be leveraged depends on the environment and the application. As the power consumption of electronic devices decreases permanently to the range of only μW, even formerly disregarded energy sources have become attractive.

Vibrational Energy

Devices which convert mechanical motion into electricity can be categorized in electromagnetic, electrostatic and piezoelectric converters. In the case of electromagnetic converters, a coil is oscillating in a static magnetic field and induces a voltage. In electrostatic converters, electric charge on variable capacitor plates creates a voltage if the plates are moved. Piezoelectric converters, finally, exploit the ability of some materials like crystals or ceramics to generate an electric potential in response to mechanical stress. A prominent example for the employment of vibrational harvesters is the watch industry, where vibrational energy converters have been used with success to power wristwatches.

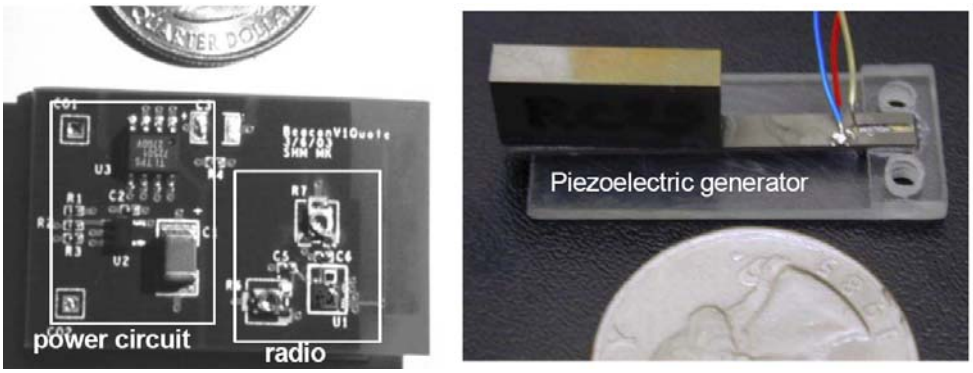
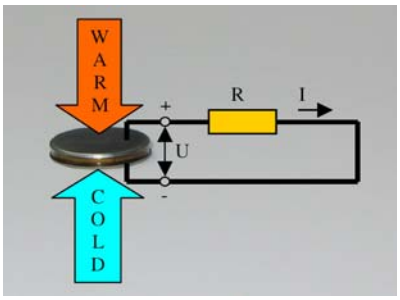


Fig. 3: Piezoelectric generator, power circuit and radio powered from vibrations [RSF+04].

In the context of wireless sensor networks, the authors of [RWR03] have analyzed common vibration sources such as cars, buildings or trains. The power harvested from the occurring low level vibrations can amount up to 300 $\frac{\mu W}{cm^3}$. On the other hand, also human activity can be the source of vibrational energy. In [SP01], a piezoelectric-powered RFID system for shoes has been presented which harvests energy from human walking activity. A similar approach has been taken in [RSF+04], where piezoelectric generators have been developed as an attractive method to power wireless transceivers (see Figure 3). In [ABM+05], one of the first prototype sensor nodes with a vibration harvesting micro power generator has been presented.

Thermal Energy

In principle, the presence of a temperature difference between two different metals or semiconductors causes a voltage. This effect is called Seebeck effect. Using thermoelectric conversion, thermal gradients in the environment can be directly converted into electric energy. An example for a commercial product which exploits thermoelectric conversion is Thermo Life ® [Sta06]. Thermo Life ® is a small thermoelectric generator manufactured by thin film technology which offers a power output of a few 10 to 100 μW and voltages in the Volt-range. For this output power, temperature gradients of only a few Kelvin are necessary. As displayed in Figure 4, the Thermo Life ® module has a diameter of $\approx$ 1 cm and a total height of 1.4 mm. Examples for possible application areas are biomedical implants, wearable electronic systems as well as structure embedded wireless micro sensors.



Electrical Parameter	@ ΔT=5 K	
Open circuit voltage	V	5.2
Short-circuit current	µA	22
Resistance	kΩ	236
Voltage @ matched load	V	2.6
Current @ matched load	µA	11
Power @ matched load	µW	28

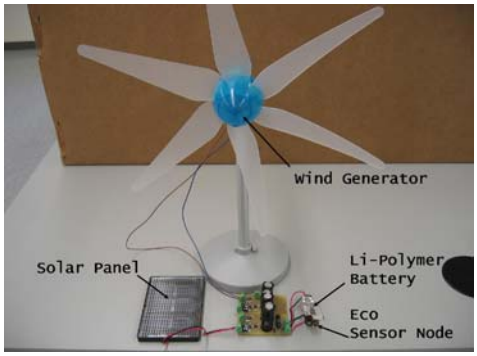
Fig. 4: Working principle of Thermo Life ® [Sta06].

Wind Energy

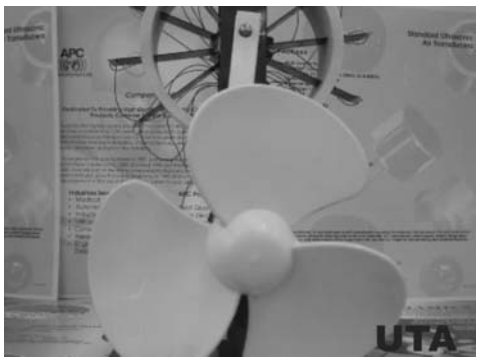
Compared to other methods, harvesting wind energy is limited to special application scenarios. Although it is less attractive due to the size of the mechanical components, some efforts to generate power at a very small

scale have been made recently.

As shown in [RSF+04], the power densities harvested from air velocity are quite promising. In addition, the authors of [PCFZ05] designed a windmill which uses piezoelectric elements to generate electricity from wind energy, see Figure 5(b). A power output of 10.3 mW was reported for a wind flow which leads to 6 rotations per minute. Furthermore, a maximum power point tracker for a small windmill was implemented in [PC06]. The experimental setup used in [PC06] is illustrated in Figure 5(a).



(a) AmbiMax hardware with a solar panel, wind generator, lithium polymer battery and Eco Node [PC06].



(b) Piezoelectric windmill prototype [PCFZ05].

Fig. 5: Wind harvesting circuits.

1.3.2 Photovoltaic Energy Harvesting

For outdoor deployments, solar energy harvested by photovoltaic cells is readily available in many sensor network scenarios. It becomes evident that solar energy is one of the most powerful energy sources [RSF+04]. Moreover, there is another reason why solar energy deserves a dedicated treatment in this thesis: Compared to other forms of environmental energy, solar energy is to some extent *predictable*. As already mentioned in the previous section, predictability is a prerequisite for meaningful planning of the future energy consumption.

Solar Cell Characteristics and Maximum Power Point Tracking (MPPT)

Photovoltaic systems have been used for decades to generate electricity, spanning from large-scale systems generating megawatts of power to small-scale systems operating in the range of milliwatts. For sensor network applications in an outdoor environment, several hundreds of $\frac{mW}{cm^2}$ are achievable.

Figure 6 depicts the voltage-current curve of a solar cell for a certain light condition. Obviously, the output power P_{solar} of the solar cell – which is the product of the voltage V_{solar} and the current I_{solar} – is very sensitive to the solar cell's voltage V_{solar}. The operation point in which the solar cell is providing the maximum power for a certain light condition is called the maximum power point (MPP). The current, voltage and power in the MPP are called maximum power point current I_{MPP}, voltage V_{MPP} and power P_{MPP}. If the light intensity is increasing, also the voltage V_{MPP} and the maximum power P_{MPP} are increasing.

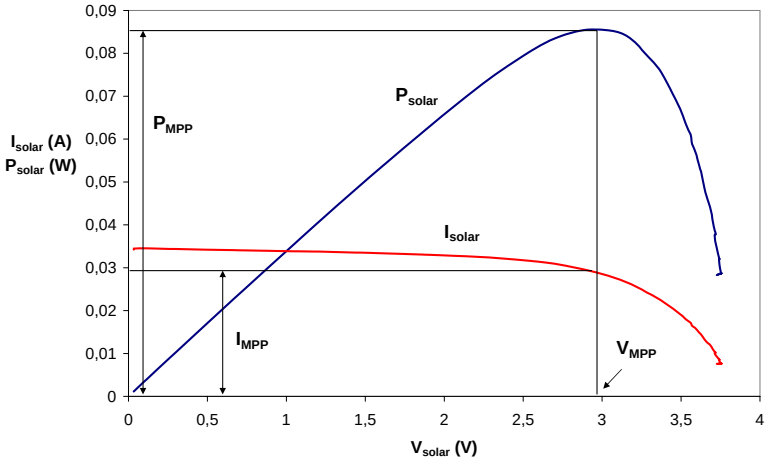
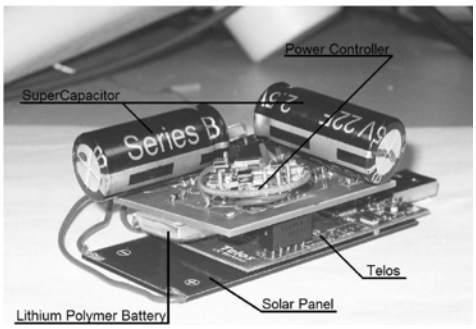


Fig. 6: Measured voltage-current and voltage-power curves of a solar cell for a certain light condition.

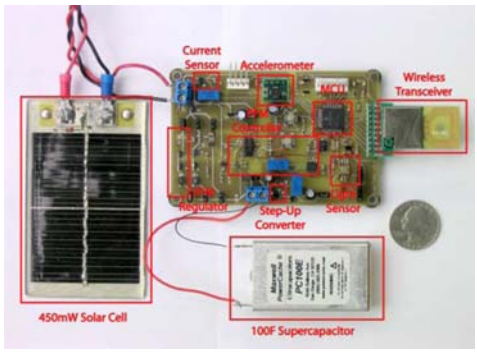
A lot of research has been done in the field of maximum power point tracking techniques [EC07]. These techniques try to aim for a continuous operation of the solar cell in its maximum power point (MPP) while adapting to varying light conditions. The goal is to harvest as much energy as possible from the available solar energy. For large-scale systems, where plenty of energy is available, dedicated microcontrollers are used for maximum power point tracking.

Photovoltaic Energy Harvesters for Sensor Nodes

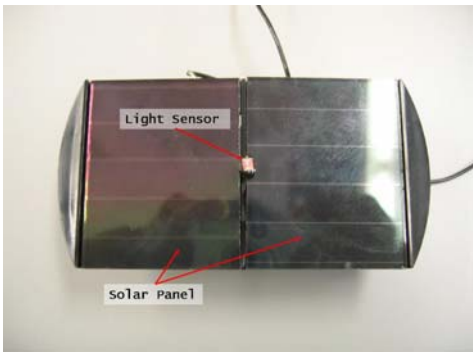
Recently, a number of solar powered prototype sensor nodes have been presented which perform more and more efficient energy conversion. Two of the first prototypes were Heliomote [HKF+05] and Prometheus [JPC05a]. In both systems, the solar panels are directly connected with the storage device. A picture of the energy scavenger Prometheus is depicted in Figure 7(a). Here, the solar cell is directly connected to a supercapacitor. This means that especially for low supercapacitor voltages, the solar cell generates much less power than its maximum power P_{MPP}.



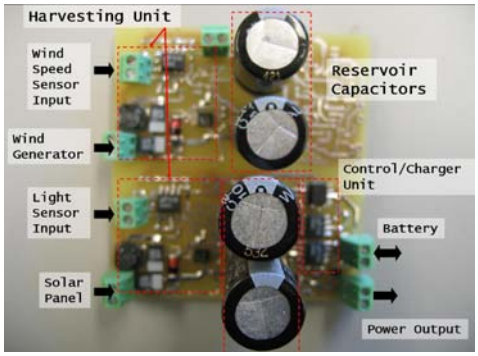
(a) Prometheus sensor node [JPC05a].



(b) Everlast prototype system [SC06].



(c) AmbiMax solar panel with light sensor [PC06].



(d) AmbiMax board with supercapacitors [PC06].

Fig. 7: Solar powered wireless sensor nodes.

An efficient solar harvesting system should adapt the electrical operating point of the solar cell to the given light condition. For solar cells the size of a few cm^2 particular care has to be taken in order not to waste the few mW generated by the solar cell. To this end, Everlast [SC06] uses the fractional short-circuit current technique (see Figure 7(b)). This technique is easy and cheap to implement and does not necessarily require DSP or microcontroller control. The voltage V_{MPP} is estimated based on the open-circuit voltage of the solar cell, which is measured periodically by momentarily shutting down the power converter that is connected to the solar cell. During this time, no energy is harvested. The AmbiMax [PC06] system exploits a small photosensor to detect the ambient light conditions and to force the solar cell to work in its MPP (see Figure 7(c)). The circuit presented in [BBMT08] is similar in nature to AmbiMax; however, instead of a photosensor miniaturized photovoltaic modules are used as pilot cells.

Latest versions of MPPT circuits accurately track the MPP of a solar cell and can adapt to changing light conditions quickly. These prototypes have successfully demonstrated that solar energy is a realistic energy source for sensor nodes and perpetual operation is indeed possible.

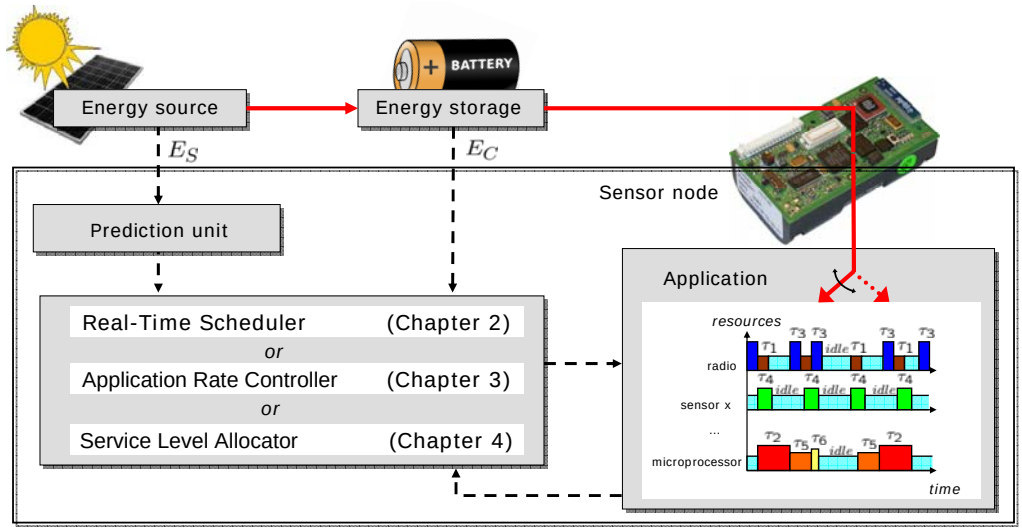


Fig. 8: Illustration of the system model and chapter overview.

1.4 Thesis Outline and Contributions

In this thesis, we present a set of novel power management solutions for energy harvesting embedded systems. As example scenario, we will stick to wireless sensor nodes which receive their energy from solar cells. Figure 8 sketches the principal setup which is investigated in this thesis: An energy source (e.g. a solar cell) generates electrical energy E_S which can be stored in an energy storage device (e.g. a battery). The stored energy E_C can be used by the sensor node to drive various applications. In Figure 8, the solid arrows indicate physical power flow whereas the dashed arrows represent information or control flow, respectively.

As illustrated in Figure 8, this thesis investigates three different application scenarios, namely **Real-Time Scheduling**, **Application Rate Control** as well as **Reward Maximization**. We provide dedicated algorithms for these application scenarios which will be presented in Chapters 2, 3 and 4, respectively. For all application scenarios, an estimation of the future energy harvesting is required to optimize the system performance. As illustrated in Figure 8, the estimation of the prediction unit is used as an input to the online scheduler which controls the application. In addition, the currently stored energy in the energy storage is measured.

The Real-Time Task Schedulers presented in Chapter 2 guarantee optimal task ordering in a *short-term perspective*. For instance, for time scales of milliseconds or seconds, the scheduler decides how to assign energy to time critical tasks. Application parameters as well as average power consumption are *not* influenced by the Real Time Task Scheduler.

Both the Application Rate Controller and the Service Level Allocator decide on the usage of harvested energy in a *long-term perspective*. At

this, parameters of the application are adapted for the next days or even weeks. As a result, the average power consumption of the system is optimized. The main difference between the approaches in Chapter 3 and Chapter 4 is the objective function, i.e., the way how the overall system performance is evaluated: While Chapter 3 addresses the case of (piece-wise) *linear* objective functions, Chapter 4 deals with *concave* objective functions. The resulting controllers for the application scenario in Chapter 3 are look-up tables, which basically store optimal application parameters for every possible input. The Service Level Allocator presented in Chapter 4, however, consists of a polynomial time algorithm which only requires the storage of a few internal variables. In return, the Application Rate Controllers in Chapter 3 can be designed for a wider variety of system dynamics and constraints.

All three application models and corresponding solution methods turn out to be of practical concern. They can be applied independently as presented in the respective chapters. Furthermore, it is possible to combine the proposed application models. For a discussion on how to combine the different models and methods, the reader is referred to Section 5.2.

So far, not much attention has been paid to the peculiarities of power management in energy harvesting systems. Most of the related works deal with maximizing energy savings in systems with finite energy sources. In this thesis, we demonstrate that classical power management solutions have to be reconceived and/or new problems arise in the context of energy harvesting. For the application scenarios Real-Time Scheduling and Reward Maximization, to the best of our knowledge, we have been the first to identify, formulate and solve the respective optimization problems. In general, compared to systems running state of the art power management solutions (which may not be tailored to energy harvesting systems), our methods offer the following advantages:

- Our methods optimize the system performance and significantly outperform state of the art approaches. The superior performance manifests, e.g., in a greater number of tasks which is schedulable (see Chapter 2), a better average performance (see Chapter 3) or a higher achievable system reward (see Chapter 4).

- The other way round, to achieve the same system performance, our methods allow the usage of less powerful energy sources (e.g. smaller solar panels) compared to state of the art approaches.

- Using our techniques, substantially smaller energy storages suffice to achieve the same performance as state of the art approaches, resulting in a reduced size, weight and cost of the embedded system.

- Our methods are carefully designed and exhibit low computational complexity. In other words, our software solutions are well-suited for resource constrained embedded systems.

Our results can be applied to, e.g., sensor nodes which are situated in an outdoor environment and are directly exposed to sunlight. Moreover, our results are applicable to other systems which are powered by an environmental energy source which has some kind of predictable behaviour. For instance, for sensor networks deployed in office buildings or hospitals, the harvested photovoltaic energy of the indoor illumination can also be predicted to some extent. On the one hand, lights in corridors and offices are switched on/off according to a fixed schedule. On the other hand, the different attendance times of employees can be learned readily by a prediction algorithm (day, night, weekday, weekend, holidays, etc.). Another example are sensor network deployments for monitoring vibrations of industrial machines, where the harvested vibrational energy can be predicted using the schedule of the machine activities.

In the following, the individual contributions of the three main chapters are summarized.

Chapter 2: Real-Time Scheduling

In Chapter 2, we consider applications with real-time requirements where *tasks* are given by arrival times, deadlines, computation times as well as a certain amount of energy which is required to complete each task. We point out that greedy scheduling is not suitable if tasks are processed using regenerative energy. We present Lazy Scheduling LSA, an optimal real-time scheduling algorithm for energy harvesting systems. Further we present an admittance test that decides whether a task set can be scheduled without deadline violations. Simulation results how that our algorithm allows substantial reductions of the battery size compared to EDF scheduling. The contributions described in this chapter are as follows:

- We present an energy-driven scheduling scenario for a system whose energy storage is recharged by an environmental source.

- For this scenario, we state and prove optimal online algorithms that dynamically assign power to arriving tasks. These algorithms are energy-clairvoyant, i.e., scheduling decisions are driven by the knowledge of the future incoming energy.

- We present an admittance test that decides, whether a set of tasks can be scheduled with the energy produced by the harvesting unit,

taking into account both energy and time constraints. For this purpose, we introduce the concept of energy variability characterization curves (EVCC). In addition, a formal comparison to EDF scheduling is provided.

- Using our admittance test, we show how to dimension important system parameters like, e.g., the size of the battery or the solar cells.

- By means of simulation, we demonstrate significant capacity savings of our algorithms compared to the classical EDF algorithm. Finally, we provide approximations which make our theoretical results applicable to practical energy harvesting systems.

Chapter 3: Application Rate Control

In contrast to the application model in Chapter 2 where the occurance of *tasks* is given and not a controllable parameter, in Chapter 3, tasks are actively invoked with certain *rates*. These rates are adapted using feedback controllers such that a maximal utility of the application is obtained. We present a framework for performance optimization in energy harvesting systems which allows the specification of various objectives, constraints as well as dependencies between application rates. The problems are formulated as linear programs. In order to avoid online optimization, we apply multiparametric linear programming to precompute state feedback controllers which are well-suited for resource constrained systems like, e.g., sensor nodes. Specifically, the chapter contains the following contributions:

- We present a specification model that is able to capture the performance and parameters of a large variety of rate-based applications in environmentally powered systems.

- We suggest the use of simple model predictive controllers for performance optimization. Concretely, we are applying results of the well-established field of multiparametric programming to the emerging area of energy harvesting systems.

- A hierarchical software design is presented which increases the robustness towards energy prediction mistakes. By designing the upper control layer for worst-case situations, depletion of the energy storage is avoided and robustness of the overall system is increased. In addition, we show that the hierarchical design reduces the computation overhead and storage demand significantly.

- We present a new algorithm for approximative multiparametric linear programming. The resulting control laws are rough approximations of the optimal solution and reduce the involved online overhead substantially. An experimental setup reveals that the achieved performance is not necessarily decreased compared to the optimal solution.
- We evaluate our methods by means of simulation using longterm measurements of solar energy as input data. In this way, we could extensively test the performance of our algorithms for time scales one usually wants to achieve with, e.g., solar powered sensor networks.
- We propose practical techniques for the efficient implementation of the controllers, give a thorough analysis of the involved implementation overhead and demonstrate the practical relevance of our approach by measurements of the controller running on a real sensor node.

Chapter 4: Reward Maximization

In Chapter 4, we explore how to maximize the system reward for embedded systems which provide *services* periodically with adjustable quality. At this, different qualities of services are expressed utilizing the notion of rewards. We assume the reward of a service to be monotonically increasing and concave with respect to its energy consumption. For this particular optimization problem, we propose polynomial-time algorithms that derive optimal assignments in energy consumption to maximize the overall reward. The detailed contributions are listed below.

- We formulate the *general reward maximization on energy harvesting* problem, which is to maximize the sum of rewards for concave reward functions due to constraints of a regenerative energy source and the energy buffer.
- We propose a two-stage mechanism for service level allocation including polynomial time algorithms that derive optimal service levels to maximize the overall reward.
- To provide insights for system designers, we show how to determine the minimum battery capacity and a sufficient prediction horizon for a given power source.
- Our results are supported by simulations based on long-term measurements of photovoltaic energy.

- Finally, we outline how our algorithms can be implemented efficiently on embedded systems and elaborate on implementation details.

2

Real-Time Scheduling

Energy harvesting has recently emerged as a feasible option to increase the operating time of sensor networks. If each node of the network, however, is powered by a fluctuating energy source, common power management solutions have to be reconceived. This holds in particular if real-time responsiveness of a given application has to be guaranteed. Task scheduling at the single nodes should account for the properties of the energy source, capacity of the energy storage as well as deadlines of the single tasks. We show that conventional scheduling algorithms (like e.g. EDF) are not suitable for this scenario. Based on this motivation, we have constructed optimal scheduling algorithms that jointly handle constraints from both energy and time domain. Further we present an admittance test that decides for arbitrary task sets, whether they can be scheduled without deadline violations. To this end, we introduce the concept of energy variability characterization curves (EVCC) which nicely captures the dynamics of various energy sources. Simulation results show that our algorithms allow significant reductions of the battery size compared to Earliest Deadline First scheduling.

2.1 Introduction

In this chapter, we investigate scheduling policies for *application scenarios with real-time requirements*, like e.g. fire or intruder detection systems. In general, one can classify real-time application scenarios for wireless sensor networks into safety critical systems, smart spaces as well as entertain-

ment [SAL+03]. For all these scenarios, our research reveals fundamental problems and tradeoffs when real-time behaviour has to be guaranteed although a sensor's driving energy source is highly unstable.

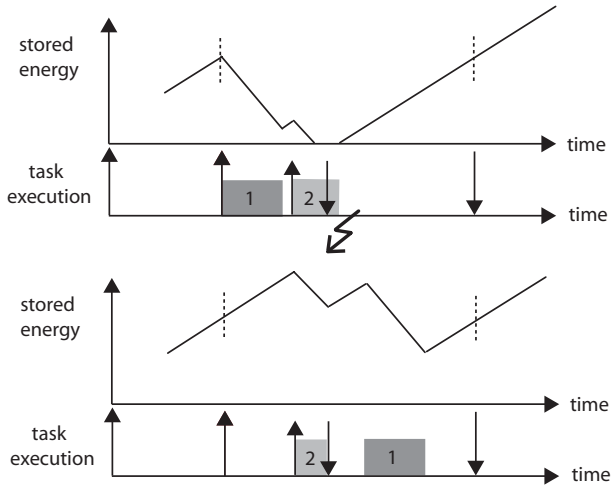


Fig. 9: Greedy vs. lazy scheduling.

The example in Figure 9 illustrates why greedy scheduling algorithms (like Earliest Deadline First EDF) are not suitable in the context of regenerative energy. Let us consider a node with an energy harvesting unit that replenishes a battery. For the sake of simplicity, assume that the harvesting unit provides a constant power output. Now, this node has to perform an arriving task "1" that has to be finished until a certain deadline. Meanwhile, a second task "2" arrives that has to respect a deadline which is earlier than the one of task "1". In Figure 9, the arrival times and deadlines of both tasks are indicated by up and down arrows respectively. As depicted in the top diagrams, a greedy scheduling strategy violates the deadline of task "2" since it dispenses overhasty the stored energy by driving task "1". When the energy is required to execute the second task, the battery level is not sufficient to meet the deadline. In this example, however, a scheduling strategy that hesitates to spend energy on task "1" meets both deadlines. The bottom plots illustrate how a *Lazy Scheduling Algorithm* described in this chapter outperforms a naive, greedy approach like EDF in this situation. Lazy scheduling algorithms can be categorized as non-work conserving scheduling disciplines. Unlike greedy algorithms, a lazy scheduler may be idle although waiting tasks are ready to be processed.

The research presented in this chapter is directed towards sensor nodes. But in general, our results apply for all kind of energy harvesting systems which must schedule processes under deadline constraints. For these systems, new scheduling disciplines must be tailored to the energy-

driven nature of the problem. This insight originates from the fact, that energy – contrary to the computation resource "time" – is storable. As a consequence, every time we withdraw energy from the battery to execute a task, we change the state of our scheduling system. That is, after having scheduled a first task the next task will encounter a lower energy level in the system which in turn will affect its own execution. This is not the case in conventional real-time scheduling where time just elapses either used or unused.

The rest of this chapter is organized as follows: In the next section, we review related work. Subsequently, Section 2.3 gives definitions that are essential for the understanding of the results in this chapter. In Section 2.4, we present *Lazy Scheduling Algorithms* for optimal online scheduling and proof their optimality. Admittance tests for arbitrary task sets are the topic of Section 2.5. Simulation results are presented in Section 2.6 and Section 2.7 deals with practical issues. At the end, Section 2.8 concludes the chapter.

2.2 Related Work

In [KPS04], the authors use a similar model of the power source as we do. But instead of executing concrete tasks in a real-time fashion, they propose tuning a node's duty cycle dependent on the parameters of the power source. Nodes switch between active and sleep mode and try to achieve sustainable operation. This approach only indirectly addresses real-time responsiveness: It determines the latency resulting from the sleep duration.

The approach in [RMM03] is restricted to a very special offline scheduling problem: Periodic tasks with certain rewards are scheduled within their deadlines according to a given energy budget. The overall goal is to maximize the sum of rewards. Therefore, energy savings are achieved using Dynamic Voltage Scaling (DVS). The energy source is assumed to be solar and comprises two simple states: day and night. Hence the authors conclude that the capacity of the battery must be at least equal to the cumulated energy of those tasks, that have to be executed at night. In contrast, our work deals with a much more detailed model of the energy source. We focus on scheduling decisions for the online case when the scheduler is *indeed* energy-constraint. In doing so, we derive valuable bounds on the necessary battery size for arbitrary energy sources and task sets.

The research presented in [AM01] is dedicated to offline algorithms for scheduling a set of periodic tasks with a common deadline. Within this so-called "frames", the order of task execution is *not* crucial for whether

the task set is schedulable or not. The power scavenged by the energy source is assumed to be constant. Again – by using DVS – the energy consumption is minimized while still respecting deadlines. Contrary to this work, our systems (e.g. sensor nodes) are predominantly energy constrained and the energy demand of the tasks is fixed (no DVS). We propose algorithms that make best use of the available energy. Provided that the average harvested power is sufficient for continuous operation, our algorithms minimize the necessary battery capacity.

The primary commonality of [PUBG01] and our work is the term "lazy scheduling". In [PUBG01], lazy packet scheduling algorithms for transmitting packetized information in a wireless network are discussed. The approach is based on the observation that many channel coding schemes allow to reduce the energy per packet if it is transmitted slower, i.e. over a longer duration. Goal of this approach is to minimize the energy, whereas in our work the energy consumption is fixed (given by the task set). Furthermore, their scheduling algorithms are fully work conserving, which is not true for our algorithms.

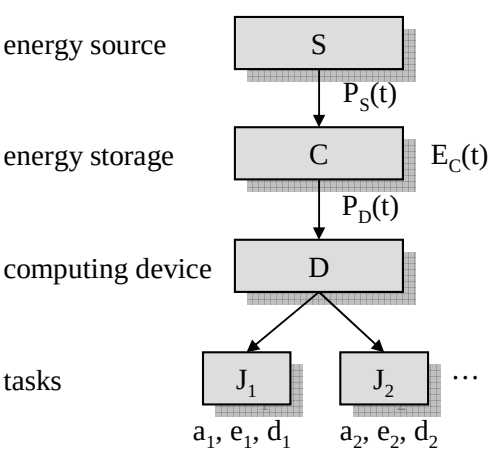
2.3 System Model

The chapter deals with a scheduling scenario depicted in Fig. 10(a). At some time t, an energy source harvests ambient energy and converts it into electrical power $P_S(t)$. This power can be stored in a device with capacity C. The stored energy is denoted as $E_C < C$. On the other hand, a computing device drains power $P_D(t)$ from the storage and uses it to process tasks with arrival time a_i, energy demand e_i and deadline d_i. We assume that only one task is executed at time t and preemptions are allowed.

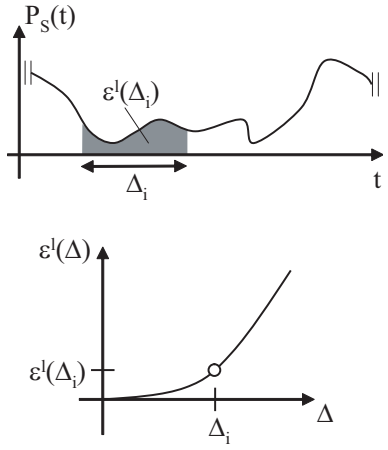
The problem statement presented in this section comprises two major constraints which have to be satisfied: First, tasks can be processed exclusively with energy E_S generated by the energy source. And second, timing constraints in terms of tasks' deadlines d_i must be respected. For this purpose, two degrees of freedom can be exploited. The scheduler may decide which task J_i of all ready tasks to execute and what amount of power P_D to assign to this task. The following subsections define the relations between these quantities in more detail.

2.3.1 Energy Source

Many environmental power sources are highly variable with time. Hence, in many cases some charging circuitry is necessary to optimize the charging process and increase the lifetime of the storage device. In our model,



(a) Energy-driven scheduling scenario.



(b) Power trace and energy variability characterization curve EVCC.

Fig. 10: Representation of the system model.

the power P_S incorporates all losses caused by power conversion as well as charging process. In other words, we denote $P_S(t)$ the charging power that is actually fed into the energy storage. The respective energy E_S in the time interval $[t_1, t_2]$ is given as

$$E_S(t_1, t_2) = \int_{t_1}^{t_2} P_S(t)\mathrm{d}t \quad .$$

In order to characterize the properties of an energy source, we define now energy variability characterization curves (EVCC) that bound the energy harvested in a certain interval Δ: The EVCCs $\epsilon^l(\Delta)$ and $\epsilon^u(\Delta)$with $\Delta \geq 0$ bound the range of possible energy values E_S as follows:

$$\epsilon^l(t_2 - t_1) \leq E_S(t_1, t_2) \leq \epsilon^u(t_2 - t_1) \quad \forall t_2 > t_1$$

Given an energy source, e.g., a solar cell mounted in a building or outside, the EVCCs provide guarantees on the produced energy. For example, the lower curve denotes that for any time interval of length Δ, the produced energy is at least $\epsilon^l(\Delta)$ (see Fig. 10(b)). Three possible ways of deriving an EVCC for a given scenario are given below:

- A sliding window of length Δ is used to find the minimum/maximum energy produced by the energy source in any time interval $[t_1, t_2)$ with $t_2 - t_1 = \Delta$. To this end, one may use a long power trace or a set of traces that have been measured. Since the resulting EVCC bounds only the underlying traces, these traces must

be selected carefully and have to be representative for the assumed scenario.

- The energy source and its environment is formally modeled and the resulting EVCC is computed.
- Approximations to EVCCs can be determined on-line by using appropriate measurement and estimation methods, see Section 2.7.1.

In Section 2.5, the lower EVCC ϵ^l will be used in an admittance test which decides, whether a task set is schedulable given a certain energy source. Furthermore, both EVCCs will serve as energy predictors for the algorithms simulated in Section 2.6.

2.3.2 Energy Storage

We assume an ideal energy storage that may be charged up to its capacity C. According to the scheduling policy used, power $P_D(t)$ and the respective energy $E_D(t_1, t_2)$ is drained from the storage to execute tasks. If no tasks are executed and the storage is consecutively replenished by the energy source, an energy overflow occurs. Consequently, we can derive the following constraints

$$\begin{aligned} 0 \leq E_C(t) &\leq C && \forall t \\ E_C(t_2) &\leq E_C(t_1) + E_S(t_1, t_2) - E_D(t_1, t_2) && \forall t_2 > t_1 \end{aligned}$$

and therefore

$$E_D(t_1, t_2) \leq E_C(t_1) + E_S(t_1, t_2) \qquad \forall t_2 > t_1 \,.$$

2.3.3 Task Scheduling

As illustrated in Fig. 10(a), we utilize the notion of a computing device that assigns energy E_C from the storage to dynamically arriving tasks. We assume that the power consumption $P_D(t)$ is limited by some maximum value P_{max}. In other words, the processing device determines at any point in time how much power it uses, that is

$$0 < P_D(t) < P_{max} \,.$$

We assume tasks to be independent from each other and preemptive. More precisely, the currently active task may be preempted at any time and have its execution resumed later, at no additional cost. If the node

decides to assign power $P_i(t)$ to the execution of task J_i during the interval $[t_1, t_2]$, we denote the corresponding energy

$$E_i(t_1, t_2) = \int_{t_1}^{t_2} P_i(t)\mathrm{d}t \quad .$$

The effective starting time s_i and finishing time f_i of task i are dependent on the scheduling strategy used: A task starting at time s_i will finish as soon as the required amount of energy e_i has been consumed by it. We can write

$$f_i = \min\{t : E_i(s_i, t) = e_i\} \ .$$

The actual running time $(f_i - s_i)$ of a task i directly depends on the amount of power $P_i(t)$ which is driving the task during $s_i \leq t \leq f_i$. At this, the energy demand e_i of a task is independent from the power P_i used for its execution. Note that we are *not* using energy-saving techniques like Dynamic Voltage Scaling (DVS), where $e_i = \mathrm{f}(P_i)$. In our model, power P_i and execution time w_i behave inversely proportional: The higher the power P_i, the shorter the execution time w_i. In the best case, a task may finish after the execution time $w_i = \frac{e_i}{P_{max}}$ if it is processed without interrupts and with the maximum power P_{max}.

Current hardware technology does not support variable power consumption as described above. So clearly, the continuous task processing model presented in this section is idealized. However, a microprocessor for example may stepwise advance a task by switching power on ($P_i = P_{max}$) and off ($P_i = 0$). By tuning the so-called duty cycle accordingly, devices can approximate the average power $0 \leq \overline{P}_i \leq P_{max}$. For a more detailed discussion about practical task processing and the system model in general, see Section 2.7.

2.4 Lazy Scheduling Algorithms LSA

After having described our modeling assumptions, we will now state and prove optimal scheduling algorithms. In subsection 2.4.1, we will start with the analysis of a simplified scheduling scenario where tasks need only energy as computation resource but may execute in zero time. By disregarding the computation resource time, we focus on the energy-driven nature of the scheduling scenario. In Section 2.4.2, we will consider finite execution times as well and construct a more general algorithm which manages to optimally trade off energy and time constraints. Theorems which prove optimality of both algorithms will follow in subsection 2.4.3.

2.4.1 Simplified Lazy Scheduling

We start with a node with infinite power $P_{max} = +\infty$. As a result, a task's execution time w_i collapses to 0 if the available energy E_C in the storage is equal to or greater than the task's energy demand e_i. This fact clearly simplifies the search for an adequate scheduling algorithm but at the same time contributes to the understanding of the problem.

As already indicated in the introduction, the naive approach of simply scheduling tasks with the EDF algorithm may result in unnecessary deadline violations, see Fig. 9. It may happen, that after the execution of task "1" another task "2" with an earlier deadline arrives. If now the required energy is not available before the deadline of task "2", EDF scheduling produces a deadline violation. If task "1" would hesitate instead of executing directly, this deadline violation might be avoidable. These considerations directly lead us to the principle of *Lazy Scheduling*: Gather environmental energy and process tasks only if it is necessary.

The *Lazy Scheduling Algorithm* LSA-I for $P_{max} = \infty$ shown below attempts to schedule a set of tasks J_i, $i \in Q$ such that deadlines are respected. Therefore, the processing device has to decide between three power modes. The node may process tasks with the maximal power $P_D(t) = P_{max}$ or not at all ($P_D(t) = 0$). In between, the node may choose to spend only the currently incoming power $P_S(t)$ from the harvesting unit on tasks. The algorithm is based on the three following rules:

- **Rule 1**: If the current time t equals the deadline d_j of some arrived but not yet finished task J_j, then finish its execution by draining energy $(e_j - E_j(a_j, t))$ from the energy storage instantaneously.
- **Rule 2**: We must not waste energy if we could spend it on task execution. Therefore, if we hit the capacity limit ($E_C(t) = C$) at some times t, we execute the task with the earliest deadline using $P_D(t) = P_S(t)$.
- **Rule 3**: Rule 1 overrules Rule 2.

Note that LSA-I degenerates to an *earliest deadline first* (EDF) policy, if $C = 0$. On the other hand, we find an *as late as possible* (ALAP) policy for the case of $C = +\infty$.

2.4.2 General Lazy Scheduling

The LSA-I algorithm instantaneously executes a task at its deadline. However, with limited power consumption P_D and finite, minimal computation times $w_i = \frac{e_i}{P_{max}}$ a general algorithm has to determine earlier starting times $s_i \leq d_i$ in order to respect deadlines. In the following, we sketch

Algorithm 1 Lazy Scheduling Algorithm LSA-I ($P_{max} = \infty$)

Require: maintain a set of indices $i \in Q$ of all ready but not finished tasks J_i
  $P_D(t) \Leftarrow 0$;
  while (true) **do**
    $d_j \Leftarrow \min\{d_i : i \in Q\}$;
    process task J_j with power $P_D(t)$;
    $t \Leftarrow$ current time;
    if $t = a_k$ **then** add index k to Q;
    if $t = f_j$ **then** remove index j from Q;
    if $t = d_j$ **then** $E_C(t) \Leftarrow E_C(t) - e_j + E_j(a_j, t)$;
      remove index j from Q;
      $P_D(t) \Leftarrow 0$;
    if $E_C(t) = C$ **then** $P_D(t) \Leftarrow P_S(t)$;

how to determine optimal starting times s_i that balance the *time* and *energy* constraints for each single task J_i. For a more detailed derivation of the starting times s_i the reader is referred to [MBTB06].

At first sight, starting a task as late as possible (ALAP) seems to be a promising approach. The upper plots in Fig. 11 display a straightforward ALAP-translation of the starting time for task "1": To fulfill its time condition, task "1" begins to execute at starting time $s_1 = d_1 - \frac{e_1}{P_{max}}$. As illustrated, it may happen that shortly after s_1 an unexpected task "2" arrives. Assume that this unexpected task "2" is nested in task "1", i.e., it also has an earlier deadline than "1". This scenario inevitably leads to a deadline violation, although plenty of energy is available. This kind of timing conflict can be solved by shifting s_1 to earlier times and thereby reserving time for the unpredictable task "2" (see lower plots Fig. 11). But starting earlier, we risk to "steal" energy that might be needed at later times (compare Fig. 9). According to the "lazy" principle we have to take care that we don't start *too* early.

From the above example, we learned that it may be disadvantageous to prearrange a starting time in such a way, that the stored energy E_C cannot be used before the deadline of a task. If the processing device starts running at time s_i with P_{max} and cannot consume all the available energy before the deadline d_i, time conflicts may occur. On the other hand, if we remember the introductory example in Fig. 9, energy conflicts are possible if the stored energy $E_C(t)$ is 0 at some time $t < d_i$. Hence we can conclude the following: The optimal starting time s_i must guarantee, that the processor *could* continuously use P_{max} in the interval $[s_i, d_i]$ and empty the energy storage $E_C(d_i) = 0$ exactly at time d_i. Before the optimal starting time s_i, the scheduler has to conserve energy and keep the storage level E_C as high as possible.

A necessary prerequisite for the calculation of the optimal starting time s_i is the knowledge of the incoming power flow $P_S(t)$ for all future times

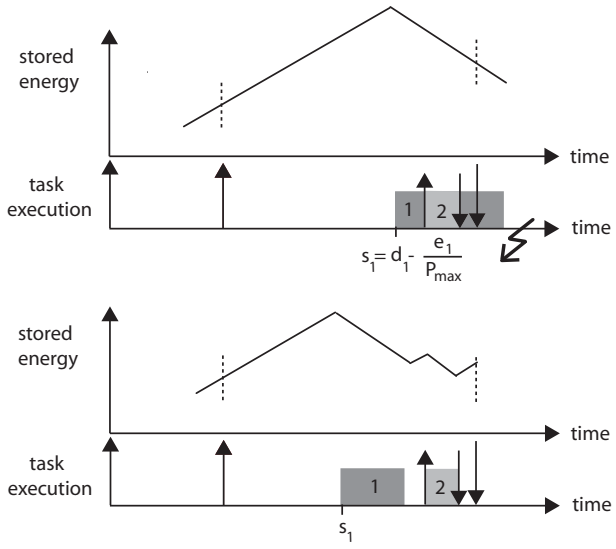


Fig. 11: ALAP vs. lazy scheduling.

$t \leq d_i$. Finding useful predictions for the power $P_S(t)$ can be done, e.g., by analyzing traces of the harvested power, as we will see in Section 2.6. In addition, we assume that $P_S(t) < P_{max}$, that is, the incoming power $P_S(t)$ from the harvesting unit never exceeds the power consumption P_{max} of a busy node. Besides from being a realistic model in many cases, this assumption ensures that no energy is wasted if the energy storage is full and the system is running with P_{max}.

To calculate the optimal starting time s_i, we determine the maximum amount of energy $E_C(a_i) + E_S(a_i, d_i)$ that may be processed by the node before d_i. Next, we compute the minimum time required to process this energy without interruption and shift this time interval of continuous processing just before the deadline d_i. In other words, we calculate the starting time s_i^* as

$$s_i^* = d_i - \frac{E_C(a_i) + E_S(a_i, d_i)}{P_{max}}.$$

If now the energy storage is filled before s_i^*, it is reasonable to advance task J_i with power P_S in order to avoid an energy overflow (compare Rule 2 of LSA-I). However, this also means that not all energy $E_C(a_i) + E_S(a_i, d_i)$ can be processed continuously in $[s_i^*, d_i]$ and we find $E_C(t) = 0$ at some time $t < d_i$. Thus a better starting time s_i' allows for the reduced amount of energy $C + E_S(s_i', d_i)$ which is processible in this situation:

$$s_i' = d_i - \frac{C + E_S(s_i', d_i)}{P_{max}}$$

By choosing the maximum of s_i^* and s_i' we find the optimal starting time

$$s_i = \max\left(s_i', s_i^*\right),$$

which precisely balances energy and time constraints for task J_i.

The pseudo-code of a general Lazy Scheduling Algorithm LSA-II is shown below. It is based on the following rules:

- **Rule 1**: EDF scheduling is used at time t for assigning the processor to all waiting tasks with $s_i \leq t$. The currently running task is powered with $P_D(t) = P_{max}$.
- **Rule 2**: If there is no waiting task i with $s_i \leq t$ and if $E_C(t) = C$, then all incoming power P_S is used to process the task with the smallest deadline ($P_D(t) = P_S(t)$).

Algorithm 2 Lazy Scheduling Algorithm LSA-II (P_{max} = const.)

Require: maintain a set of indices $i \in Q$ of all ready but not finished tasks J_i
  $P_D(t) \Leftarrow 0$;
  while (true) **do**
    $d_j \Leftarrow \min\{d_i : i \in Q\}$;
    calculate s_j;
    process task J_j with power $P_D(t)$;
    $t \Leftarrow$ current time;
    if $t = a_k$ **then** add index k to Q;
    if $t = f_j$ **then** remove index j from Q;
    if $E_C(t) = C$ **then** $P_D(t) \Leftarrow P_S(t)$;
    if $t \geq s_j$ **then** $P_D(t) \Leftarrow P_{max}$;

Although it is based on the knowledge of the future incoming energy E_S, LSA-II remains an online algorithm. The calculation of s_i must be performed once the scheduler selects the task with the earliest deadline. If the scheduler is not energy-constraint, i.e., if the available energy is more than the device can consume with power P_{max} within $[a_i, d_i]$, the starting time s_i will be before the current time t. Then, the resulting scheduling policy is EDF, which is reasonable, because only time constraints have to be satisfied. If, however, the sum of stored energy E_C plus generated energy E_S is small, the scheduling policy changes towards an ALAP policy. In doing so, LSA avoids spending scarce energy on the "wrong" tasks too early.

In summary, LSA-II can be classified as an energy-clairvoyant adaptation of the Earliest Deadline First Algorithm. It changes its behaviour according to the amount of available energy, the capacity C as well as the maximum power consumption P_{max} of the device. For example, the

lower the power P_{max} gets, the greedier LSA-II gets. On the other hand, high values of P_{max} force LSA-II to hesitate and postpone the starting time s_i. For $P_{max} = \infty$, all starting times collapse to the respective deadlines, and we identify LSA-I as a special case of LSA-II. In the remainder of the chapter, we will solely consider the general LSA-II algorithm derived in this section. From now on, we will denote this algorithm LSA.

2.4.3 Optimality of Lazy Scheduling

In this section, we will show that Lazy Scheduling algorithms optimally assign power P_D to a set of tasks. For this purpose, we formulate Theorem 1 and Theorem 2 which show that LSA makes the best use of the available time and energy, respectively. With the help of Theorem 1 and 2, we proof optimality of LSA in Theorem 3.

The scheduling scenario presented in this chapter is inherently energy-driven. Hence, a scheduling algorithm yields a deadline violation if it fails to assign energy e_i to a task before its deadline d_i. We distinguish between two types of deadline violations:

- A deadline cannot be respected since the time is not sufficient to execute available energy with power P_{max}. At the deadline, unprocessed energy remains in the storage and we have $E_C(d) > 0$. We call this the **time limited** case.
- A deadline violation occurs because the required energy is simply not available at the deadline. At the deadline, the battery is exhausted (i.e., $E_C(d) = 0$). We denote the latter case **energy limited**.

For the following theorems to hold we suppose that at initialization of the system, we have a full capacity, i.e., $E_C(t_i) = C$. Furthermore, we call the computing device *idle* if no task i is running with $s_i \leq t$.

Let us first look at the time limited case.

Thm. 1: *Let us suppose that the LSA algorithm schedules a set of tasks. At time d the deadline of a task J with arrival time a is missed and* $E_C(d) > 0$. *Then there exists a time* t_1 *such that the sum of execution times* $\sum_{(i)} w_i = \sum_{(i)} \frac{e_i}{P_{max}}$ *of tasks with arrival and deadline within time interval* $[t_1, d]$ *exceeds* $d - t_1$.

Proof. Let us suppose that t_0 is the maximal time $t_0 \leq d$ where the processor was idle. Clearly, such a time exists.

We now show, that at t_0 there is no task i with deadline $d_i \leq d$ waiting. At first, note that the processor is constantly operating on tasks in time interval $(t_0, d]$. Suppose now that there are such tasks waiting and task i is actually the one with the earliest deadline d_i among those. Then, as $E_C(d) > 0$ and because of the construction of s_i, we would have $s_i < t_0$.

Therefore, the processor would actually process task i at time t_0 which is a contradiction to the idleness.

Because of the same argument, all tasks i arriving after t_0 with $d_i \leq d$ will have $s_i \leq a_i$. Therefore, LSA will attempt to directly execute them using an EDF strategy.

Now let us determine time $t_1 \geq t_0$ which is the largest time $t_1 \leq d$ such that the processor continuously operates on tasks i with $d_i \leq d$. As we have $s_i \leq a_i$ for all of these tasks and as the processor operates on tasks with smaller deadlines first (EDF), it operates in $[t_1, d]$ only on tasks with $a_i \geq t_1$ and $d_i \leq d$. As there is a deadline violation at time d, we can conclude that $\sum_{(i)} w_i > d - t_1$ where the sum is taken over all tasks with arrival and deadline within time interval $[t_1, d]$.

□

It can be shown that a related result holds for the energy limited case, too.

Thm. 2: *Let us suppose that the LSA algorithm schedules a set of tasks. At time d the deadline of a task J with arrival time a is missed and* $E_C(d) = 0$. *Assume further, that deadline d is the first deadline of the task set that is violated by LSA. Then there exists a time* t_1 *such that the sum of task energies* $\sum_{(i)} e_i$ *of tasks with arrival and deadline within time interval* $[t_1, d]$ *exceeds* $C + E_S(t_1, d)$.

Proof. Let time $t_1 \leq d$ be the largest time such that (a) $E_C(t_1) = C$ and (b) there is no task i waiting with $d_i \leq d$. Such a time exists as one could at least use the initialization time t_i with $E_C(t_i) = C$. As t_1 is the last time instance with the above properties, we can conclude that everywhere in time interval $[t_1, d]$ we either have (a) $E_C(t) = C$ and there is some task i waiting with $d_i \leq d$ or we have (b) and $E_C(t) < C$.

It will now be shown that in both cases a) and b), energy is not used to advance any task j with $d_j > d$ in time interval $[t_1, d]$. Note also, that all arriving energy $E_S(t_1, d)$ is used to advance tasks.

In case a), all non-storable energy (i.e. all energy that arrives from the source) is used to advance a waiting task, i.e., the one with the earliest deadline $d_i \leq d$. In case b), the processor would operate on task J with $d_j > d$ if there is some time $t_2 \in [t_1, d]$ where there is no other task i with $d_i \leq d$ waiting and $s_j \leq t_2$. But s_j is calculated such that the processor could continuously work until d_j. As $d_j > d$ and $E_C(d) = 0$ this can not happen and $s_j > t_2$. Therefore, also in case b) energy is not used to advance any task j with $d_j > d$.

As there is a deadline violation at time d, we can conclude that $\sum_{(i)} e_i > C + E_C(t_1, d)$ where the sum is taken over all tasks with arrival and deadline within time interval $[t_1, d]$.

□

From the above two theorems we draw the following conclusions: First, in the **time limited** case, there exists a time interval before the violated deadline with a larger accumulated computing time request than available time. And second, in the **energy limited** case, there exists a time interval before the violated deadline with a larger accumulated energy request than what can be provided at best. These considerations lead us to one of the major results of the chapter:

Thm. 3: (Optimality of Lazy Scheduling) *Let us consider a system characterized by a capacity C and power P_{max} driven by the energy source E_S. If LSA cannot schedule a given task set, then no other scheduling algorithm is able to schedule it. This holds even if the other algorithm knows the complete task set in advance.*

Proof. The proof follows immediately from Theorems 1 and 2. Assume a set of tasks is scheduled with LSA and at time d the deadline of task J is missed. Assume further, that deadline d is the first deadline of the task set that is violated by LSA. In the following, we distinguish between the case where the energy at the deadline is $E_C(d) > 0$ and $E_C(d) = 0$, respectively.

In the first case, according to Theorem 1, there exists a time t_1 such that the sum of execution times $\sum_{(i)} w_i = \sum_{(i)} \frac{e_i}{P_{max}}$ of tasks with arrival and deadline within time interval $[t_1, d]$ exceeds $d - t_1$. Here, knowing arrival times, energy demands and deadlines in advance does not help, since every scheduling algorithm will at least violate one deadline in $[t_1, d]$.

In the energy limited case with $E_C(d) = 0$, according to Theorem 2, there exists a time t_1 such that the sum of task energies $\sum_{(i)} e_i$ of tasks with arrival and deadline within time interval $[t_1, d]$ exceeds $C + E_S(t_1, d)$. Hence, no algorithm can hold deadline d without violating an earlier deadline in $[t_1, d]$. This holds also for omniscient scheduling algorithms, since (a) at the beginning of the critical interval $[t_1, d]$, the energy level may be $E_C(t_1) = C$ at most and (b) the execution of the critical tasks can start at time t_1 at the earliest.

So every time LSA violates deadline d, we have either the time limited case ($E_C(d) > 0$) or the energy limited case ($E_C(d) = 0$). Since in both cases it is impossible for another algorithm to respect deadline d and all earlier deadlines simultaneously, we conclude that LSA is optimal.

□

If we can guarantee that there is no time interval with a larger accumulated computing time request than available time and no time interval with a larger accumulated energy request than what can be provided at best, then the task set is schedulable. This property will be used to determine the admittance test described in the next section.

On the other hand, given a task set and a certain energy source $E_S(t)$ we can also make a statement about the necessary hardware requirements of the sensor node: Due to its optimality, LSA requires the minimum processing power P_{max} and the minimum capacity C necessary to avoid deadline violations:

Thm. 4: (Optimal tuple $(C; P_{max})$) *Let us assume a given task set has to be scheduled using an energy source E_S. Among all algorithms, LSA requires the minimum capacity C and the minimum power P_{max} that are necessary to successfully schedule the task set.*

Proof. We proceed by contradiction. Let us denote C and P_{max} the minimum capacity and the minimum processing power which are needed to schedule a given task set with LSA. Assume that an adversary algorithm succeeds to schedule the task set with some $C' < C$ or $P'_{max} < P_{max}$ given the same energy source. This means the adversary algorithm can schedule the respective task set with (C', P'_{max}) and LSA cannot. This however contradicts the optimality of LSA according to Theorem 3.

□

The admittance test in the next section will allow us to explicitly determine the minimum values of C and P_{max} for LSA scheduling.

2.5 Admittance Test

2.5.1 Lazy Scheduling Algorithm

In this section, we will determine an offline schedulability test in case of periodic, sporadic or even bursty sets of tasks. In particular, given an energy source with lower EVCC $\epsilon^l(\Delta)$, the device parameters $(C; P_{max})$ and a set of periodic tasks J_i, $i \in I$ with period p_i, *relative* deadline d_i and energy demand e_i, we would like to determine whether all deadlines can be respected.

To this end, let us first define for each task its arrival curve $\alpha(\Delta)$ which denotes the maximal number of task arrivals in any time interval of length Δ. The concept of arrival curves to describe the arrival patterns of sets of tasks is well known (request bound functions) and has been used explicitly or implicitly in, e.g., [Bar03] or [WMT05]. To simplify the discussion, we limit ourselves to periodic tasks, but the whole formulation allows to deal with much more general classes (sporadic or bursty) as well.

In case of a periodic task set, we have for periodic task J_i, see also Fig. 12:

$$\alpha_i(\Delta) = \left\lceil \frac{\Delta}{p_i} \right\rceil \quad \forall \Delta \geq 0$$

In order to determine the maximal energy demand in any time interval of length Δ, we need to maximize the accumulated energy of all tasks having their arrival and deadline within an interval of length Δ. To this end, we need to shift the corresponding arrival curve by the relative deadline. We are doing this since the actual energy demand becomes due at the deadline of the respective task. In case of a periodic task J_i, this simply leads to:

$$\overline{\alpha}_i(\Delta) = \begin{cases} e_i \cdot \alpha_i(\Delta - d_i) & \Delta > d_i \\ 0 & 0 \leq \Delta \leq d_i \end{cases}$$

In case of several periodic tasks that arrive concurrently, the total demand curve $A(\Delta)$ (called demand-bound function in [Bar03]) can be determined by just adding the individual contributions of each periodic task, see Fig. 12:

$$A(\Delta) = \sum_{i \in I} \overline{\alpha}_i(\Delta)$$

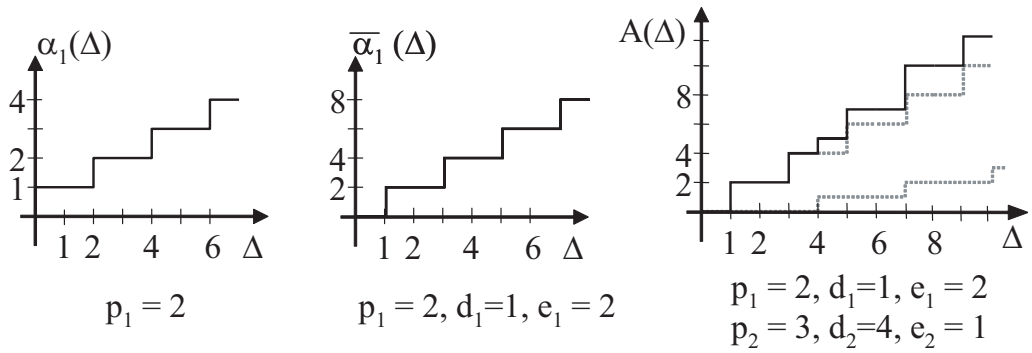


Fig. 12: Examples of an arrival curve $\alpha_i(\Delta)$, a demand curve $\overline{\alpha}_i(\Delta)$ and a total demand curve $A(\Delta)$ in case of periodic tasks.

Using the above defined quantities, we can formulate a schedulability test for the LSA algorithm that can be applied to a quite general class of tasks specifications.

Thm. 5: (LSA Schedulability Test) *A given set of tasks J_i, $i \in I$ with arrival curves $\alpha_i(\Delta)$, energy demand e_i and relative deadline d_i is schedulable under the energy-driven model with initially stored energy C, if and only if the following condition holds*

$$A(\Delta) \leq \min\left(\epsilon^l(\Delta) + C,\, P_{max} \cdot \Delta\right) \quad \forall \Delta > 0$$

Here, $A(\Delta) = \sum_{i \in I} e_i \cdot \alpha_i(\Delta - d_i)$ denotes the total energy demand of the task set in any time interval of length Δ, $\epsilon^l(\Delta)$ the energy variability characterization curve of the energy source, C the capacity of the energy storage and P_{max} the maximal processing power of the system. In case of periodic tasks we have $A(\Delta) = \sum_{i \in I} e_i \cdot \left\lceil \frac{\Delta - d_i}{p_i} \right\rceil$.

Proof. The proof of the if direction is omitted, since it is a direct consequence of Theorems 1 and 2. We just prove the only-if direction.

Remember that the total demand curve $A(\Delta)$ denotes the maximal energy demand of tasks in any interval $[t_1, t_2]$ of length Δ. It equals the maximal accumulated energy of tasks having their arrival **and** deadline within $[t_1, t_2]$. Therefore, in order to satisfy all deadlines for these tasks, at least energy $A(t_2 - t_1)$ must be available.

Let us suppose that the condition in Theorem 5 is violated for some Δ due to missing energy. Let us suppose also that the task arrival curve and the energy variability characterization curve are strict, i.e., there exists some time interval $[t_1, t_2]$ where the energy demand is $A(t_2 - t_1)$ and at the same time the energy $E_S(t_2 - t_1)$ is received. Then in time interval $[t_1, t_2]$ with $\Delta = t_2 - t_1$ the difference between the energy demand and the received energy is larger than the maximal stored energy C as $A(\Delta) > \epsilon^l(\Delta) + C$. As a result, the task set is not schedulable.

On the other hand, whenever the demanded computation time $\frac{A(\Delta)}{P_{max}}$ of a task set in the interval Δ is larger than the interval itself, a task set is not schedulable. Therefore it is evident, that both the energy condition $A(\Delta) \leq \epsilon^l(\Delta) + C$ and the time condition $A(\Delta) \leq P_{max} \cdot \Delta$ must be fulfilled in order to avoid deadline violations.

□

Theorem 5 tells us that we can decouple energy and time constraints if we have to decide whether a task set is schedulable or not. On the one hand, only if

$$P_{max} \geq \max_{0 \leq \Delta} \left(\frac{A(\Delta)}{\Delta} \right),$$

the system is fast enough to process the given task set. This condition is independent of the energy provided by the environmental source (i.e. ϵ^l) and the capacity of the storage device. Even increasing the capacity does not help. If a task set however satisfies the time constraint, the role of the capacity C as a design parameter for the energy harvesting system becomes important.

Suppose now that the time constraint is fulfilled. For this case, Theorem 5 states that the capacity C needed to schedule a task set with $A(\Delta)$ using a source with $\epsilon^l(\Delta)$ is constrained by

$$C \geq \max_{0 \leq \Delta} \left(0, A(\Delta) - \epsilon^l(\Delta) \right).$$

Fig. 13 illustrates an example schedulability test. The left diagram displays the total demand curve $A(\Delta)$ for two periodic tasks with p_1=2, d_1=1, e_1=2 and p_2=3, d_2=4, e_2=1. Furthermore, the EVCC $\epsilon^l(\Delta)$ is given by

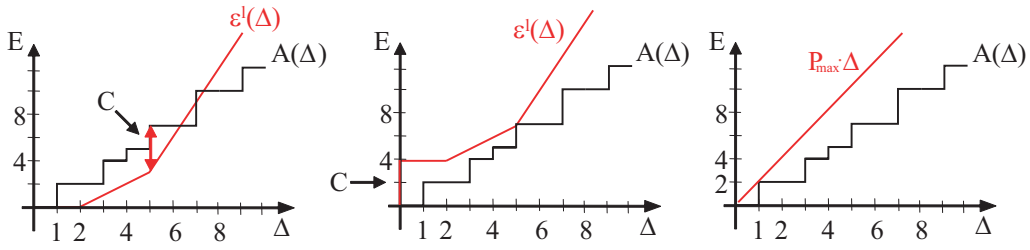


Fig. 13: Determining the optimal tuple $(C; P_{max})$ according to theorem 5.

a piecewise linear function using three pieces (0,0,0),(2,0,1),(5,3,3), where each piece i is defined by the triple of the form (initial Δ_i, initial $\epsilon^l(\Delta_i)$, slope of piece i). Now, the maximal difference between the total demand curve A and the EVCC ϵ^l can be computed. It has value 4 and is obtained at $\Delta = 5$. Therefore, one can conclude that the set of tasks can be scheduled (with LSA) using a minimal capacity $C = 4$. The respective schedulability test with $C = 4$ is shown in the middle diagram of Fig. 13. As displayed in the right diagram, power $P_{max} = 2$ is required to fulfill the time condition in theorem 5.

As a last point to mention, let us consider the middle diagram in Fig. 13 once again. Regarding the slopes of the curves, we can guess that A and ϵ^l won't intersect after the critical time interval of length 5. Formally, this is because the minimum average power $\lim_{\Delta\to\infty} \frac{\epsilon^l(\Delta)}{\Delta}$ is higher than the maximum average power demand $\lim_{\Delta\to\infty} \frac{A(\Delta)}{\Delta}$ of the task set. Simply stated, the average harvested power is higher than the average power demand of the application. It becomes evident that an optimal algorithm, like LSA, can only optimize the short-term behaviour of the system by suitable power management. LSA achieves the minimum capacity C needed to temporarily buffer energy for single tasks, but on the long run the average of power P_D is dictated by the task set.

2.5.2 Comparison to EDF

It is useful to formally compare LSA with the well know EDF algorithm in terms of the schedulability region and the required capacity C. In order to simplify the discussion, we will investigate an energy limited scenario only, see Section 2.4.1 and 2.4.3. Then we can state the following result:

Thm. 6: **(EDF Schedulability Test)** *A given set of tasks J_i, $i \in I$ with arrival curves $\alpha_i(\Delta)$, energy demand e_i and relative deadline d_i is schedulable with initially stored energy C, only if the following condition holds for any deadline d_k, $k \in I$:*

$$\sum_{i\in I} e_i \cdot \alpha_i(\Delta - d_k) \leq C + \epsilon^l(\Delta) \quad \forall \Delta > 0$$

In case of periodic tasks with period p_i we have $\alpha_i(\Delta) = \lceil \frac{\Delta}{p_i} \rceil$.

Proof. Remember that the left hand side of the condition denotes the maximal energy used by all the tasks in any interval $[t_1, t_2]$ of length $t_2 - t_1 = \Delta - d_k$. There is an instance of task arrivals compliant with the arrival curves α_i such that task J_k arrives at time t_2, i.e. by correctly adjusting the phase of all instances of task J_k. In this case, the deadline of the task instance arriving at t_2 is $t_2 + d_k$. In order to be able to execute this instance within its deadline, the available energy in any interval $[t_1, t_2 + d_k]$ must be larger than $\sum_{i \in I} e_i \cdot \alpha_i(t_2 - t_1)$, i.e. the energy used by tasks arriving in $[t_1, t_2]$. The maximal energy available in $[t_1, t_2 + d_k]$ is in the worst case given by $C + \epsilon^l(t_2 + d_k - t_1)$. Replacing $t_2 - t_1$ by $\Delta - d_k$ yields the desired result.

□

The strongest bound is obtained by using the task J_k with the smallest deadline $d^{min} = \min_{i \in J}\{d_i\}$. Comparing Theorems 5 and 6 in the energy-constraint case we obtain the two constraints $\sum_{i \in I} e_i \cdot \alpha_i(\Delta - d^{min}) \leq C + \epsilon^l(\Delta)$ for EDF and $\sum_{i \in I} e_i \cdot \alpha_i(\Delta - d_i) \leq C + \epsilon^l(\Delta)$ for LSA. Clearly, EDF has a smaller schedulability region as $\sum_{i \in I} e_i \cdot \alpha_i(\Delta - d^{min}) \geq \sum_{i \in I} e_i \cdot \alpha_i(\Delta - d_i)$ for all $\Delta \geq 0$.

Finally, let us derive specialized results in the case of periodic tasks J_i with $p_i = d_i$ (period equals deadline) and a simple energy variability characterization curve $\epsilon^l(\Delta)$ shown in Fig. 14 with $\epsilon^l(\Delta) = \max\{\sigma \cdot (\Delta - \rho), 0\}$.

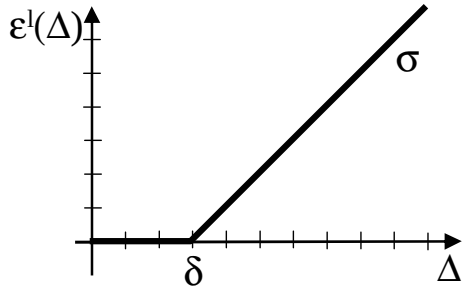


Fig. 14: Simple EVCC for comparing EDF and LSA scheduling.

We also suppose that the available average power σ from the energy source is sufficient to support the long term power demand $\bar{\sigma}$ of the task set

$$\sigma \geq \bar{\sigma} = \sum_{i \in I} \frac{e_i}{p_i}$$

as otherwise, deadline violations are unavoidable. In the following comparison, we suppose that the energy source has a minimal average power, i.e. $\sigma = \bar{\sigma}$, i.e. it is as weak as possible. Under these assumptions (periodic task with periods equal deadlines, energy-limited scenario) and using the

results of Theorems 5 and 6, one can compute the minimal possible capacities of the energy storage for the two scheduling methods LSA and EDF as follows:

$$C^{EDF} = \sum_{i \in I} e_i + (\delta - p^{min}) \cdot \bar{\sigma} \qquad ; \qquad C^{LSA} = \delta \cdot \bar{\sigma}$$

Therefore, the relative gain in the necessary storage capacity between the two scheduling methods can be quantified and bounded by

$$0 \leq \frac{C^{EDF} - C^{LSA}}{C^{LSA}} = \frac{1}{\delta \cdot \bar{\sigma}} (\sum_{i \in I} e_i + p^{min} \cdot \bar{\sigma}) \leq \frac{p^{max} - p^{min}}{\delta}$$

For the bounds we use the fact that $p^{min} \leq (\sum e_i)(\sum (e_i/p_i)) \leq p^{max}$ where p^{min} and p^{max} denote the minimal and maximal period of tasks J_i, respectively. In other words, the maximal relative difference in storage capacity depends on the differences between the task periods. The larger the difference between the largest and smallest period is, the large the potential gain in storage efficiency for the LSA algorithm.

2.6 Simulation Results

In the previous section, a method to compute the minimum capacity for a certain energy source characterization ϵ^l was presented. In the following, we will call this optimal value C_{min}. The value C_{min} obtained represents a lower bound since it is obtained for energy-clairvoyant LSA scheduling. In addition, it remains unclear, which capacities C^*_{min} are required if other scheduling disciplines are applied. For this reason, we performed a simulative study to evaluate the achievable capacity savings in a more realistic scenario. The EDF algorithm – which is optimal in traditional scheduling theory– serves as a benchmark for our studies.

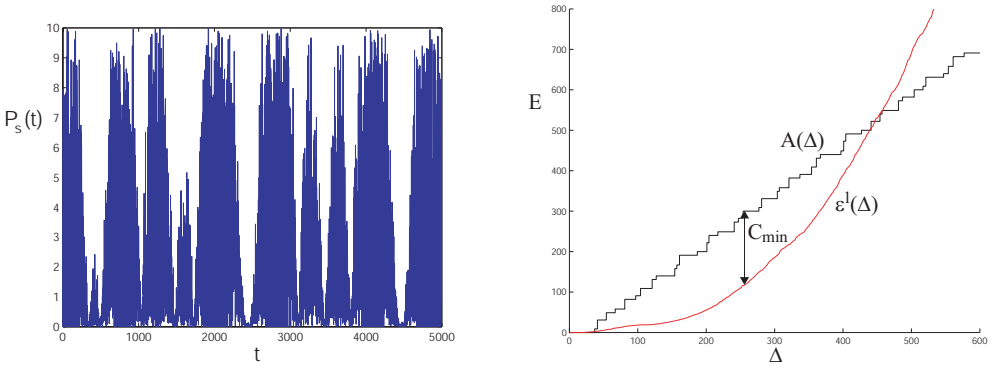
We investigated variants of LSA which utilize the measured EVCCs ϵ^l and ϵ^u to *predict* the future generated energy $E_S(t)$ for the LSA algorithm. Each time a starting time s_i has to be calculated for the task i with the earliest deadline d_i, the energy $\epsilon^l(d_i - a_i)$ (or $\epsilon^u(d_i - a_i)$) plus the stored energy $E_C(a_i)$ is assumed to be processible before the deadline.

The trace of the power source $P_S(t)$ is generated by a random number generator according to

$$P_S(t) = \left| 10 \cdot N(t) \cdot \cos(\frac{t}{70\pi}) \cdot \cos(\frac{t}{100\pi}) \right| ,$$

where $N(t)$ denotes a normally distributed random variable with mean 0 and variance 1. The values of P_S have been cut off at the value $P_{S,max} = 10$.

As illustrated in Fig. 15(a), the obtained power trace $P_S(t)$ exhibits both stochastic and deterministic, periodic behaviour. The latter is simulating day and night periods similar to those experienced by solar cells in an outdoor environment. From this trace we compute the average power $\overline{P_S}$ as well as upper and lower EVCCs ϵ^u and ϵ^l.



(a) Power trace $P_S(t)$.

(b) Determining C_{min} for a random task set.

Fig. 15: Calculation of C_{min} in two steps: (1) Extract $\epsilon^l(\Delta)$ from $P_S(t)$ and (2) Compute C_{min} for every task set with respective energy demand $A(\Delta)$.

A task set consists of an arbitrary number of periodic tasks. Periods p are taken from a set $\{10, 20, 30, \ldots, 100\}$, each value having an equal probability of being selected. The initial phases φ are uniformly distributed between [0,100]. For simplicity, the relative deadline d is equal to the period p of the task. The energies e of the periodic tasks are generated according to a uniform distribution in $[0, e_{max}]$, with $e_{max} = \overline{P_S} \cdot p$.

We define the utilization $U \in [0, 1]$ of a scheduler as

$$U = \sum_i \frac{\frac{e_i}{\overline{P_S}}}{p_i}.$$

One can interpret U as the percentage of processing time of the device if tasks are solely executed with the average incoming power $\overline{P_S}$. A system with, e.g., $U > 1$ is processing more energy than it scavenges on average and will deplete its energy reservoir.

In dependence of the generated power source $P_S(t)$, N task sets are generated which yield a certain processor utilization U. For that purpose, the number of periodic tasks in each task set is successively incremented until the intended utilization U is reached. Hence, the accuracy of the utilization U is varying ±1% with respect to its nominal value.

At the beginning of the simulation, the energy storage is full. We set $P_{max} = 10$. The simulation terminates after 10000 time units and is repeated for 5000 task sets. In order to show the average behaviour of all task sets in one plot, we normalized the capacities C with the respective C_{min} of the task set. Fig. 15(b) shows the calculation of C_{min} for a random task set.

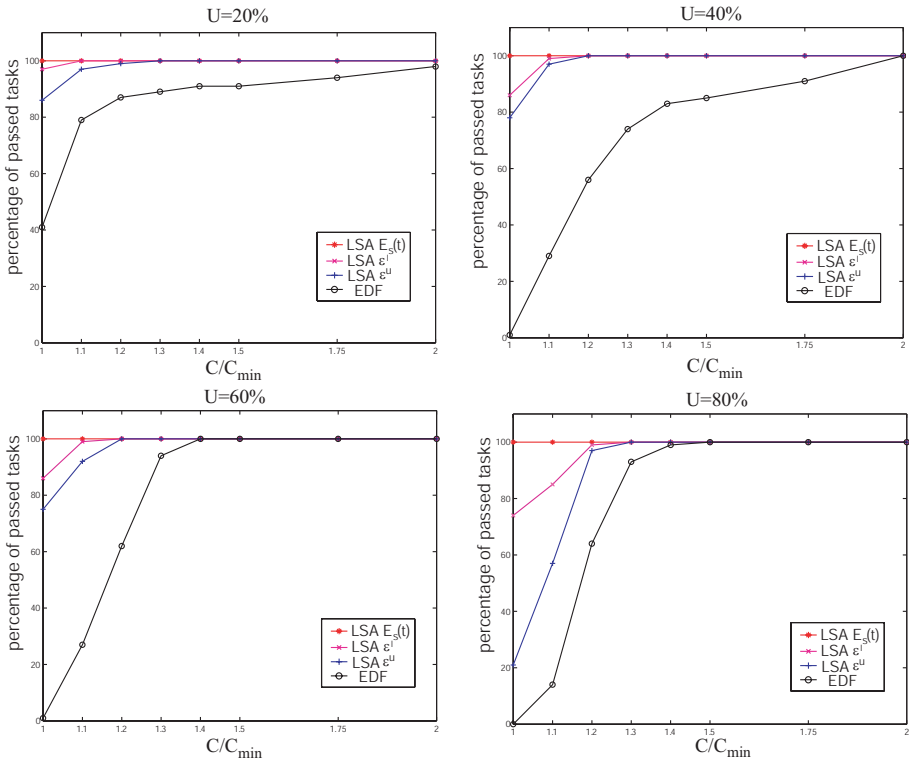


Fig. 16: Comparison of pure LSA, LSA using ϵ^l, LSA using ϵ^u and EDF for different utilizations U.

Fig. 16 illustrates the percentage of tasks that could be scheduled without deadline violations for different utilizations U. Clearly, no deadline violations occur for energy-clairvoyant LSA scheduling and values of $\frac{C}{C_{min}} \geq 1$. For all values of U, both approximations of LSA with ϵ^l and ϵ^u outperform the EDF algorithm, whereat the lower curve ϵ^l seems to be the better approximation. At $U = 40\%$ and $C = C_{min}$, e.g., almost no task set is schedulable with EDF. Here, LSA with ϵ^u is able to schedule $\approx 78\%$ of all task sets; LSA with ϵ^l even $\approx 85\%$.

Concerning the relative capacity savings achieved with our algorithms, we are especially interested in the smallest capacities C necessary to avoid any deadline violations. The highest savings are obtained at $U = 20\%$, where EDF needs more than $2.0 \cdot C_{min}$ to respect deadlines whereas LSA using ϵ^l shows the same behaviour at $1.1 \cdot C_{min}$. This trans-

lates into capacity savings of ≈ 45%. At higher values of the utilization U these savings are decreasing, yet they are still significant: At utilizations of $U = 40\%, 60\%, 80\%$, the capacity savings of LSA with ϵ^l compared to EDF are still ≈ 40%, 21%, 20%, respectively.

Albeit randomized task sets are not necessary representative for all kind of applications, these simulation results demonstrate that significant capacity savings are possible. If the application involves bursty instead of periodic task processing, the benefits of Lazy Scheduling may be indeed even more striking: As showed in [MBTB06], a greedy algorithm like EDF may violate an arbitrary number of deadlines and may suffer from worst case scenarios. This holds in particular for sensor nodes, where the energy demands of different tasks are highly varying (e.g. communication, sensing and data processing tasks) and tasks have to satisfy various timing constraints (e.g. urgent and less urgent tasks which have to run in parallel).

2.7 Practical Considerations

The system model introduced in Section 2.3 and used throughout this chapter implies idealized modelling abstractions, which demand further explanations. Therefore, this section is dedicated to general implementation aspects and possible application scenarios

2.7.1 Energy Source Predictability

Clearly, the performance of LSA is strongly dependent on the accuracy of the predicted power $P_S(t)$ of the harvesting unit. The better the approximation, the better the algorithm performs in terms of optimality. As illustrated by the simulation results of the previous section, energy variability characterization curves (EVCC) are suitable for that purpose. Especially for small utilizations U of the sensor node, EVCCs appear to converge towards the optimal, energy-clairvoyant LSA. It should be mentioned, that the prediction of $E_S(t)$ by EVCCs may even be improved if the sensor node is learning the characteristics of the energy source adaptively: By observing energy values $E_S(\Delta')$ for past intervals Δ', the prediction for future intervals Δ can be optimized online. This extension, however, increases at the same time the computational demand of the scheduler, which is one of the advantages of using simple EVCCs.

Solar energy harvesting through photovoltaic conversion is deemed a number one candidate for the power source P_S described in our model. If we assume the sensor node to be placed in an outdoor environment, the impinging radiation is variable with time, but follows a diurnal as well

as annual cycle. Moreover, during short time intervals, the produced power P_S can be regarded as constant and sudden changes of the light intensity are improbable. Due to this specific nature of solar energy, a two-tiered prediction methodology is self-evident: On the one hand, long-run predictions must be made for less urgent tasks with rather late deadlines. Here, using exponential decaying factors to weight the history of recorded powers P_S is one possibility. An alternative is to combine daily and seasonal light conditions of the past with the knowledge about a sensor's environment and possible shadowing. One can think of a plurality of prediction mechanisms, which are clearly out of the scope of this work.

For urgent tasks with close deadlines within milliseconds or seconds, intelligent prediction algorithms may not be necessary. Here, tasks like, e.g., sending a few bits over the wireless channel may be planned assuming constant power $P_S(t) = P_{S,const}$ during $s_i \leq t \leq d_i$. For stationarity of the power inflow P_S the calculation of the starting time s_i for a task i simplifies to

$$s_i = d_i - \min\left(\frac{E_C(a_i) + (d_i - a_i)P_{S,const}}{P_{max}}, \frac{C}{P_{max} - P_{S,const}}\right).$$

In the worst case, a sensor node is powered by an energy source with pure stochastic behaviour. If nothing is known about this source, the currently stored energy E_S is the only indicator for making scheduling decisions. By iteratively updating the starting time $\hat{s}_i = d_i - \frac{E_C(t)}{P_{max}}$ (and thereby increasing the computational overhead) starting task i too early can be avoided. However, once the device is running with P_{max}, the incoming energy $E_S(\hat{s}_i, d_i)$ may not be processible in the remaining interval $[\hat{s}_i, d_i]$. Consequently, optimality cannot be guaranteed for this scenario since the starting time $\hat{s}_i$ is always earlier then the optimal starting time s_i.

2.7.2 Task Processing

The task processing model presented in this chapter exhibits two major assumptions:

1. We assume the power $P_D(t)$ driving a task to be *continuously* adjustable with respect to its value in $[0; P_{max}]$ as well as with respect to time t. That is, at any point in time a task can be advanced with an accurately defined amount of power.

2. We assume a *linear* relationship between the power P_D used for executing a task and the execution time w. We can say: the higher the power P_D, the shorter the execution time w.

The first modelling assumption is only needed in situations when the energy storage is full ($E_C(t) = C$). In practice, there is no existing hardware that supports a continuous consumption of the scavenged power $P_S(t)$ as claimed by LSA. A microcontroller, e.g., drains roughly constant power from the battery when running a piece of program. The same holds for the radio interface. When transmitting a certain amount of data, most radios won't operate properly with unstable power supply. Therefore, we assume that the respective hardware attempts to approximate the power level of the power source by continuously switching power on ($P_D = P_{max}$) and off ($P_D = 0$). In Fig. 17, the achieved average power $\overline{P_D} \approx P_S$ is sketched.

It becomes evident that in an implementation, one will have to respect a certain granularity. The scheduler needs to determine when the energy storage is full and then, a task is executed for a given interval of time Δt which results in an energy consumption of $\Delta E = P_{max} \cdot \Delta t - E_S(\Delta t)$. For a microcontroller or a sensing unit, the time intervals Δt can be considered rather short while radio communication may require larger Δt due to packetized nature of the transmitted data. During the subsequent idle time, the stored energy is recovering again. In the worst case, this "duty cycling" results in a stored energy that is reduced by

$$\Delta E_{max} = P_{max} \cdot \max\{\Delta t\} - \epsilon^l(\max\{\Delta t\}))$$

in comparison to ideal LSA. With $\max\{\Delta t\}$ we denote the maximum period of continuous processing that can be observed for a given task. In terms of the admittance test in section 2.5, a task set is schedulable if it is schedulable under ideal LSA with reduced capacity $C - \Delta E_{max}$.

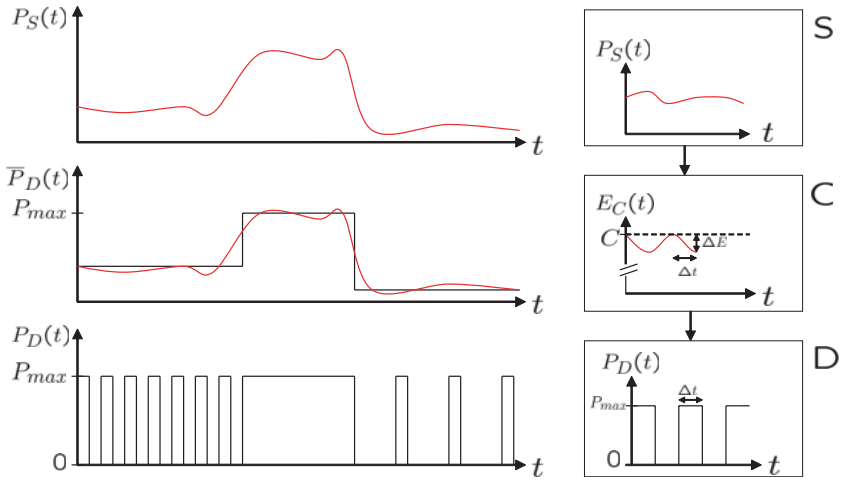


Fig. 17: Approximated power consumption $\overline{P_D} \approx P_S$ by means of duty cycling and resulting non-ideal storage level $E_C(t) = C - \Delta E$.

In the light of the considerations regarding assumption 1, also assumption 2 becomes plausible. Unlike common power management solution like Dynamic Voltage Scaling DVS, the energy e_i consumed by a task is the same regardless of the power P_D used during its execution. Although attractive due to its potential to *save* energy, DVS will come to its end if feature sizes of ICs get smaller. Hence, DVS or similar techniques were not considered in our work. In contrast, time and energy are assumed to be directly proportional – as indicated in Fig. 17 – with power $\overline{P_D}$ as constant of proportionality. The greater the number and size of time slots allocated to a given task, the higher the average power $\overline{P_D}$ and the faster the execution.

2.7.3 Energy Storage Model

An important step for the validation of the theory presented in this chapter is the discussion of the energy storage model. Looking at the various devices available on the market, there are two principal methods to store energy in a small volume or mass device: using an electro-chemical process or just performing physical separation of electrical charges across a dielectric medium. The first technique is used by rechargeable batteries and it is currently the most common and for long time it was the only method to achieve high capacities in a small size. Nevertheless, research in the last years has found new materials in order to increase the specific energy of capacitors, producing devices that are called supercapacitor or ultra-capacitor [KC00]. As shown in the so-called 'Ragone plot' in Fig. 18, supercapacitors offer a trade-off between power- as well as energy-density, filling the gap between batteries and capacitors. Beyond their ability to support higher power flows than batteries, supercapacitors overcome many other drawbacks of batteries: They have very long lifetimes and tolerate an almost unlimited number of charge/recharge cycles without performance degradation. Unlike batteries, no heat is released during charging/discharging due to parasitic, chemical reactions. In the following, we will focus on supercapacitors as possible candidates for an energy storage device.

2.7.3.1 Charge retention

Self-discharging is a natural phenomena that occurs in all kind of storage devices. It is caused by leakage currents that flow inside the device discharging it. Supercapacitors exhibit leakage currents that are typically in the order of magnitude of μA (see e.g. [Mt06]). Moreover, it should be mentioned that the leakage is proportional to the energy level. In [JPC05b], the leakage behaviour of different supercapacitors have been

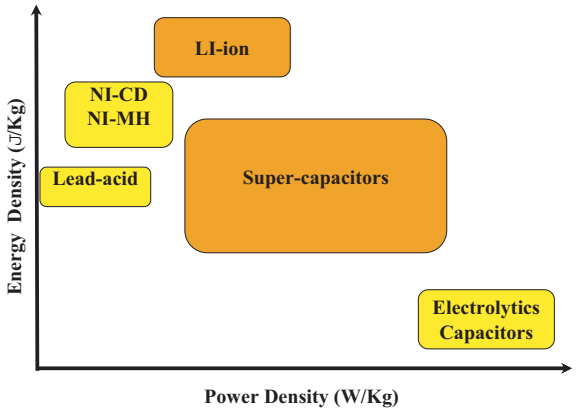


Fig. 18: Power and energy characteristics of storage devices.

tested. From Fig.2 in the latter work it becomes evident, that there exist a potential to minimize the leakage of fully charged supercapacitors by appropriate choice (manufacturer technology) and arrangement (serial, parallel) of devices.

Apart from the intrinsic energy leakage of supercapacitors, also the idle power consumption of the sensor node has to be considered. This "external leakage" can be reduced by switching to a low-power mode if no tasks are executed. In case of a low power wireless sensor node like Moteiv's Tmote Sky [Cor06], its ultra low power Texas Instruments MSP430 F1611 microcontroller exhibits a maximum current of $3.0\mu A$ in low power mode (LPM3). The wakeup to active mode is finished after $6\mu s$.

Altogether, it can be assumed that the energy conservation laws described in Section 2.3.2 hold and introducing an additional term allowing for energy leakage is dispensable.

2.7.3.2 Monitoring the stored energy

An important feature of energy harvesting systems is the capability to estimate the remaining energy in the storage device. Unlike batteries, the energy of supercapacitors can be measured in a straightforward way: The equations describing the physical behaviour of supercapacitors are nearly the same as the ones for ordinary capacitors, and the energy stored is hence $E_C \approx \frac{1}{2}CV^2$.

2.7.3.3 Storage efficiency

The efficiency η of a supercapacitor can be regarded as the quantity that relates the power flows and energies displayed in Fig. 18. Since charg-

ing a supercapacitor with P_S and discharging it with P_D can be seen as symmetrical operations, let us consider the efficiency η when the supercapacitor is charged. In this case, the increment of stored energy in time interval $\Delta = t_2 - t_1$ can be written as

$$E_C(t_2) - E_C(t_1) = \int_{t_1}^{t_2} P_S(t)\mathrm{d}t = \eta \int_{t_1}^{t_2} P_{S,raw}(t)\mathrm{d}t\,,$$

where $P_{S,raw}$ denotes the power fed into the supercapacitor. In general, supercapacitors may suffer from low efficiencies η due to their high equivalent series resistance [BR03]. This circumstance, however, does not jeopardize our modeling assumptions as such since, as stated in section 2.3.1, all losses are included in the definition of P_S. Actually, the more important property of the efficiency η is its independence of the energy level $E_C(t)$ at time t, which is approximately true for supercapacitors. Moreover, supercapacitors barely produce thermal heat which could reinforce non-linear charging/discharging behaviour.

In summary, we conclude that a linear charging/discharging behaviour with constant efficiency η is a reasonable abstraction for the example of a supercapacitor and hence our modeling assumptions hold. It should be mentioned, that possible variations of the efficiency η have been disregarded in related work like [JPC05b] and [KPS04], too.

2.8 Chapter Summary

We studied the case of an energy harvesting sensor node that has to schedule a set of tasks with real-time constraints. The arrival times, energy demands and deadlines of the tasks are not known to the node in advance and the problem consists of assigning the right amount of power in the right order to those tasks. For this purpose, we constructed optimal Lazy Scheduling Algorithms LSA which are energy-clairvoyant, i.e., the generated energy in the future is known. Contrary to greedy scheduling algorithms, LSA hesitates to power tasks until it is necessary to respect timing constraints. As a further result, we discuss an admittance test that decides, whether a set of energy-driven tasks can be scheduled on a sensor node without violating deadlines. This admittance test simultaneously shads light on the fundamental question of how to dimension the capacity of the energy storage: Provided that the average harvested power is sufficient for continuous operation, we are able to determine the minimum battery capacity necessary. Furthermore, achievable capacity savings between 20% and 45% are demonstrated in a simulative study, comparing the classical Earliest Deadline First algorithm with a variant of

LSA, which uses energy variability characterization curves EVCC as energy predictor. Finally, practical considerations are provided that suggest practical applicability of the theoretical results.

By starting to study a single node, we believe that extensions towards multihop networks where end-to-end deadlines have to be respected are possible. If sensors jointly perform a common sensing task, distributed energy management solutions are needed. An example for the common task could be redundantly deployed sensors with overlapping coverage regions.

3

Application Rate Control

In Chapter 2, an optimal real-time scheduling algorithm for energy harvesting systems was presented. Taking into account the available time as well as the harvested energy, an optimal *task ordering* was determined based on the prediction of the available energy. The parameters of the application in Chapter 2 are assumed to be given (in form of task arrival times, energy consumptions and deadlines). By scheduling the different tasks, short-term decisions on the use of environmental energy are made which do not affect the average power consumption.

In contrast, in this chapter we are *adapting parameters of the application* to optimize the performance in a long-term perspective. By adapting the activation rates of tasks, we control the average power consumption of the embedded system. We demonstrate that many optimization problems in energy harvesting systems can be modeled by the class of *linear programs*. As a main contribution, a framework for adaptive power management is proposed which builds on *multiparametric programming* techniques. In doing so, we link methods from control theory with the software design of environmentally powered systems. In addition, particular care has been taken to account for the unreliable nature of environmental energy. On the one hand, we address the design for worst-case situations in order to guarantee sustainable operation. On the other hand, an approximate multiparametric programming algorithm is presented which results in an acceptable system behaviour, avoiding an overly precise computation of the application parameters. In summary, we propose a low complexity and robust software design which is well-suited for resource constrained systems like, e.g., sensor nodes.

3.1 Introduction

Recently, there has been a substantial interest in the design of systems that receive their energy from regenerative sources such as solar cells. In contrast to approaches that minimize the power consumption subject to performance constraints, we are concerned with optimizing the performance of an application while respecting the limited and time-varying amount of available power. Based on a prediction of the future available energy, we adapt parameters of the application in order to maximize the utility in a longterm perspective. The chapter presents a formal specification of the corresponding optimization problem including constraints concerning buffer sizes, timing and rates. Instead of solving the optimization problem online which may be prohibitively complex in terms of running time and energy consumption, we apply multiparametric programming to pre-compute the application parameters offline for different environmental conditions and system states. In order to guarantee sustainable operation, we propose a hierarchical software design which comprises a worst-case prediction of the incoming energy. As a further contribution, we suggest a new method for approximate multiparametric linear programming which substantially lowers the computational demand and memory requirement of the embedded software. Our approaches are evaluated using longterm measurements of solar energy in an outdoor environment. Furthermore, we provide a detailed analysis of the implementation overhead in terms of computation and storage demand for embedded systems. For the particularly interesting application area of wireless sensor networks, we implement selected applications on a sensor node platform and measure the actual implementation overhead.

This chapter is organized as follows: In the next section, we discuss related works. In Section 3.3, a brief overview of the system concept is presented, followed by a detailed discussion of the models and corresponding notation in Section 3.4. Section 3.5 describes the basic principles of multiparametric linear programming and illustrates its application with practical examples. A hierarchical system design to separate decisions on energy usage and energy savings is the topic of Section 3.6. In Section 3.7, a new method for approximate multiparametric linear programming is presented. Finally, we have a detailed look at implementation issues in Section 3.8 before Section 3.9 concludes the chapter.

3.2 Related Work

In [KHZS07], the authors point out how the problem of adapting the duty cycle of a solar powered sensor can be modeled by a linear program. As

objective, the *average* duty cycle shall be optimized. Instead of periodically solving this linear program online, a heuristic algorithm of reduced complexity is proposed which attempts to decrease the duty cycle in times when the scavenged energy is low (e.g. at night) and increase the duty cycle when scavenged energy is high (e.g. during the day). In contrast, the class of linear programs presented in this chapter is capable of modeling a wider variety of application scenarios, constraints and optimization objectives. We are able to handle arbitrary objectives such as maximizing the *minimum* duty cycle. For the latter objective, we are able to achieve a more balanced system behaviour, i.e., we try to prevent an embedded system from shutting down during periods with little harvested energy. The work in [VGB07] improves on the results in [KHZS07]; however, the assumed optimization objective and application remain very specific.

The approach in [RMM03] is restricted to a very special scheduling problem: Periodic tasks with certain rewards are scheduled within their deadlines according to a given energy budget. The overall objective is to maximize the sum of rewards. Therefore, energy savings are achieved using Dynamic Voltage Scaling (DVS). The energy source is assumed to be solar and comprises a simplified day state as well as a night state. A similar reward maximization problem has been recently presented in [MCT08]. In the latter work, the received reward is assumed to be a concave function over the energy consumption. Again, our approach generalizes the work in [RMM03, MCT08] for a much larger variety of application scenarios and objective functions and the works in [RMM03, MCT08] can be seen as special cases of our approach.

3.3 System Concept

The system model is depicted in Figure 19. The whole hardware/software system is powered by an energy harvesting device that delivers in a unit time interval starting at t the energy $E_S(t)$. In the same time interval, the system uses energy $E_R(t)$. At time t, there is the stored energy $E_C(t)$ available. It is a typical feature of recent energy harvesting circuits that solar energy E_S is either used directly by the system or is stored in the energy storage for later use. In the latter case, the charging of the battery is done with some efficiency η whereas the direct usage is free of energy losses. This possibility of bypassing the energy storage can be exploited to *save* energy by minimizing the round trip losses in the energy storage device. For a discussion on how this system model matches real hardware implementations, the reader is referred to Section 3.8.3.

Besides the application, there are two additional software tasks running on the target architecture. The estimator predicts future energy

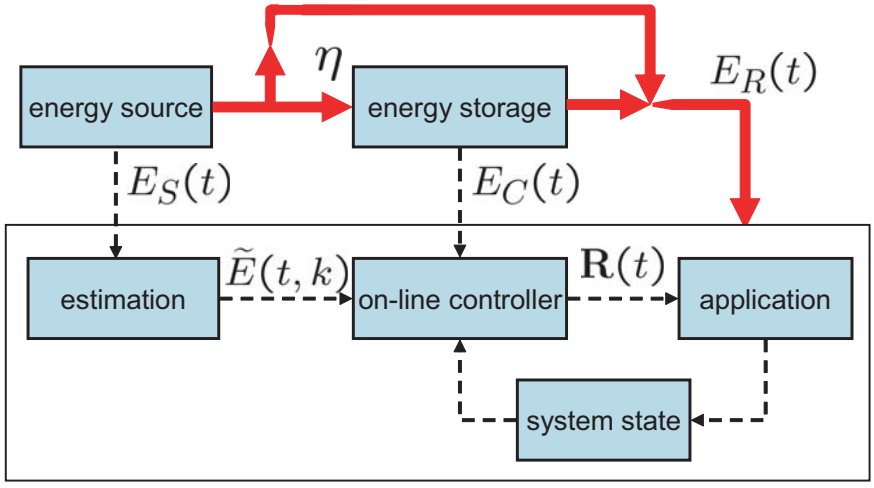


Fig. 19: Illustration of the system concept.

production of the harvesting device based on measurements of the past. The controller adapts properties of the application, e.g. task activation rates, based on the estimation of future available energy, the currently stored energy and additional information about the system state, e.g. the amount of available data memory. Parameters of the application are modified by the online controller. During execution, the system state (e.g. the amount of information stored in local memory and the stored energy) is changed.

It is the duty of the controller to adjust properties of the application such that longterm objectives are optimized (for example maximizing the sampling rate of a sensor) while respecting system constraints (for example using not more than the available memory). The complexity of the controller design is caused by the fact that (a) the harvested energy changes in time, (b) the available future energy can only be estimated and (c) the overall objective is usually a long term goal, e.g. maximizing the minimal sampling rate of a sensor in the future. For example, a sensor node powered by an outdoor solar cell may save energy during the day in order to have enough energy available at night for transmitting sensor data. But it may also store the sensor data at night and send them during the day. If one now also takes into account that it is more advantageous to use solar energy directly (e.g. at noon), complex controller strategies may be required.

3.4 Basic Models and Methods

3.4.1 Power Flow and Energy Storage Model

The modeling is based on the notion of discrete time $t \in \mathbb{Z}_{\geq 0}$ where the difference in physical time between two discrete time instances is denoted as T. Energy related sensing and control may happen only at times t. In a practical setting, one may have a basic time interval T of a few minutes or even an hour.

The energy harvesting device is modeled as a power source which delivers energy $E_S(t)$ in the time interval $[t, t+1)$ of length T. Therefore, in time interval $[t_1, t_2)$ with $t_1, t_2 \in \mathbb{Z}_{\geq 0}$ it delivers energy $E_S(t_1, t_2) = \sum_{t_1 \leq u < t_2} E_S(u)$. The incoming power can be stored in an energy storage device, e.g. a rechargeable battery or a supercapacitor. We denote η the storage efficiency of a non-ideal storage device. The energy level at time t is denoted as $E_C(t)$. In dependence on the energy $E_R(t)$ drawn from the sensor node, the increment $\Delta E_C(t)$ of the stored energy is defined at each time t according to the following statement:

$$\begin{aligned} \textbf{if } E_S(t) > E_R(t) \quad \textbf{then} \quad & \Delta E_C(t) = \eta \cdot (E_S(t) - E_R(t)) \\ \textbf{else} \quad & \Delta E_C(t) = (E_S(t) - E_R(t)) \end{aligned}$$

If the generated energy $E_S(t)$ is higher than energy $E_R(t)$, the sensor node is powered directly by the solar cell and excess energy is used to replenish the energy storage. As in [KHZS07] and [MTBB08], we account for the efficiency η during the charging of the energy storage. As the arrangement in Figure 19 is fully symmetrical, one could also consider η when the storage is discharged. Alternatively, efficiencies of $\frac{\eta}{2}$ for charging and discharging are thinkable; from a control point of view, all three mappings of the efficiency η are equivalent.

The energy $E_R(t)$ is used to execute tasks on various system components. For sensor nodes, tasks may be as diverse as sensing, signal processing, A/D or D/A conversion, computing, or communicating. A task $\tau_i \in \mathbf{I}$ from the set of tasks $\mathbf{I}$ needs energy e_i for completing a single instance. We suppose that a task is activated with a time-variant rate $s_i(t)$, i.e. during the basic time interval T starting at t, the task is executed $s_i(t)$ times. Therefore, a task needs energy $E_i(t_1, t_2) = \sum_{t_1 \leq u < t_2} e_i \cdot s_i(u)$ in time interval $[t_1, t_2)$ for successful execution. Finally, we denote $\mathbf{S}(t)$ the vector of all task rates s_i at time t. The detailed application and task model will be described in the next section.

3.4.2 Rate-Based Application Model

As described in Section 3.3, parameters of the application are changed at run-time in order to optimally use the available energy in the future. In

this chapter, we restrict ourselves to a rate-based application model.

The application consists of tasks τ_i, $i \in \mathbf{I}$. A task is instantiated $s_i(t) \geq 0$ times in the interval of length T starting at time t and the execution of each instance needs energy e_i. The activation of tasks can be modeled by a rate graph whose nodes and edges represent tasks and activation relations, respectively. In particular, an edge from i to j denotes that task τ_i activates task τ_j. We may also say, that an edge (i, j) is activated r_{ij} times; such an activation is caused by the activation of task τ_i and leads to an activation of task τ_j.

A scaling factor σ_{ij} is associated to an edge (i, j) that represents how often task τ_j is activated for each activation of it. The default scaling is $\sigma_{ij} = 1$. If there are several edges leaving a node, then the sum of their rates equals the activation rate s of the source node, e.g. $s_1 = r_{12} + r_{13}$. This way, we can model a decision in the application, i.e. the execution of a task may either lead to the activation of one or another subsequent task. If there are several edges leaving from the same (graphical) location at some node then their activation rates are equal. This models the case that two subsequent tasks are activated with the same rate, i.e. $r_{12} = r_{13}$ if $(1, 2)$ and $(1, 3)$ have their tails at the same location.

The rate relations that are covered by a rate graph as defined above can be formulated as a set of linear (in)-equalities. Free variables $R_k(t)$ in this system are determined by the controller shown in Figure 19. In general, we can formulate the rate equations as

$$\mathbf{P} \cdot \mathbf{R}(t) + \mathbf{Q} = \mathbf{S}(t) \tag{3.1}$$

$$\mathbf{F} \cdot \mathbf{R}(t) + \mathbf{G} \geq \mathbf{0} \tag{3.2}$$

where $\mathbf{S}(t)$ is a vector containing all activation rates $s_i(t)$, $\mathbf{R}$ contains all controlled parameters $R_k(t)$ and $\mathbf{P}$, $\mathbf{Q}$, $\mathbf{F}$ and $\mathbf{G}$ are vectors and matrices of appropriate dimensions.

The next Figure 20 exemplifies the rate graph and gives an example for the associated rate equations.

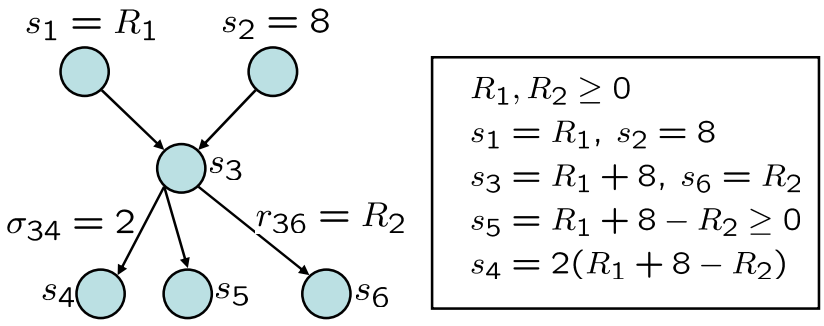


Fig. 20: Illustration of rate graph and rate equations.

3.4.3 Energy Prediction and Receding Horizon Control

According to the system model described in Section 3.3, the online controller receives energy estimations $\widetilde{E}(t,k)$ of the future harvested energy. Based on these predictions, the online controller computes future control parameters $\mathbf{R}$ which optimize the longterm behaviour of the system. In order to keep the control problem computationally tractable, both energy prediction and calculation of future control rates are planned for a finite horizon, leading to the concept of receding horizon control (RHC) [ML99].

The estimation unit receives tuples $(t, E_S(t))$ for all times $t \geq 1$ and delivers N predictions on the energy production of the energy source. We assume that the prediction intervals are of equal size denoted as the number L (in units of the basic time interval T). We denote the total prediction horizon $H = N \cdot L$ (again in units of the basic time interval T), see also Figure 21. At time t, the predictor produces estimations $\widetilde{E}_S(t + k \cdot L, t + (k+1) \cdot L)$ for all $0 \leq k < N$. We write $\widetilde{E}(t,k) = \widetilde{E}_S(t + k \cdot L, t + (k+1) \cdot L)$ as a shorthand notation, i.e. the estimation of the incoming energy in the $(k+1)$st prediction interval after t.

The prediction algorithm should depend on the type of the energy source and the system environment. Standard techniques known from automatic control and signal processing can be applied here. In this chapter, we will, e.g., provide prediction algorithms which are useful for solar cells that operate in an outdoor environment. The algorithms used for the experimental results will be presented in the respective sections.

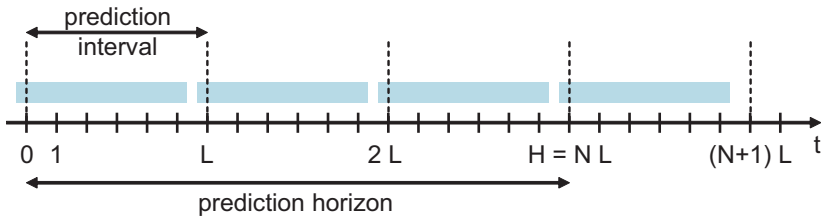


Fig. 21: Illustration of the prediction horizon.

At time t, the controller is computing the control sequence $\mathbf{R}(t + k \cdot L)$ for all prediction intervals $0 \leq k < N$ based on the estimates $\widetilde{E}(t,k)$ as well as the current system state (e.g. $E_C(t)$). In other words, the rates s_i of the different tasks are planned to be constant during each prediction interval. However, *only the first* control rates $\mathbf{R}(t)$ are applied to the system during the first time step T. The rest of the control sequence is discarded. At time $t + T$, a new vector $\mathbf{R}(t)$ is computed which extends the validity of the previous vector $\mathbf{R}(t-1)$ by one time step. Again only the first control is used, yielding a receding horizon control (RHC) strategy.

3.4.4 Linear Program Specification

The first step in constructing the online controller is the formulation of the optimization problem in form of a parameterized linear program (LP). Corresponding solution methods will be described in Section 3.5 and 3.7.

The equations and inequations of the linear program can be partitioned into the following two classes:

- *State Relations*: It is apparent from Figure 19 that executing a task changes the state of the system. This state may not only be the fill level of the energy storage but also, e.g., the used memory.

- *Reward Relations*: The reward relations describe the objective that needs to be optimized. Note that we will model the future evolution of the application and the system state over a finite horizon, i.e. for a time interval covered by the energy prediction. This way, it is possible to take into account long term expectations.

One of the essential states of the system is the stored energy. Using the power model described in Section 3.4.1 we obtain the following state equations:

$$\begin{aligned} E_C(t + k \cdot L) = & E_C(t) - k \cdot \zeta + \sum_{j=0}^{k-1} \left(\widetilde{E}(t, j)\right) \\ & - \sum_{j=0}^{k-1} \left(L \cdot \mathbf{E}^T \cdot \mathbf{S}(t + j \cdot L)\right) - (1 - \eta) \sum_{j=0}^{k-1} \lambda(j) \qquad (3.3) \\ \lambda(j) \geq & \, 0 \qquad (3.4) \\ \lambda(j) \geq & \, \widetilde{E}(t, j) - L \cdot \mathbf{E}^T \cdot \mathbf{S}(t + j \cdot L) \qquad (3.5) \end{aligned}$$

They determine the expected contents of the energy storage at times $t + kL$ for $1 \leq k \leq N$. $\mathbf{S}(t)$ is a vector containing all activation rates $s_i(t)$ and $\mathbf{E}^T$ is a (row)-vector that contains all energy requirements e_i for all $i \in \mathbf{I}$. Therefore, $\mathbf{E}^T \cdot \mathbf{S}(t) = \sum_{i \in \mathbf{I}} e_i \cdot s_i(t)$. The factor ζ accounts for energy leakage of the storage device. The auxiliary variable λ accounts for the physical switching behaviour described in Section 3.4.1: In intervals when the energy provided by the source is higher than the energy demand of the sensor node, the energy storage is charged with efficiency η. The other way round, the energy storage is discharged in intervals when $\widetilde{E}(t, j)$ is low and λ is forced to 0.

In a similar way, we can also model other system states, for example memory. A task could produce a certain amount of data that is stored and

another removes it, e.g. by means of communication to another node. In this case we would have for $1 \leq k \leq N$ the state equations:

$$M(t + k \cdot L) = M(t) + L \sum_{j=0}^{k-1} \mathbf{M}^T \cdot \mathbf{S}(t + j \cdot L) \tag{3.6}$$

$M(t)$ denotes the amount of stored data at time t and m_i is the amount of data produced or consumed by a task τ_i with rate s_i in a time interval of length T. $\mathbf{M}^T$ is a (row)-vector that contains all data amounts m_i for all $i \in \mathbf{I}$. Of course, equations (3.3) to (3.6) provide only examples of possible system states and their associated changes. Moreover, there may be constraints on the feasible states, for example

$$0 \leq E_C(t + k \cdot L) \leq E_{\max} \tag{3.7}$$

$$0 \leq M(t + k \cdot L) \leq M_{\max} \tag{3.8}$$

for $0 \leq k \leq N$.

One can now easily combine (3.1)-(3.8) and obtain a system of linear equalities and inequalities that contain as free variables $M(t + k \cdot L)$ (the state of the memory), $E_C(t + k \cdot L)$ (the state of the energy) for $1 \leq k \leq N$ and the rate control $\mathbf{R}(t + k \cdot L)$ for $0 \leq k < N$.

So far, no optimization goal has been formulated and therefore, any feasible rate control $\mathbf{R}(t + k \cdot L)$ could be a solution. Any linear objective function J that makes use of the free variables given above is possible in this case. One may also define additional variables in order to model specific objectives. One possible (very simple) example would be the objective

$$\begin{aligned} &\text{maximize } J = \mu \\ &s_1(t + k \cdot L) \geq \mu \qquad \forall 0 \leq k < N \end{aligned} \tag{3.9}$$

which would attempt to maximize the minimal rate with which the task τ_1 is operated in the finite horizon $0 \leq k < N$. This could for example be a task that gathers sensor data and it is desired that the minimal rate is as large as possible. In terms of intervals, the objective translates into a minimization of the maximum interval between any two consecutive measurements. Hence, one could apply this objective in scenarios where one attempts to minimize unobserved time periods like e.g. in environmental monitoring or intruder detection applications.

Clearly, for the objective in (3.9), the controller would continuously try to empty the storage at the horizon to obtain an optimal objective value. Therefore, we formulate a final state constraint that ensures energy neutral, sustainable operation:

$$E_C(t + N \cdot L) \geq E_C(t) + \alpha(t) \tag{3.10}$$

A common choice for the prediction horizon is $H \cdot T = 24h$, see e.g. [HZK+06] or [MTBB08]. In [HZK+06], the energy offset $\alpha(t)$ is set to 0 for all times t. Following a diurnal circle, solar energy is assumed to behave similarly on consecutive days. In [MTBB08], α is manually tuned to some fixed value. However, it has become evident that the choice of $\alpha(t)$ severely influences the performance of a system: decreasing $\alpha(t)$ results in a more aggressive control behaviour, running the risk to deplete energy E_C. On the other hand, increasing $\alpha(t)$ may lead to overly conservative rates **S**, poor performance and an high energy level E_C. In Section 3.6, a solution will be presented how α can be tuned automatically using an hierarchical control approach.

Following the well known concept of receding horizon control, we could optimize the linear program in (3.3)-(3.10) at each time step t and obtain the desired values for the free variables **R**. They are valid for the next time interval of length T. After this time interval t is incremented and we would receive new energy estimates $\widetilde{E}(t, j)$ and new state information $(E_C(t), M(t))$ which will be used to set up a new linear program, i.e. a linear program with the same structure but different parameters. Its solution will determine new controller variables $\mathbf{R}(t)$. This process is repeated for every time step t.

Obviously, solving at each time step t a linear program in a resource limited system is prohibitive in general. However, we can conclude that the optimal rate control can be determined by solving a parameterized linear program, where the parameters are $\widetilde{E}(t, j)$, $E_C(t)$ and $M(t)$. In the next section, we will describe an optimal method for solving the above parameterized linear program offline and using the result for constructing an optimal online controller. In Section 3.7, we will show how this online controller can be approximated to obtain a less precise but much simpler controller.

3.5 Multiparametric Control Design

Next, we will show how to design an online controller based on multiparametric linear programming (mp-LP) which avoids solving a linear program at each time step. Thereby, we are following the ideas in [BBM02], where the regulation of discrete-time constrained linear systems is studied in the context of model predictive control. In [MTBB07], the application of multiparametric linear programming has been proposed for the first time for energy harvesting systems. We will briefly recall the main results in

Section 3.5.1 and illustrate the approach with the help of simple examples in Sections 3.5.2 and 3.5.3. The last example presented in Section 3.5.4 shows the limits of the approach.

3.5.1 Controller Generation

As a first step, we define a state vector $\mathbf{X}$ consisting of the actual system state, the level of the energy storage as well as the estimation of the incoming energy over the finite prediction horizon (cp. Figure 19). Resuming the system dynamics formulated in (3.3)-(3.10), the state vector $\mathbf{X}$ can be written as

$$\mathbf{X}(t) = \left(E_C(t)\,,\, M(t)\,,\, \widetilde{E}(t,0)\,,\, \ldots\,, \widetilde{E}(t,N-1)\right)^T \tag{3.11}$$

Furthermore, let us denote the vector of optimal control inputs to the system, i.e., the vector of planned rates $\mathbf{R}$ as

$$\mathbf{U}^*(\mathbf{X},t) = \left(\mathbf{R}^T(t)\,,\, \mathbf{R}^T(t+L)\,,\, \ldots\,, \mathbf{R}^T\left(t+(N-1)\cdot L\right)\right)^T. \tag{3.12}$$

The state space of $\mathbf{X}$ (in our case $\mathbb{R}^{N+2}$ bounded by possible constraints on $E_C(), M()$ and $\widetilde{E}()$) can now be subdivided into a number N_{CR} of polyhedrons. For each of these polyhedrons j (also called critical regions) the optimal solution $\mathbf{U}^*(\mathbf{X})$ of the control problem can be made available explicitly as

$$\mathbf{U}^*(\mathbf{X}) = \mathbf{B}_j\mathbf{X} + \mathbf{C}_j \quad \text{if } \mathbf{H}_j\mathbf{X} \leq \mathbf{K}_j, j = 1,\ldots,N_{CR} \tag{3.13}$$

where $\mathbf{B}_j \in \mathbb{R}^{N\times(N+2)}, \mathbf{C}_j \in \mathbb{R}^N$ and $\mathbf{H}_j\mathbf{X} \leq \mathbf{K}_j, j = 1\ldots N_{CR}$ is a polyhedral partition of the state space of $\mathbf{X}$. For simplicity, we dropped the dependence on t of the state vector $\mathbf{X}$. The computation of the vectors and matrices of control law (3.13) is done offline using, e.g., the algorithm presented in [BBM03] or other efficient solvers cited in the latter work.

In the online case, the controller has to identify to which region j the current state vector $\mathbf{X}$ belongs. After this membership test, the optimal control moves $\mathbf{U}^*$ for the next N prediction intervals may be computed by evaluating a linear function of $\mathbf{X}$. However, according to the receding horizon policy it is sufficient to calculate only the first rates $\mathbf{R}(t)$ for the next interval. These rates $\mathbf{R}(t)$ are identical to the rates one would obtain by solving the linear program. However, the computational demand is greatly reduced compared to solving a LP online. After having solved the mp-LP in advance, a set of N_{CR} polyhedra with associated control laws has to be stored and evaluated at each time step t. The computation demand in the online case now depends on

- the number of critical regions N_{CR} which have to be tested,

- the size of the state vector $\mathbf{X}$ (in particular the number of prediction intervals N),
- and finally on the number of controlled rates $\mathbf{R}$ which have to be determined.

If the number of critical regions N_{CR} gets large, the computational effort still may be large as many tests of the form $\mathbf{H}_j\mathbf{X} \leq \mathbf{K}_j$ must be performed. Typically, the computational effort spent for these matrix multiplications is much higher than evaluating the linear function $\mathbf{B}_j\mathbf{X} + \mathbf{C}_j$. In related work, there have been proposals how the online complexity of a given control law (3.13) can be reduced. In [BBBM01], e.g., this issue has been addressed by finding a representation of the polyhedral partition which allows more efficient region testing. In [MTBB07], the average number of region tests could be reduced with the help of probabilistic region checking. Nevertheless, those techniques come to their end if the number N_{CR} of critical regions is high. As we will see in Section 3.5.4, such an explosion of regions may happen for applications of practical concern. Indeed, this general shortcoming of the mp-LP approach may render the calculated controllers inapplicable for resource constrained systems. By dividing the problem in subproblems within a hierarchical framework (see Section 3.6) and by proposing an approximate, sub-optimal multiparametric solver (see Section 3.7) we will show how complex control problems can be mastered anyhow. To this end, we will reduce the number of critical regions N_{CR} for a given control problem.

Multiparametric programming approaches are not limited to problems with continuous variables and linear objectives as formulated in equations (3.1) – (3.10). If the performance index is quadratic, the optimization problem can be solved using multiparametric quadratic programming (mp-QP). An efficient algorithm to compute the explicit state feedback controller in case of mp-QP has been presented in [BMDP02]. In addition, techniques for the broader class of hybrid systems have been discussed in literature. Here, the authors of [BBM00] address optimization of systems with continuous as well as discrete dynamics, leading to the concept of multiparametric mixed-integer linear programming (mp-MILP) and multiparametric mixed-integer quadratic programming (mp-MIQP), respectively.

3.5.2 Adaptation of Sensing Rate (Example I)

We implemented online controllers for exemplary case studies using the MATLAB toolbox in [KGB04]. Measurements of solar light intensity $[\frac{W}{m^2}]$ recorded at [Sun06] serve as energy input $E_S(t)$. Of course, one would have to scale the measured power profile with the size, number and efficiency

of the actually used solar panels. The energy *prediction* algorithm that has been used is the same as in [MTBB07]. It is similar in nature to the predictor used in [HZK+06] and attempts to predict the most probable, *average* energy values for the prediction intervals.

Let us assume the following example: A sensor node is expected to measure some physical quantity like, e.g., ambient temperature or mechanical vibrations and has to transmit the sampled data to a base station. We can model these requirements as a single sensing task τ_1 with rate $R_1(t)$, i.e., a task which is instantiated R_1-times in the interval $[t, t+T)$. For the sake of simplicity, the sensing task τ_1 drains at every instantiation 1 energy unit from the battery. Assume further that we want the maximum interval between two consecutive reports to be as small as possible, as in (3.9). Assuming an ideal energy storage process with an efficiency $\eta = 1$ and no leakage $\zeta = 0$, we can formulate the linear program LP I as shown below. Note that the last inequality in LP I is used to stabilize the receding horizon controller.

maximize $J = \lambda$ subject to:	(LP I)
$R_1(t + k \cdot L) \geq \lambda$	$\forall 0 \leq k < N$
$E_C(t + k \cdot L) = E_C(t) + \sum_{j=0}^{k-1} \left(\widetilde{E}(t, j) - L \cdot R_1(t + j \cdot L) \right)$	$\forall 1 \leq k \leq N$
$E_C(t + k \cdot L) \geq 0$	$\forall 1 \leq k \leq N$
$E_C(t + N \cdot L) \geq E_C(t) - 100$	

In general, the number of partitions N_{CR} of a multiparametric solution grows with the size of the state vector $\mathbf{X}$. Hence, it is of practical concern to keep the number of prediction intervals $\widetilde{E}(t, i)$ and therewith the dimension of $\mathbf{X}$ as small as possible. We chose $L = 24$ and $N = 6$ and obtain the states $\mathbf{X}(t) = \left(E_C(t), \widetilde{E}(t, 0), \ldots, \widetilde{E}(t, 5) \right)^T$. The resulting online controller consists of $N_{CR} = 7$ partitions. Figure 22 visualizes the partition $\mathbf{H}_i \mathbf{X}(t) \leq \mathbf{K}_i, i = 1, \ldots, 7$ for an arbitrarily chosen set of parameters $\widetilde{E}(t, 2) - \widetilde{E}(t, 5)$.

In Figure 23, the generated controller is optimizing the sensing rate R_1 over a time period of 7 days. We started the simulation with an energy level $E_C(0) = 500$ and found a nearly constant rate R_1 during the whole simulation time. On the other hand, the stored energy $E_C(t)$ is highly varying, since the controller successfully compensates the unstable power supply $E_S(t)$. As a consequence, the stored energy $E_C(t)$ is increasing during the day and decreasing at night. Even the 4th displayed day with significant less sunshine is not jeopardizing the sensing rate R_1. Figure 23 demonstrates that the online controller manages to meet the optimization goal for this simple example.

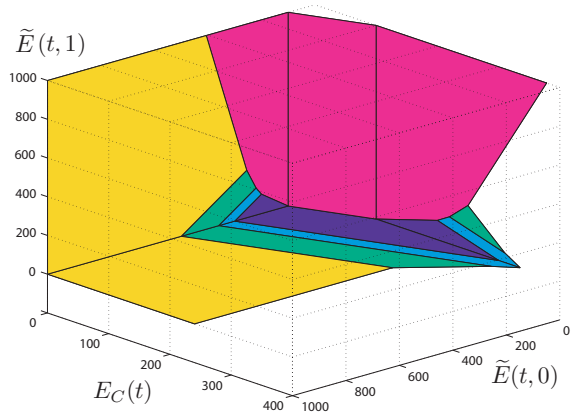


Fig. 22: 3-dimensional view of the polyhedral partition of the state space **X** for LP I. Cut through $\widetilde{E}(t,2) = 100.0, \widetilde{E}(t,3) = 120.0, \widetilde{E}(t,4) = 300.0, \widetilde{E}(t,5) = 10.0$.

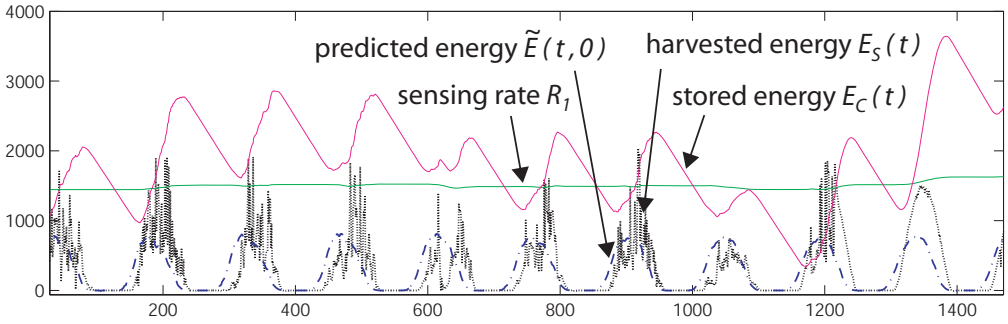


Fig. 23: Adaptation of the sensing rate R_1.

3.5.3 Local Memory Optimization (Example II)

Another simple scenario may consist of two tasks running on a sensor node: A first task τ_1 is sampling some physical quantity, i.e., performing an A/D-conversion and storing the data in some local memory. A second task τ_2 is transmitting stored samples with a rate R_2 and thereby frees memory from the storage device. Clearly, the scaling factor $\sigma_{12} = R_1/R_2$ represents the ratio with which the amount of stored data M is increasing ($\sigma_{12} < 1$) or decreasing ($\sigma_{12} > 1$).

For this application, two reasonable optimization objectives would be (a) to minimize the unobserved intervals between any two consecutive samples and (b) to minimize the amount of stored data M. The purpose of the second objective is twofold: On one hand, sensor nodes are usually small, inexpensive low power devices with constrained hardware resources such as memory. On the other hand, the objective may to some extent enforce the freshness of data arriving at the base station. In gen-

eral, radio communication is the main energy consumer on a sensor node. Hence we set the energies $e_1 = 0.1$ and $e_2 = 0.9$. The corresponding linear program LP II is given below.

maximize $J = (\lambda - \mu)$ subject to: (LP II)

$$
\begin{aligned}
R_1(t + k \cdot L) &\geq \lambda && \forall 0 \leq k < N \\
M(t + N \cdot L) &\leq \mu \\
E_C(t + k \cdot L) = E_C(t) + \textstyle\sum_{j=0}^{k-1} \widetilde{E}(t, j) - \sum_{j=0}^{k-1} (L \cdot [0.1 \;\; 0.9] \cdot \mathbf{R}(t + j \cdot L)) &&\forall 1 \leq k \leq N \\
E_C(t + k \cdot L) &\geq 0 && \forall 1 \leq k \leq N \\
M(t + k \cdot L) = M(t) + L \textstyle\sum_{j=0}^{k-1} [1 \quad -1] \cdot \mathbf{R}(t + j \cdot L) &&\forall 1 \leq k \leq N \\
M(t + k \cdot L) &\geq 0 && \forall 1 \leq k \leq N \\
E_C(t + N \cdot L) &\geq E_C(t) - 150
\end{aligned}
$$

To account for the additional system state $M(t)$ we reduced the number of prediction intervals and set $L = 36$ and $N = 4$. The state space of $\mathbf{X}(t) = \left(E_C(t), M(t), \widetilde{E}(t,0), \ldots, \widetilde{E}(t,3)\right)^T$ is divided by the control law in $N_{CR} = 39$ critical regions.

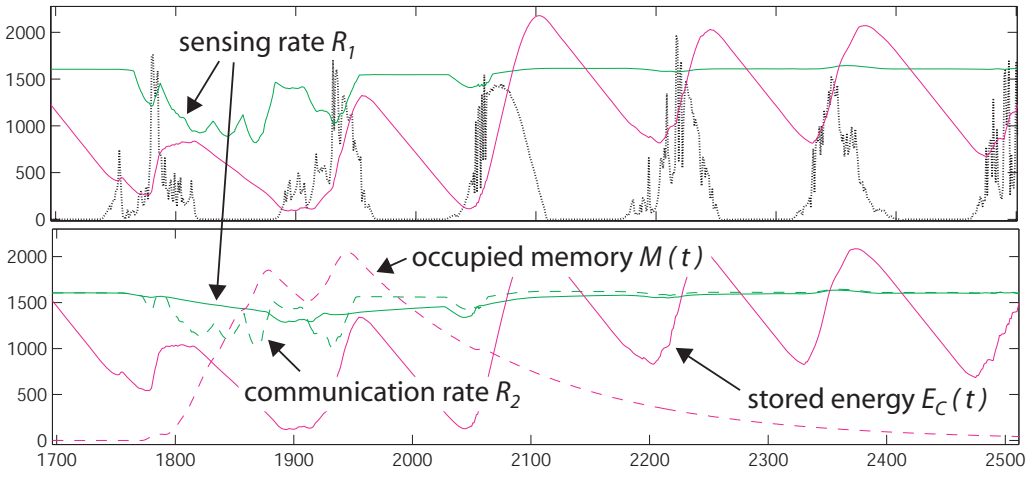


Fig. 24: Top: scenario without local memory. Bottom: scenario with optimized local memory.

Figure 24 displays the simulated curves of the state and control variables during 6 days. The figure at the top represents a scenario with no use of local memory, i.e. the control problem in LP I whereas the bottom figure shows a comparable scenario using local memory according to LP II. Until $t = 1700$, both tasks are adjusted to the same rate $R_1 = R_2$ and the memory $M(t)$ is empty. However, after two days with little harvested energy, the energy level $E_C(t)$ on the sensor node is falling and the controller starts to suspend the energy-costly communication task τ_2 by reducing rate R_2. Consequently, the number of buffered samples M is increasing starting

before $t = 1800$. In the following, the controller achieves to autonomously regulate the tradeoff between $E_C(t)$ and $M(t)$. While a lack of $E_C(t)$ would cause the non-initiation of task τ_1, an increasing $M(t)$ directly affects the second optimization objective. After $t = 1950$, the controller reduces the amount of occupied local memory by increasing the rate R_2. It becomes obvious that in terms of the minimum sampling rate R_1, the controller for LP II outperforms the controller for LP I. The minimum rate $R_{1,\min} \approx 820$ for LP I is significantly lower than the minimum rate $R_{1,\min} \approx 1280$ for LP II.

3.5.4 Optimization with Non-Ideal Energy Storage (Example III)

In this section, we discuss a more involved example to pinpoint the limits of a pure multiparametric control approach. Firstly, we show that explosions of the polyhedral partition of the online controller may happen for practical examples. And secondly, we show that care has to be taken when choosing the energy prediction algorithm, since the robustness of the overall system severely depends on it.

maximize $J = \mu$ subject to:	(LP III)
$R_1(t + k \cdot L) \geq \mu$	$\forall 0 \leq k < N$
$R_1(t + k \cdot L), R_2(t + k \cdot L) \geq 0$	$\forall 0 \leq k < N$
$E_C(t + k \cdot L) = E_C(t) + \sum_{j=0}^{k-1} \left(\widetilde{E}(t, j) - (L \cdot [0.1 \;\; 0.9] \cdot \mathbf{R}(t + j \cdot L)) - (1 - \eta)\lambda(j) \right)$	$\forall 1 \leq k \leq N$
$\lambda(j) \geq \widetilde{E}(t, j) - (L \cdot [0.1 \;\; 0.9] \cdot \mathbf{R}(t + j \cdot L))$	$\forall 0 \leq j < N$
$\lambda(j) \geq 0$	$\forall 0 \leq j < N$
$E_C(t + k \cdot L) \geq 0$	$\forall 1 \leq k \leq N$
$M(t + k \cdot L) = M(t) + \sum_{j=0}^{k-1} L \cdot [1 \;\; -1] \cdot \mathbf{R}(t + j \cdot L)$	$\forall 1 \leq k \leq N$
$0 \leq M(t + k \cdot L) \leq M_{\max}$	$\forall 1 \leq k \leq N$
$E_C(t + N \cdot L) \geq E_C(t)$	

Example III is similar to Example II. The main difference is that we now consider an energy storage device with a non-ideal roundtrip efficiency $\eta = 0.8$. Assume a video surveillance application where a first task τ_1 is recording images, performing some image compression and stores the data in some local memory. The task is instantiated with rate $r_1(t)$ and at every instantiation, one data unit is stored. The maximum storage capacity of the sensor node is $M_{\max} = 1000$. A second task τ_2 with rate $r_2(t)$ frees memory by transmitting images to a base station. We choose again $e_1 = 0.1$ and $e_2 = 0.9$ as energy demands of the tasks $r_1(t)$ and $r_2(t)$, respectively. For this application, we want to minimize the unobserved

intervals between any two consecutive images (i.e. activations of task τ_1). The corresponding linear program LP III is given above.

We performed the experiments using longterm measurements of solar light intensity measured in [sol07] and shorter time intervals of $T = 5$ minutes. The energy predictor used is the same as in the previous sections. We set the prediction horizon to $H \cdot T = 24h$ and subdivided the horizon in $N = 6$ intervals of length $L = 48$. For the optimization problem in LP III, we compute the explicit solution via multiparametric programming for the state vector $\mathbf{X}(t) = \left(E_C(t), M(t), \widetilde{E}(t,0), \ldots, \widetilde{E}(t,5)\right)^T$. The resulting online controller consists of $N_{CR} = 1049$ partitions. Figure 25 visualizes the partition $\mathbf{H}_j\mathbf{X}(t) \leq \mathbf{K}_j, j = 1, \ldots, 1049$ for an arbitrarily chosen set of parameters $\widetilde{E}(t,1) - \widetilde{E}(t,5)$.

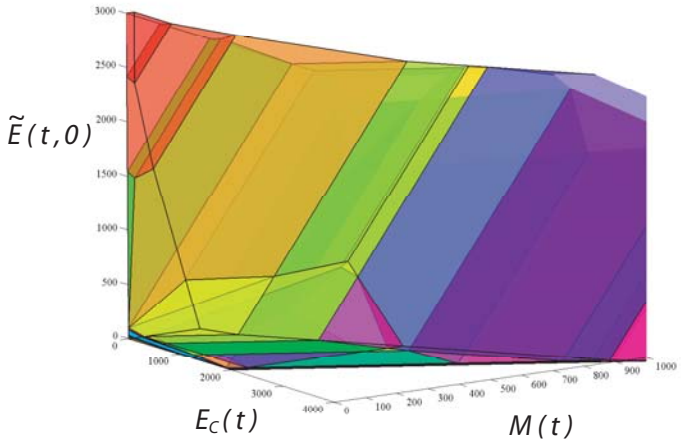


Fig. 25: 3-dimensional view of the polyhedral partition of the state space $\mathbf{X}$ for LP III. Cut through $\widetilde{E}(t,1) = 1200$, $\widetilde{E}(t,2) = 1000$, $\widetilde{E}(t,3) = 0$, $\widetilde{E}(t,4) = 100$ and $\widetilde{E}(t,5) = 500$.

The complete control law requires the storage of $(3N + 9) \cdot N_{CR}$ real numbers. In the worst case, the active region is the last one to be examined, and the controller will give a solution after $(3N + 6) \cdot N_{CR}$ multiplications, $(3N + 3) \cdot N_{CR}$ sums and $N_{CR} - 1$ comparisons. In Table 1, the complexity of the control law is given in terms of storage demand and number of operations. As it turns out, even with probabilistic region checking or efficient representations of the state space, the computational demand of the problem remains considerable. This holds in particular since the control law has to be evaluated every time step T.

In Figure 26, an exemplary situation is displayed where the generated controller is optimizing the sampling rate R_1 and the transmission rate R_2 during 7 days. During the first 5 days, the controller achieves to optimize

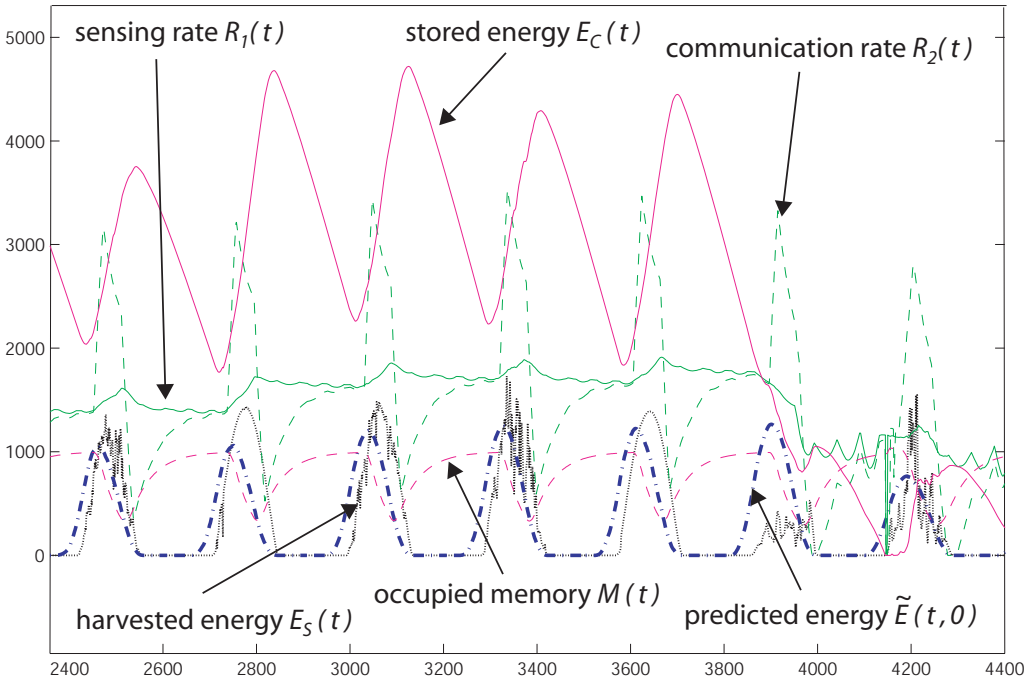


Fig. 26: Single controller for LP III, explicit solution via multiparametric programming, average energy prediction $\widetilde{E}(t,k)$.

the rate R_1 in spite of the unstable power supply $E_S(t)$. The transmission rate R_2 is oscillating around the sampling rate R_1. Since it is favourable to use energy when available, data is stored at night and transmitted during day. At this, the finite amount of storable data $M_{\max} = 1000$ is chosen large enough not to constrain the dynamic of the ratio $\frac{R_1}{R_2}$. Only if the memory is $M_{\max} < 800$, we observed that the parameter $M_{\max}$ begins to influence the overall system behaviour. Note that the parameters in Figure 26 are scaled appropriately to show them all in one diagram. The absolute values, however, are not necessary those displayed on the y-axis in Figure 26.

However, at the end of the displayed time period, the controller is forced to suspend the sampling of data and achieves the worst objective value possible: Shortly before $t = 4200$, the sampling rate R_1 is set to 0 for approximately 20 minutes. One of the main reasons for this breakdown is the averaging energy predictor $\widetilde{E}$ which – per definition – is unable to foresee extreme situations. As illustrated in Figure 26, an overestimation of the actually incoming energy E_S may deplete the stored energy quickly, resulting in an unpredicted performance degradation. From our experiments we know that the controller computed for LP III is unable to avoid these kind of breakdowns with the chosen predictor $\widetilde{E}$, independent of the length of the prediction horizon H and the number of intervals N.

3.6 Hierarchical System Model

3.6.1 Design Principles

From the examples presented in the last section, we identify unexpected energy depletion as a major challenge for environmentally powered systems. In this section, we present a hierarchical system design whose primary purpose is to prevent the system from running out of energy.

The main idea is to subdivide the controller for the problem formulated in (3.3)-(3.10) into two subcontrollers, each with its dedicated energy prediction algorithm. The parameters of the prediction horizons are denoted L_1, N_1 and H_1 for the upper control layer, and L_2, N_2 and H_2 for the lower control layer. According to Figure 27, subcontroller 1 receives estimates of the daily energy in order to ensure longterm sustainability of the energy harvesting system. Therefore, we assume a prediction interval to be 1 day, i.e. $L_1 \cdot T = 24h$ and the total prediction horizon should be chosen in the order of several weeks, e.g. $H_1 \cdot T = 30days$. At the interface of both control layers, we define the energy allowance $E_D(t)$ as the sum of energies which shall be used by the system at the next day.

By calculating $E_D(t)$, the upper layer determines an energy budget which serves as input to the lower control layer. Receiving hourly estimates with a prediction horizon of $H_2 \cdot T = 24h$, subcontroller 2 decides how to set the controlled parameters $\mathbf{R}(t)$ during the next day. At this, the efficiency of the energy utilisation is optimized by e.g. exploiting the sunlight directly when available.

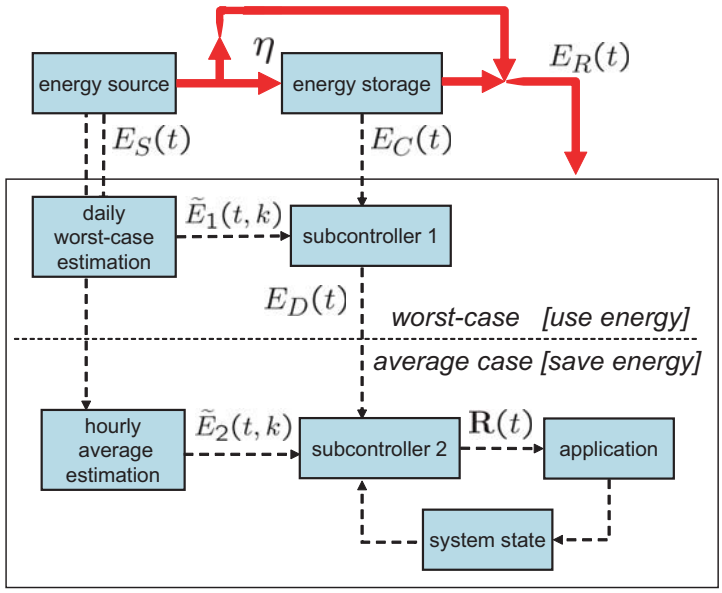


Fig. 27: Illustration of the hierarchical control model.

Algorithm 3 Daily worst-case energy prediction.

Input: $t, E_S(t), N_1, L_1$
Output: $\widetilde{E}_1(t,k) \; \forall \; 1 \leq k \leq N_1$
if $t == 0$ **then**
$\epsilon_s^l(\Delta) = \infty \quad \forall \; 1 \leq \Delta \leq N_1$
if $t \bmod L_1 == 0$ **then**

$$E_{S,d}(d) = \sum_{i=(d-1)\cdot L_1}^{d\cdot(L_1-1)} E_S(t-i) \qquad \forall \; 1 \leq d \leq N_1 \quad \textbf{(S1)}$$

$$\widetilde{\epsilon}_s(\Delta) = \sum_{i=1}^{\Delta} E_{S,d}(i) \qquad \forall \; 0 \leq \Delta \leq N_1 \quad \textbf{(S2)}$$

$$\epsilon_s^l(\Delta) = \epsilon_s^l(\Delta) + \Delta \cdot \gamma \qquad \forall \; 1 \leq \Delta \leq N_1 \quad \textbf{(S3)}$$

$$\epsilon_s^l(\Delta) \; = \min\left[\epsilon_s^l(\Delta), \widetilde{\epsilon}_s(\Delta)\right] \qquad \forall \; 1 \leq \Delta \leq N_1 \quad \textbf{(S3)}$$

$$\widetilde{E}_1(t,k) = \max_{0 \leq l \leq N_1 - k} \left\{\epsilon_s^l(k+l) - \widetilde{\epsilon}_s(l)\right\} \qquad \forall k \geq 1 \quad \textbf{(S4)}$$

The advantages of the hierarchical control model can be summarized as follows (see also experimental results in Section 3.6.2): First, by virtue of its worst-case design, the upper control layer avoids depletion of the energy storage and increases the robustness of the overall system. Second, the control formulation renders manual tuning of the final state constraint unnecessary (cp. equation (3.10)), resulting in an automatical optimization and stabilization of the system behaviour. Third, by decoupling the control problem into two subproblems, the complexity of the online controller is reduced significantly. In the following, the two control layers will be described in detail.

3.6.1.1 Longterm Performance Optimization

The optimization objective of subcontroller 1 is to maximize the minimal available energy $E_D(t)$ per day. In other words, we would like to guarantee sustainable operation in a worst-case sense, similar as in objective (3.9). For this reason, an energy prediction for a worst-case scenario is required. Unlike most of the energy estimation algorithms presented in related work, we would like to determine the *minimal accumulated energy* for the next days, which we denote as $\widetilde{E}_1(t,k)$. For prediction intervals of size $L_1 \cdot T = 24h$, the algorithm is given in Algorithm 3. Note that contrary to the definition of $\widetilde{E}(t,k)$ in Section 3.4.3 $\widetilde{E}_1(t,k)$ denotes the *cumulated* energy for the next k intervals.

During the day, only a counter is running which accumulates the harvested energy of the current day. Once per day, Algorithm 1 is executed and new energy estimates for the next N_1 days are computed. The har-

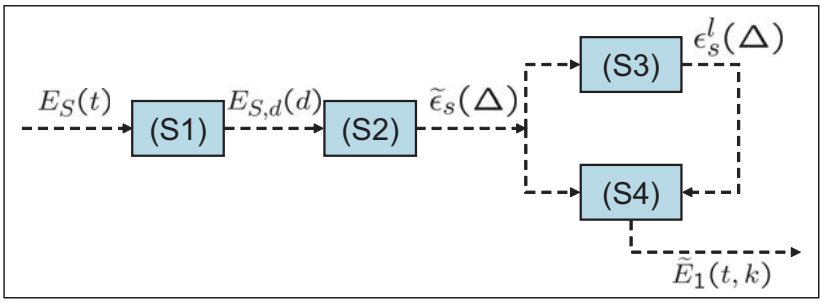


Fig. 28: Visualization of the daily energy prediction in Algorithm 3.

vested energies of the last N_1 days are stored in the vector $E_{S,d}(d)$. In a first step **(S1)**, vector $E_{S,d}(d)$ is updated by shifting all entries by one day and storing the harvested energy $E_{S,d}(1)$ of the current day. In step **(S2)**, the daily energies $E_{S,d}(d)$ are summed up to obtain the energies $\widetilde{\epsilon}_s(\Delta)$ for time periods Δ of several days. As an internal state, the estimator stores the minimum of all vectors $\widetilde{\epsilon}_s(\Delta)$ observed so far in a variable denoted $\epsilon_s^l(\Delta)$. Next, Step **(S3)** checks whether the energies $\widetilde{\epsilon}_s(\Delta)$ harvested in the past Δ days are smaller then the minimum energies $\epsilon_s^l(\Delta)$ observed so far for the respective time interval Δ. In addition, a constant γ is added to the minimum $\epsilon_s^l(\Delta)$ to account for aging of old measurements. In doing so, the algorithm avoids pessimistic predictions based on low values of $\epsilon_s^l(\Delta)$ measured long time ago. Finally, step **(S4)** improves the estimates $\epsilon_s^l(\Delta)$ if intervals with little harvested energy $\widetilde{\epsilon}_s(\Delta)$ have occurred recently.

For a small number of days, the worst-case energy prediction algorithm gives pessimistic predictions $\widetilde{E}_1(t,k)$, which is reasonable if continuous operation of a sensor node has to be guaranteed. During this time, an underestimation of the actually scavenged energy may happen. In a longterm perspective of a few weeks, however, the worst-case energy prediction algorithm begins to predict the average incoming energy just as conventional estimation algorithms.

The linear optimization problem underlying subcontroller 1 is given below. With the help of the worst case energy predictions $\widetilde{E}_1(t,k)$ it is now possible to maximize the smallest energy $E_D(t+k\cdot L_1)$ for all $0 \le k < N_1$ in the prediction horizon. However, only the first energy $E_D(t)$ is passed to subcontroller 2 and will be used during the next day.

maximize $J = \mu$ subject to:	(subcontroller 1)
$E_D(t+k\cdot L_1) \ge \mu$	$\forall 0 \le k < N_1$
$E_C(t+k\cdot L_1) = E_C(t) + \widetilde{E}_1(t,k) - \sum_{j=1}^{k}(E_D(t+(j-1)\cdot L_1))$	$\forall 1 \le k \le N_1$
$0 \le E_C(t+k\cdot L_1) \le E_{\max}$	$\forall 1 \le k \le N_1$

3.6.1.2 Shortterm Power Saving

As already indicated, the prediction horizon of the lower control layer spans $H_2 \cdot T = 24h$. For a general control problem and a prediction $\widetilde{E_2}(t,k)$, the final state constraint of the lower control layer can be set to $\alpha(t) = E_D - \sum_{i=0}^{N_2-1} \widetilde{E_2}(t,i)$ in order to force the system to spend energy E_D during the next day. Hence, the final state constraint $\alpha(t)$ is calculated automatically by the upper layer.

In this chapter, we focus on a concrete application (Example III) where energy losses due to storage efficiency η shall be minimized. Nevertheless, our methods are also applicable to other power saving techniques, like e.g. dynamic voltage scaling (DVS), energy-efficient packet scheduling for wireless links or applications with data compression.

For the optimization problem with non-ideal energy storage in LP III, we have to ensure that the sensor node uses no more than energy E_D during the next day. The only degree of freedom to be exploited by subcontroller 2 is *when* to transmit the images with rate $R_2(t)$ in order to *save* as much energy as possible. Thus, we want to maximize the energy $E_C(t+L_2 \cdot N_2)$ at the end of the day. For this setup, we can formulate the linear program to be solved by subcontroller 2 as follows:

maximize $J = E_C(t + L_2 \cdot N_2)$ subject to:	(subcontroller 2)
$R_1 = \frac{\eta \cdot E_D(t)}{N_2 \cdot (e_1 + e_2)}$	
$\sum_{j=0}^{N_2-1} R_2(t + j \cdot L_2) = \frac{\eta \cdot E_D(t)}{e_1 + e_2}$	
$R_2(t + k \cdot L_2) \geq 0$	$\forall 0 \leq k < N_2$
$E_C(t + k \cdot L_2) = E_C(t) + \sum_{j=0}^{k-1} \widetilde{E_2}(t,j) - e_1 \cdot L_2 \cdot k \cdot R_1 -$	
$e_2 \cdot L_2 \cdot \sum_{j=0}^{k-1} R_2(t + j \cdot L_2) - (1-\eta) \sum_{j=0}^{k-1} \lambda(j)$	$\forall 1 \leq k \leq N_2$
$\lambda(j) \geq \widetilde{E_2}(t,j) - e_1 \cdot L_2 \cdot R_1 - e_2 \cdot L_2 \cdot R_2(t + j \cdot L_2) \geq 0$	$\forall 0 \leq j < N_2$
$M(t + k \cdot L_2) = M(t) + \sum_{j=0}^{k-1} (R_1(t + j \cdot L_2) - R_2(t + j \cdot L_2))$	$\forall 1 \leq k \leq N_2$
$0 \leq M(t + k \cdot L_2) \leq M_{\max}$	$\forall 1 \leq k \leq N_2$

To exploit the typical profile of solar energy during a day, an energy predictor $\widetilde{E_2}(t,k)$ is needed which predicts the most probable energy values for the next hours. In contrast to $\widetilde{E_1}(t,k)$, the hourly estimation $\widetilde{E_2}(t,k)$ should keep a history of harvested energies and use a representative, *average* value as predictor.

3.6.2 Optimization with Non-Ideal Energy Storage (cont. Example III)

Next, we subdivide the control problem in LP III into two hierarchically structured subproblems as described in the previous section. Using multiparametric linear programming, we compute the explicit solutions for subcontroller 1 and subcontroller 2. Concerning the upper control layer, we chose a prediction horizon of 30 days, where each interval spans one day. We set the aging of old data $\gamma = 100$ for the worst case prediction algorithm $\widetilde{E}_1$. To avoid unnecessary control overhead, subcontroller 1 is not activated every $T = 5$ minutes, but only once per day. As in the previous experiment with a single controller, the constraint $E_{\max}$ on the maximum storable energy is omitted for subcontroller 1. For the lower control layer, the prediction algorithm in [MTBB07] has been used, again with the same parameters. As for the single controller, the prediction horizon of the lower control layer spans one day, with 6 intervals of 4 hours length.

Tab. 1: Complexity reduction due to hierarchical control design

control design	N	$H \cdot T$	activation frequency	N_{CR}	storage (real numbers)	ops (worst case)
single controller	6	$1day$	$T = 5$ min	1049	28323	52449
hierarchical, subcontroller 1	$N_1 = 30$	$H_1 \cdot T = 30days$	$L_1 \cdot T = 1day$	30	1920	3689
subcontroller 2	$N_2 = 6$	$H_2 \cdot T = 1day$	$T = 5$ min	161	2898	4829

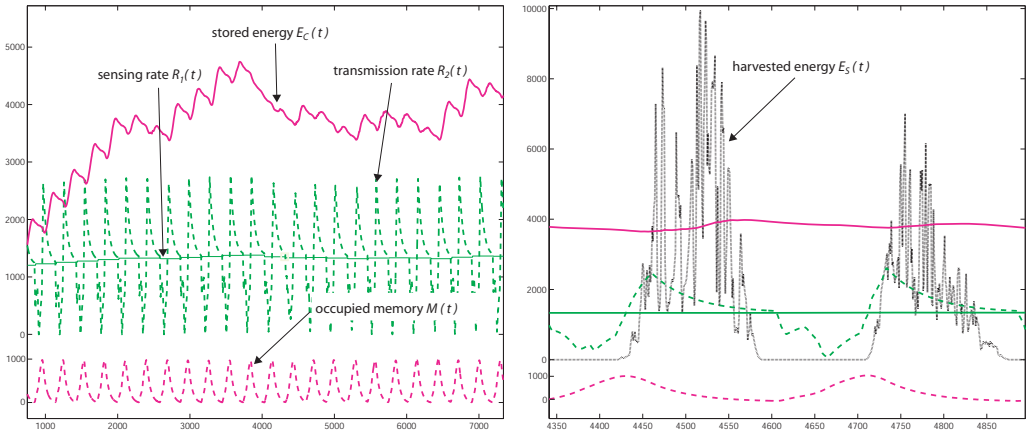


Fig. 29: Hierarchical system, multiparametric programming, worst case prediction $\widetilde{E}_1(t,k)$, average prediction $\widetilde{E}_2(t,k)$.

Table 1 summarizes the design parameters and the implementation overheads of both control designs. In terms of storage demand, the hierarchical approach yields a substantial reduction of the storage requirement of 83,0%. Since subcontroller 1 is only activated once per day, one has to divide its computation demand by L_1 to obtain the number of operations per interval T. More precisely, assuming that *all* regions have to be tested before evaluating the control law, 3689 operations have to be executed at every 288th multiply of the basic time interval T. Hence, only 12.8 operations plus 4829 operations of subcontroller 2 have to be executed at every time step T. This yields a reduction of the worst case computation demand of 91,0% compared to a single controller.

In dependence on how the control laws are implemented in practice, the number of necessary operations will be smaller than the worst case values in Table I. As it has been shown in [MTBB07], the *average* number of region tests can be reduced by exploiting the fact, that some regions are more frequently activated then others. For the probabilistic region checking method proposed in [MTBB07], we experienced that the average number of regions tests is proportional to the total number of regions N_{CR} (and hence also to the number of operations in the worst case). Since the worst-case number of region tests is independent of the implementation strategy used, we opted for this metric to indicate the computational effort. Nevertheless, our methods simultaneously reduce the average as well as the worst-case number of operations.

Clearly, the daily energy estimation can be seen as an additional overhead of the hierarchical approach. However, apart from a counter which is accumulating the incoming energy E_S, all calculations are performed only once per day. Thus, we assume the contribution of the second predictor to the overall overhead to be negligible.

Figure 29 depicts the performance of the hierarchical control approach. The simulation result is plotted for 22 days, covering also the 7 days of simulation period of the previous experiment. This time, the controllers successfully stabilize the sampling rate at $r_1 \approx 1300$. Depending on the season, we found that rate r_1 may slightly increase (in summer) or decrease (during winter). In any case, subcontroller 1 prevents the system to run out of energy and simultaneously maximizes the amount of energy E_D per day. For a small number of days, like e.g. 1-4 days, the predictor $\widetilde{E}_1$ is generating rather pessimistic estimates, accounting for a couple of cloudy days. For several days or weeks, the accumulated energy is very likely converging towards an average value, like the estimate $\widetilde{E}_2$ used for subcontroller 2. Exploiting this mechanism, subcontroller 1 autonomously regulates the offset $\alpha(t)$ of the final state constraint and optimizes the longterm system behaviour.

On the other hand, to exploit the possibility to save power it is essential to work with the most probable energy estimates for the next hours, like the ones generated by $\widetilde{E}_2$. This is now possible since the stability of the system is guaranteed from the upper control layer. In addition, prediction mistakes of $\widetilde{E}_2$ do not jeopardize the stability of the system anymore. Hence, one can say that our approach also improves the robustness against prediction mistakes.

In the left diagram of Figure 29, all data displayed is scaled in the same way as in Figure 26 except of the stored energy E_C. In fact, the stored energy E_C in Figure 29 is divided by a factor of 6 in comparison to Figure 26. In other words, the stored energy is actually varying up to 30000 units instead of 5000. On sunny days, however, the accumulated daily energy may be more than 10000 energy units. In consideration of this fact, a sensor node running our hierarchical control approach would require an energy storage which is able to store solar energy of 3 days in advance. It becomes evident that this is the capacity of the energy storage which is necessary for stable, sustainable operation. Using our methods, one can simulate the system and dimension the capacity $E_{\max}$ of the battery in a pre-operational phase.

3.7 Approximate Control Design

In this section, we present an algorithm for approximative multiparametric linear programming. As a matter of fact, only a few such algorithms have been proposed in the literature so far (among them, e.g., the algorithms in [Fil04] and [JBM07]). The algorithm is presented in Section 3.7.1 and in Section 3.7.2 we evaluate the algorithm for the exemplary sensor

application (Example III) from the previous sections.

3.7.1 An Approximative MP Linear Programming Algorithm

It is well known in the control community that the size of the explicit solution obtained by multiparametric linear programming grows quickly if the complexity of the control problem increases. In deed, already a moderate number of control variables and parameters may result in a huge number N_{CR} of critical regions. At this, adjacent regions are often characterized by almost identical control laws. This circumstance, however, has neglectable impact on the resulting control profile in many cases. Rather, numerous regions N_{CR} entail an high overhead in terms of storage requirement, running time as well as energy consumption.

Beyond these general short-comings, one could also argue in favour of an approximate control approach due to the stochastic nature of the harvested energy. The future energy provided by photovoltaic cells can only be estimated. The predicted energy values which are available in practice may be too inexact and too unreliable to provide the basis for "exact" procedures like linear programming. From this point of view, a precise calculation of the optimal control values turns out to be worthless if significant prediction errors occur.

In this section, we present a new algorithm for approximative multiparametric linear programming. The basic idea was

- to take a large number of samples $\mathbf{X}_i$ of the state space of $\mathbf{X}$ (compare equation (3.11)),
- to solve a linear program for each sample $\mathbf{X}_i$ to obtain the respective optimal solution $\mathbf{U}_i^*$,
- to find a (preferably simple) fitting function $\hat{\mathbf{U}}^*(\mathbf{X})$ for the multidimensional data $(\mathbf{X}_i, \mathbf{U}_i^*)$,
- and finally to use $\hat{\mathbf{U}}^*(\mathbf{X})$ (which has been calculated offline) as approximation for $\mathbf{U}^*(\mathbf{X})$ in the online case.

At first, a random number generator is used to generate the samples $\mathbf{X}_i$, $1 \leq i \leq N_S$, where N_S denotes the total number of samples. We used independent, uniformly distributed random values as samples for the single elements of $\mathbf{X}$. For example, values of the stored energy E_C have been chosen according to a uniform distribution

$$f_{E_C}(E_C) = \begin{cases} \frac{1}{E_{\max}} & \text{if } 0 < E_C < E_{\max} \\ 0 & \text{else} \end{cases}, \tag{3.14}$$

with the probability density function $f_{E_C}(E_C)$ and the maximum storable energy $E_{\max}$. In the same way, we sampled M and $\widetilde{E}$ using the respective upper bounds on the available memory as well as producible energy.

As fitting algorithm, we opted for the algorithm proposed in [MB06]. This algorithm attempts to fit data samples to a set of convex, piece-wise linear candidate functions. In order to provide a good fit, the algorithm requires the function which generates the samples to have a convex curvature. The optimal control rates $\mathbf{U}^*(\mathbf{X})$, however, are not necessary convex over the state space $\mathbf{X}$. Hence, a direct fitting of the control rates is not possible using the algorithm in [MB06].

According to the following theorem, the optimal objective value $J^*(\mathbf{X})$ exhibits the wished convexity property.

Thm. 7: (cf. page 180 in [Gal95]) *The function $J^*(\mathbf{X})$ is continuous, piecewise affine and convex over $\mathbf{X}$.*

As the optimal control vector $\mathbf{U}^*(\mathbf{X})$, $J^*(\mathbf{X})$ can be computed exactly using multiparametric programming as

$$J^*(\mathbf{X}) = \mathbf{T}_j\mathbf{X} + \mathbf{V}_j \quad \text{if } \mathbf{H}_j\mathbf{X} \leq \mathbf{K}_j, j = 1, \ldots, N_{CR} \tag{3.15}$$

where $\mathbf{T}_j$ and $\mathbf{V}_j$ are matrices of appropriate dimensions. It is important to note that $J^*(\mathbf{X})$ is piecewise linear over the *same* polyhedral partition $\mathbf{H}_j\mathbf{X} \leq \mathbf{K}_j$ as the optimal control $\mathbf{U}^*(\mathbf{X})$ (cf. equation (3.13)).

For each sample $\mathbf{X}_i$, we now solve a linear program and determine the optimal control vector $\mathbf{U}^*(\mathbf{X}_i)$ as well as the optimal objective value $J^*(\mathbf{X}_i)$. This can be done using common simplex-based or interior-point solvers. Next, we implement the heuristic algorithm in [MB06] to fit the objective $J^*(\mathbf{X_i})$, i.e to solve the least square fitting problem

$$\text{minimize} \quad \sum_{i=1}^{N_S} \left(\max_{j=1,\ldots,\hat{N}_{CR}} (\hat{\mathbf{T}}_j^T \cdot \mathbf{X}_i + \hat{\mathbf{V}}_j) - J^*(\mathbf{X}_i) \right)^2 \tag{3.16}$$

Like that, we obtain the approximated objective function $\hat{J}^*(\mathbf{X})$ in the so-called "max-affine" form:

$$\hat{J}^*(\mathbf{X}) = \max_{j=1,\ldots,\hat{N}_{CR}} \{\hat{\mathbf{T}}_j^T \cdot \mathbf{X} + \hat{\mathbf{V}}_j\} \tag{3.17}$$

The piecewise affine convex function (3.17) can be recast easily in the following equivalent form which explicitly defines the polyhedral partition $\hat{\mathbf{H}}_j\mathbf{X} \leq \hat{\mathbf{K}}_j$ (see also [Sch87]).

$$\hat{J}^*(\mathbf{X}) = \hat{\mathbf{T}}_j\mathbf{X} + \hat{\mathbf{V}}_j \quad \text{if } \hat{\mathbf{H}}_j\mathbf{X} \leq \hat{\mathbf{K}}_j, j = 1, \ldots, \hat{N}_{CR} \tag{3.18}$$

Next, we group the samples $\mathbf{X}_i$ according to the region j they belong to. For each region j, we perform a simple least square fitting of the respective samples to compute the coefficients $\hat{\mathbf{A}}_j$ and $\hat{\mathbf{B}}_j$ of the approximated control rates $\hat{\mathbf{U}}^*$. As a result, we have derived an explicit form for the control rates $\hat{\mathbf{U}}^*(\mathbf{X})$ as a function of the current state $\mathbf{X}$:

$$\hat{\mathbf{U}}^*(\mathbf{X}) = \hat{\mathbf{A}}_j\mathbf{X} + \hat{\mathbf{B}}_j \quad \text{if } \hat{\mathbf{H}}_j\mathbf{X} \leq \hat{\mathbf{K}}_j, j = 1, \ldots, \hat{N}_{CR} \tag{3.19}$$

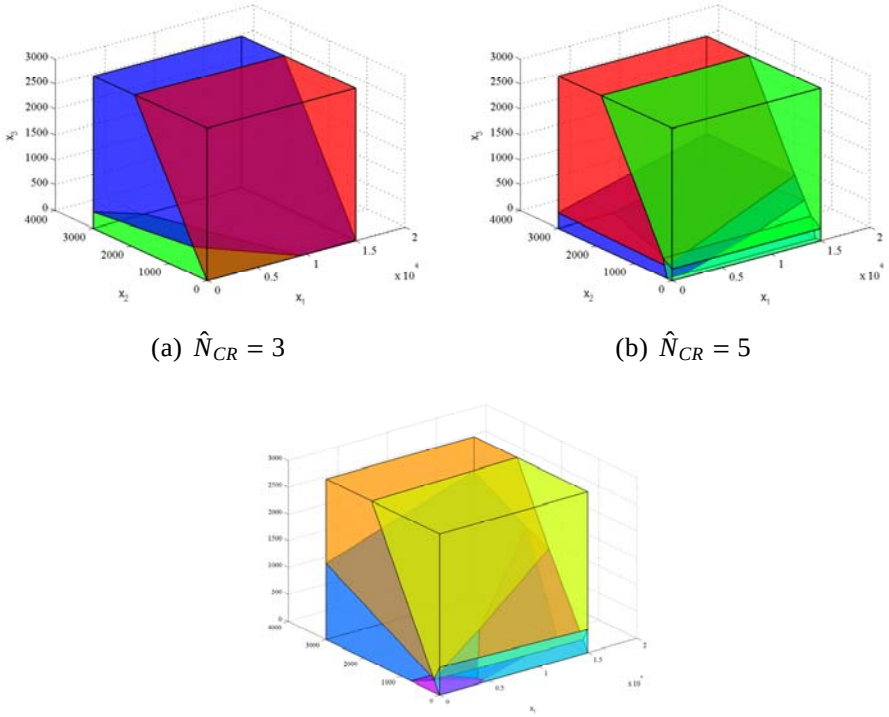
Everything done so far has to be done offline. The approximated control law in (3.19) can now be used in an online controller instead of the exact solution in (3.13). Since the convex fitting algorithm in [MB06] allows to tune the number $\hat{N}_{CR}$ of critical regions, one may chose a smaller number of regions $\hat{N}_{CR} < N_{CR}$ to reduce the complexity of the control problem. The fitting algorithm then attempts to create a smaller polyhedral partition which minimizes the least square error of the objective value function. In comparison to the optimal, multiparametric solution the fitting algorithm seems to merge smaller regions and reshape the geometry of the partition, as we will see in the next section. Albeit no performance guarantees of the heuristic algorithm are given in [MB06], it turns out that the algorithm performs well and produces suitable approximations, both in [MB06] and also in our experiments.

3.7.2 Optimization with Non-Ideal Energy Storage (cont. Example III)

In Section3.6, both controllers involved in the hierarchical control design have been computed using the optimal partitions of the parameter space obtained by multiparametric programming. In this section, we derive and test suboptimal, approximate control laws using the algorithm presented in previous section.

The key operation in finding an approximate solution for a control problem is the fitting of the (convex) objective function $\hat{J}^*(\mathbf{X}) = \max_{j=1,\ldots,\hat{N}_{CR}} \{\hat{\mathbf{T}}_j^T \cdot \mathbf{X} + \hat{\mathbf{V}}_j\}$. At this, the algorithm in [MB06] alternates between partitioning the data in new regions j and carrying out least-squares fits to update the coefficients $\hat{\mathbf{T}}_j$ and $\hat{\mathbf{V}}_j$. Starting with an initial number $\hat{N}_{CR,init}$, critical regions may be merged at every iteration, leading to a reduced number of regions. The algorithm converges if a partition remains unchanged after an iteration or some maximum number of iterations is reached. The final number of partitions $\hat{N}_{CR}$ can be influenced by appropriate choice of the initial parameters.

For subcontroller 1, we sampled the state space $\mathbf{X}_1 = \left(E_C(t), \widetilde{E}_1(t,0), \ldots, \widetilde{E}_1(t,29)\right)$ using $N_S = 1000$ samples. For subcontoller



(a) $\hat{N}_{CR} = 3$ (b) $\hat{N}_{CR} = 5$

(c) $\hat{N}_{CR} = 10$

Fig. 30: Illustration of approximated polyhedral partitions for subcontroller 2. Cut through $\widetilde{E}_2(t,1) = 1200$, $\widetilde{E}_2(t,2) = 1000$, $\widetilde{E}_2(t,3) = 0$, $\widetilde{E}_2(t,4) = 100$ and $\widetilde{E}_2(t,5) = 500$.

2, $N_S = 2000$ samples have been taken from the state space $\mathbf{X}_2 = \left(E_D(t), M(t), \widetilde{E}_2(t,0), \ldots, \widetilde{E}_2(t,5)\right)$. In Figure 30, some exemplary partitions of $\mathbf{X}_2$ are displayed. In dependence of the choice of the 3-dimensional cut, not all $\hat{N}_{CR}$ partitions may be visible in diagrams (a)-(c).

Tab. 2: Comparison of multiparametric and approximate-mp control design

control design	$\max_t \left\lvert \frac{\hat{R}_1(t)}{R_1(t)} - 1 \right\rvert$	$\max_t \left\lvert \frac{\hat{E}_C(t)}{E_C(t)} - 1 \right\rvert$	η_{avg}	N_{CR} (or $\hat{N}_{CR}$)	storage (real numbers)	ops (worst case)
MP, subcontroller 1	0%	0%	93.00%	30	1920	3689
subcontroller 2				161	2898	4829
approx., subcontroller 1	1.52%	11.57%	93.75%	4	256	308
subcontroller 2				4	108	69
approx., subcontroller 1	0.82%	5.47%	92.97%	4	256	308
subcontroller 2				9	243	173

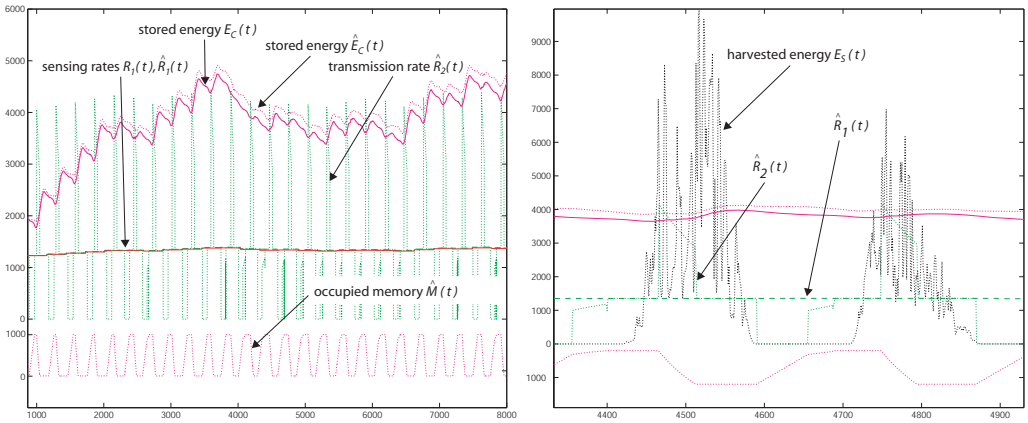


Fig. 31: Hierarchical system, approximate multiparametric programming, $\hat{N}_{CR} = 4$ for both subcontroller 1 and subcontroller 2.

We denote $\hat{R}_1$, $\hat{R}_2$, $\hat{E}_C$ and $\hat{M}$ the rate and state variables obtained if approximate control laws are applied. Let us also define the rectifier function $[\Delta E]^+$ as follows:

$$[\Delta E]^+ = \begin{cases} \Delta E & \text{if } \Delta E \geq 0 \\ 0 & \text{if } \Delta E < 0 \end{cases} \tag{3.24}$$

Using this notation, we denote the average efficiency of the energy utilisation

$$\eta_{avg} = 1 - \frac{(1-\eta) \cdot \sum_t [E_S(t) - e_1 R_1(t) - e_2 R_2(t)]^+}{\sum_t E_S(t)} \tag{3.25}$$

Figure 31 displays the evaluation of an approximate control law with $\hat{N}_{CR} = 4$ for both subcontroller 1 and subcontroller 2. The actual optimization objective of regulating the sensing rate R_1 is met almost as well as the exact solution. The maximal derivation of $\hat{R}_1$ from R_1 is 1.52%, making both lines indistinguishable in the left diagram of Figure 31. Obviously, the approximated algorithm manages to save even slightly more energy then its exact counterpart. As displayed in the right diagram of Figure 31, the transmission rate $\hat{R}_2$ is adjusted to much higher values during day and consequently set to 0 at night. Apparently, this strategy even yields a gain of 0.75% of the average efficiency η_{avg}. The stored energy $\hat{E}_C$ is varying up to 11.57% from E_C. However, the peak of $\hat{E}_C$ is just 4.03% above the one of E_C. That is, the capacity of the energy storage is required to be approximately 5% higher if the system is controlled by an approximated algorithm.

Showing a comparable performance during runtime, the main advantage of the approximation becomes obvious considering the complexity of

the control laws. According to Table 2, the storage demand is significantly reduced by 92.44% compared to the optimal solution. In terms of worst-case computation demand, the reduction even amounts 98.55%. Table 2 also outlines the results for a second low-complexity approximation. It exhibits a slightly lower efficiency η_{avg}. On the other hand the second approximation closely matches the optimal case in terms of control rate R_1.

In summary, we can state that the approximate solutions to the multiparametric control problem do not necessarily entail a degraded performance in terms of the controlled parameters. In deed, in most cases we could find simple but useful approximations for the underlying control problems. Besides the significant complexity reduction, there is another advantage of the proposed approximation technique that should be mentioned. For highly complex control problems consisting of several thousands of regions N_{CR}, conventional solvers may be unable to find the optimal polyhedral partition. We experienced that sometimes even no solution can be found at all. In other examples, the mp-LP solvers we used could not generate the optimal partition with the minimal number of regions N_{CR}. Rather, a high number of overlapping regions is constructed which cannot be removed afterwards. Here, our method turned out to be helpful to find a reasonable solution at all.

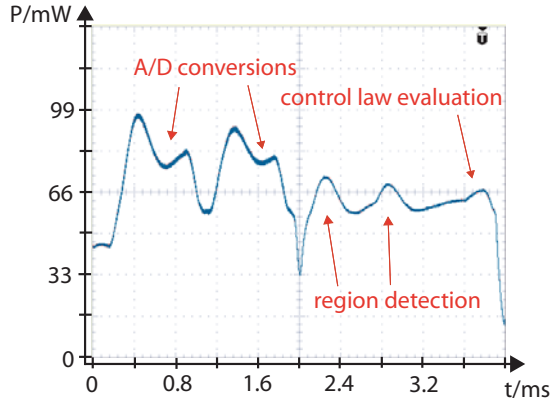
3.8 Hardware Implementation Issues

In this section, we demonstrate how the proposed control laws can be implemented efficiently and evaluate the implementation overhead on a BTnode [BDH+04]. For this purpose, we show measurements of two implementations of exemplary controllers which has been introduced and discussed in the last sections.

3.8.1 Average Computation Demand

In general, the identification of the active region j dominates the linear function evaluation of a control law (3.13) in terms of time and energy consumption. In the worst case, for all N_{CR} regions a matrix multiplication has to be performed in order to identify the active region j at time t. However, the identification of the active region j can be simplified due to the following facts: First, the matrices $\mathbf{H}_j$ are sparse which reduces the number of necessary multiplications and additions significantly. Second, usually only a subset of all N_{CR} is activated in practice and some of the regions in this subset are activated more frequently than others.

The second observation can be exploited by starting the search always



(a) Power consumption of a representative control law evaluation.

(b) The BTnode sensor node.

Fig. 32: Running the controller for the memory optimization problem (LP II) on a BTnode.

with the region with the highest statistical occurrence, continue with the second highest, and so on. To this end, an algorithm should maintain a list of regions j ordered by their frequencies which is updated every time step t.

For the memory optimization problem in LP II in Section 3.5.3, we found that on average $\approx 40\%$ of the entries of the matrices $\mathbf{H}_j$ are different from zero. Moreover, only 7 of 39 regions were used during the whole simulation period. Using probabilistic region testing as described above, the critical term of the form $\mathbf{H}_j \cdot \mathbf{X} \leq \mathbf{K}_j$ is only evaluated ≈ 1.45 times at time t, taking the average over the whole simulation period. At this, we enforced the list of regions to be pre-ordered in a worst case sense, i.e., every time a region is activated for the first time, all other regions are checked first.

Figure 32(a) displays the measured power consumption of a BTnode [BDH+04]. At first, the current energy level $E_C(t)$ of the battery and the scavenged energy $E_S(t)$ are determined via two A/D-conversions, which mark the two major peaks in plot 32(a). Focusing on the overhead of the proposed controller, we omit predicting the future energies $\widetilde{E}(t, i)$. Instead, an average situation is displayed where the region with the second highest frequency is the active one. Subsequently, the optimal control output for this region is calculated. It becomes evident, that the computations leading to the actual control actions take as long as the two A/D-conversions ($\approx 2ms$). Hence, these measurements demonstrate how the simple, but efficient implementation of the proposed controller is applicable to sensor nodes, involving only marginal computation overhead.

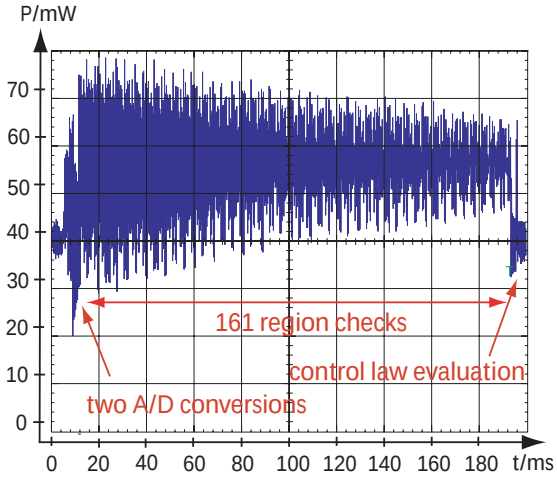


Fig. 33: Worst-case power consumption of subcontroller 2 on a BTnode.

3.8.2 Worst-Case Computation Demand and Storage Demand

For some applications, it is also important to consider the worst-case computation demand. Here, worst-case refers to the situation where the currently active region is the *last* one to be checked. For example, for the hierarchical control design presented in Section 3.6, we are interested in the worst-case computation time and energy consumption for a practical implementation.

Since the daily energy estimation and subcontroller 1 are only activated once per day, their influence on the computation demand is negligible. In our test implementation on a BTnode (see Figure 33), we measured a worst-case computation time < 190 ms for subcontroller 2. The evaluation of this control law is only done every T time units, which may be every 5 or 10 minutes, depending on the characteristics of the energy source. Compared to the energy consumption of the rates $\mathbf{R}$ of the actual application during this interval T, the evaluation of the control laws contributes a negligible control overhead.

Using multiparametric programming techniques instead of solving the optimization problem online basically means a shift from *computation* to *storage*. Thus, one has to ensure that the storage of the control law does not exceed the limits of the embedded system. From the example applications presented in this chapter, all control laws could be implemented on a BTnode. For example, the controller for LP III presented in Section 3.5.4 requires the storage of the $N_{CR} = 1049$ regions and corresponding control laws. In terms of physical memory, this amounts to 55,3 Kbyte using a 16 bit integer representation per coefficient. Using the hierarchical control approach as presented in Section 3.6, we could reduce the storage demand to 9,4 Kbyte. This reduction makes the control law implementable

on commonly used sensor nodes like the Tmote Sky [Cor06], which features 48 Kbyte Flash ROM or the BTnode [BDH+04], which exhibits 128 Kbyte Flash ROM. In summary, using the tools and methods described in this chapter, we believe that controllers for a large class of optimization problems can be implemented on commonly used sensor network platforms. To this end, both the presented hierarchical control design and the approximation algorithm may be used.

3.8.3 Realistic Modelling of the Energy Storage

In Section 3.4, the energy storage model has been presented which allows the solar cells to directly power the embedded system and to charge the energy storage with efficiency η. In the following, we will shortly elaborate on how this model fits a realistic hardware implementation.

Let us discuss the simplest possible configuration consisting of a solar cell, a battery as well as a sensor node. As a matter of fact, using a parallel connection of the three components, it can be shown that the model presented in Section 3.4 fits perfectly. The voltage over the three components is equal and – according to Kirchhoff's law – the sum of currents at the junction is 0. Thus, in terms of power flow, the power delivered by the solar cell splits into one part powering the sensor node and the rest, which is charging the battery with efficiency η.

For more sophisticated charging circuits as the ones presented in [SC06, PC06], it is less obvious why the power flow shows this behaviour. Furthermore, for energy storage devices whose efficiencies η are not constant over time, a more detailed modelling may become necessary. To this end, the efficiency η can be modeled by a piece-wise linear function over, e.g., the stored energy E_C. For instance, we also performed experiments with low efficiencies η when the stored energy is close to 0 and $E_{\max}$ and a higher efficiency η for values in between. Like that, a more realistic modelling of the energy storage becomes possible, which, of course, increases the complexity of the optimization problem.

3.9 Chapter Summary

In this chapter, we present a framework of tools and methods to optimize the performance of energy harvesting systems using multiparametric programming techniques. On the one hand, we have successfully designed and evaluated an hierarchical control design which is tailored to the requirements of solar-powered sensor nodes. Due to its worst case design, the upper control layer prevents the sensor node from running out of energy. On the lower control layer, we showed how power

saving techniques can work more efficiently if stability of the operation is guaranteed by the layer above. Simultaneously, the reformulation of the control problem into two subproblems yields a significant reduction in online complexity. On the other hand, we propose a new algorithm for approximative multiparametric programming. The resulting control laws are rough approximations of the optimal solution and reduce the involved online overhead substantially. An experimental setup reveals that the achieved performance may be comparable to the optimal solution. All methods are supported by extensive simulations results which are based on longterm measurements of solar energy. Measurements of our controllers running on a sensor node together with a detailed analysis of the implementation overhead show that our algorithms can be implemented efficiently.

4

Reward Maximization

In Chapter 2, it was pointed out that greedy scheduling disciplines are not suitable if tasks on a uniprocessor are processed using time as well as regenerative energy. An optimal scheduling algorithm which tries to avoid deadline violations was presented. In contrast, the application discussed in this chapter requires sequential execution of periodic tasks. Task preemption is not allowed, and also not necessary. Instead, we try to optimize the overall reward of the application. Similar as in Chapter 3, parameters of the applications are adapted in a long-term perspective.

In Chapter 3, we show that many optimization problems arising in energy harvesting systems can be modeled by the class of linear programs. A multiparametric linear programming approach is presented which solves optimization problems offline and stores look-up tables for online usage. As objective, (piecewise) linear functions are possible whereas this chapter focuses on concave objective functions. Moreover, this chapter presents polynomial time algorithms for a specific system dynamic which solely require the storage of a few internal variables.

4.1 Introduction

Energy management has become vitally important to battery-operated embedded systems. In the past years, significant research effort has been dedicated to achieve efficient energy utilization. This holds in particular for systems adopting dynamic voltage scaling [YDS95, CK07a], dynamic power management [JPG04, CK07b], and micro-architectural techniques

for cache re-configuration [YL04]. Such methods tempt to increase the battery lifetime and maximize energy savings while still maintaining an acceptable service level for the user.

Instead of energy consumption minimization under performance constraints, in energy harvesting systems, the energy consumption of the system should depend on the energy harvested from the environment to maximize the performance. Typically, the harvested energy by, e.g., solar cells is not constant over time but arrives in bursts. In an outdoor environment, the energy generated by a photovoltaic cell normally follows a diurnal cycle with plenty of energy arriving at noon. At night, however, the system has to survive for several hours by solely consuming energy stored in the battery which has been harvested during the last days. There exist two major constraints which arise due to the burstiness of common energy sources: The harvested energy is temporarily low and the service must be lowered or suspended. (2) During bursts of incoming energy, the harvested energy exceeds the battery capacity and is wasted. Driven by solar energy, the main challenge for such a system is to optimize its performance while respecting the time-varying amount of energy. How to design and play out a given battery capacity becomes a key concern.

For battery-driven systems *without* energy harvesting possibilities, reward maximization has been studied extensively in the literature. At this, the energy is provided by a *finite* energy source, e.g., a battery. Typically, energy saving mechanisms like dynamic voltage scaling (DVS) are exploited to save energy and maximize the overall reward. For instance, Yun and Kim [YK04a, YK04b] study reward-based voltage scheduling for hard real-time systems. Furthermore, heuristic algorithms for scheduling multiple real-time tasks with different rewards have been presented by Rusu et al. [RMM03]. Recently, Chen et al. [CKY04, CK05] have proposed new approximation algorithms for energy-efficient scheduling on DVS platforms. In our work, we also adopt the notion of rewards to express different qualities of services and, as the objective, we try to maximize the overall reward. Our methods support DVS platforms, but work as well in the absence of DVS. The main difference is that we explore systems which are powered by an *infinite* environmental energy source, e.g., solar energy. We shad light on fundamental principles how to maximize the overall reward using a prediction of the energy generated in the future.

In general, the reward associated with a service increases with the amount of computation required to provide the service. Prominent models used in literature are the imprecise computation model [LLS+91] and the increasing reward with increasing service (IRIS) model [DKT96, SLC91]. For numerous practical applications, such as image and speech processing, time-dependent planning, and multimedia applications, the

reward function is modeled as a concave function of the amount of computation [AMMA99].

In this chapter, we explore how to maximize the system reward for embedded systems which provide services periodically with adjustable quality. The reward garnered for a service is monotonically increasing and concave with respect to the energy consumption of the service. As a main result, we provide algorithms to optimally adjust service parameters dynamically. Our work is supported by simulation results which are based on long-term measurements of the power generated by real solar cells. Furthermore, we demonstrate how to dimension the embedded system, e.g., the battery capacity and elaborate on implementation details which are of practical importance.

The remainder of this chapter is organized as follows: We start by summarizing related works in Section 4.2. Section 4.3 defines the system model and the studied problem in this chapter. Section 4.4 provides the proposed algorithms as well as illustrative examples, while Section 4.5 gives the related remarks for designing an embedded system. In Section 4.6, a simulative evaluation is presented which is based on real data recorded for photovoltaic cells. A short summary concludes the chapter.

4.2 Related Work

In the following, we will discuss some works which investigate power management for energy harvesting systems and are closely related to the work presented in this chapter.

In [KHZS07], Kansal et al. show how to maximize the utilisation of solar energy and minimize the losses in the battery by tuning the duty cycle of a sensor node. However, the objective of *maximizing the average duty* is completely different from the one in our chapter. In principal, the proposed heuristic in [KHZS07] attempts to decrease the duty cycle when the harvested energy is low (e.g. at night) and tries to increase the duty cycle when plenty of energy is harvested (e.g. during the day). By optimizing the sum of reward values which are concave over the energy consumption, our algorithms avoid this unbalanced behaviour and, e.g., try to prevent the embedded systems from shutting down during periods with little harvested energy.

Rusu et al. [RMM03] explore the reward maximization for a set of real-time tasks with multiple versions for execution by applying energy harvesting devices. The execution frames are divided into two types, namely recharging frames and discharging frames. These two types of frames are then executed by applying their static schedules individually. If the scheduler observes more energy residual in the battery, three dif-

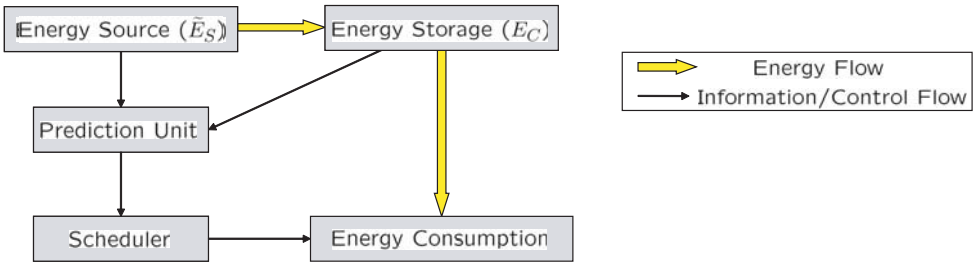


Fig. 34: Illustration of the system model.

ferent approaches are proposed to distribute the additional energy for getting more system reward. Our chapter focuses on a more fundamental problem to maximize the system reward globally, in which the energy consumption in all recharging frames and discharging frames might be different to achieve the global optimization.

4.3 System Model and Problem Statement

This chapter explores how to exploit energy harvested from the environment to maximize the system performance for applications with different reward functions. The harvested energy varies over time. For example, the energy harvested from a solar panel at a sunny noon is much more than the energy harvested at dawn. As a result, the system should adjust the quality of the associated applications to reflect the dynamic behavior for performance maximization instead of energy consumption minimization. In general, the more energy an application consumes, the more reward the system gains. However, an application might over-consume energy so that the possibility to gain more reward in the future is sacrificed.

This section will present the system model studied in this chapter, including the energy harvesting model for the energy source, the energy storage model, and the service and application models. The overview of the studied system is depicted in Figure 34, where the energy harvested from the energy source is stored in the energy storage, and the scheduler makes decisions for consuming the energy based on the information provided by the prediction unit and the available energy in the energy storage. At the end of this section, the problem definition will be presented.

4.3.1 Energy Harvesting Model

We are given an energy harvesting device, such as a solar panel, which generates energy depending on the environment. A prediction unit esti-

mates the energies harvested in each of the next K frames in the future, where K is the *number of frames of the prediction horizon*. We assume that each frame has the same length and the basic time unit is the length of one frame. We denote $\widetilde{E_S}(k)$ the accumulated energy harvested in the k-th frame. For instance, a frame may correspond to one hour, and having $K = 24$ would correspond to a prediction horizon of one day. How to determine a reasonable parameter K will be presented in Section 4.6.3 by means of simulation. For the rest of this chapter, we assume a perfect energy predictor. For a discussion about suitable energy prediction algorithms or how to handle prediction mistakes, the reader is referred to Section 4.5.1.

4.3.2 Energy Storage Model

The energy storage, e.g., a supercapacitor or a battery then stores the energy harvested from the environment. As energy storage is not a perfect process, there might be some loss of energy. The *efficiency factor* defined as the actually stored energy divided by the harvested energy can be used to model the storage process. Usually, the efficiency factor, denoted by $\eta(\widetilde{E_S})$, is a function of the harvested energy, and, by definition, is between 0 and 1. As a result, only $\eta(\widetilde{E_S}) \cdot \widetilde{E_H}(k)$ amount of the harvested energy will be stored to the energy storage in the k-th frame.

The amount of the harvested energy that will be stored to the energy storage in the k-th frame is denoted by $E_S(k)$, where $E_S(k)$ is $\eta(\widetilde{E_S}) \cdot \widetilde{E_H}(k)$. For simplicity of presentation, for the rest of this chapter, we denote $E_S(k)$ as the harvested energy in the k-th frame.

The energy storage is constrained by the maximum capacity $E_{\max}$ of the energy in the storage. If the energy storage is full, the additional harvested energy dissipates. Formally, suppose that $E_C(k)$ is the energy in the energy storage at the end of the k-th frame. After servicing the applications in the k-th frame with energy consumption e_k, the energy in the energy storage is $\min\{E_{\max}, E_C(k-1) + E_S(k) - e_k\}$. In other words, if $E_C(k-1) + E_S(k) - e_k$ is larger than $E_{\max}$, the system dissipates $E_C(k-1) + E_S(k) - e_k - E_{\max}$ amount of energy, which is a waste.

4.3.3 Application and Service Model

In every frame k, a set of N services is executed. For each service an energy consumption $\epsilon_{k,n}$ has to be chosen, where the index $1 \leq n \leq N$ indicates the respective service in each of the $1 \leq k \leq K$ frames of the prediction horizon. We assume that the energy consumption $\epsilon_{k,n}$ can have any positive value and is constant during each frame k. For extensions on how to handle upper and lower bounds on the energy consumption $\epsilon_{k,n}$ or

how to handle a discrete set of energies $\epsilon_{k,n}$, related remarks are provided in Section 4.5.4.

For a frame k, we denote e_k the total energy consumption of all services. Obviously, if one knows the energy consumptions $\epsilon_{k,n}$ of all services, one obtains the total energy consumption per frame

$$e_k = \sum_{n=1}^{N} \epsilon_{k,n}$$

by simple summation over all services.

This chapter explores how to achieve performance maximization for a variety of applications running on an embedded system, e.g., a sensor node. In fact, the service model presented in this chapter is capable of modeling numerous applications which are of practical importance. Two types of application models are studied in this chapter: one is the *computation-based* application model and the other is the *rate-based* application model. Both types of application models can be treated separately or can be even combined in a joint service model. For both application models, the quality of the provided services is evaluated in each frame k and is only dependent on the energy consumption $\epsilon_{k,n}$. Specifically, let $r_n(\epsilon_{k,n})$ denote the reward for executing the service n in a frame k with energy consumption $\epsilon_{k,n}$, where

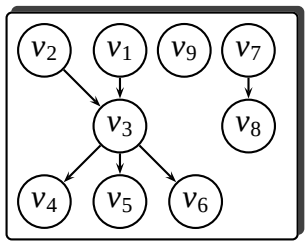
- $r_n(\epsilon_{k,n})$ is continuous and differentiable.
- $r_n(\epsilon_{k,n})$ is monotonically increasing in $\epsilon_{k,n}$.
- $r_n(\epsilon_{k,n})$ is concave in $\epsilon_{k,n}$, i.e.,

  $$\alpha \cdot r_n(\epsilon_{\kappa_1,n}) + (1-\alpha) \cdot r_n(\epsilon_{\kappa_2,n}) \leq r_n\left(\alpha \cdot \epsilon_{\kappa_1,n} + (1-\alpha) \cdot \epsilon_{\kappa_2,n}\right),$$

  for any two frames $1 \leq \kappa_1, \kappa_2 \leq K$ with $\epsilon_{\kappa_1,n}\epsilon_{\kappa_2,n} \geq 0$, $\epsilon_{\kappa_1,n} \neq \epsilon_{\kappa_2,n}$, and $1 > \alpha > 0$.

Computation-based application model

For the classical *computation-based* application model, all applications are independent from each other. Moreover, the higher the computation/workload/demand of the assigned service, the more reward the system gains for the execution, such as the imprecise computation model [LLS+91] and the increasing reward with increasing service (IRIS) model [DKT96, SLC91]. The quality of the provided service is evaluated in each frame, in which the reward function is a strictly concave and increasing function of the amount of computation, such as image and speech processing, time-dependent planning, and multimedia applications. The



$$\rho(v_1) + \rho(v_2) = \rho(v_3)$$
$$\rho(v_4) + \rho(v_5) + \rho(v_6) = \rho(v_3)$$
$$\rho(v_7) = \rho(v_8)$$

Fig. 35: An example for rate graphs.

energy consumption for a given workload of provided service is assumed to be a convex function (when dynamic voltage scaling is adopted [CK05, YDS95]) or a linear function (when the power consumption is a constant). Therefore, the reward $r_n(\epsilon_{k,n})$ is a strictly concave and increasing function of the energy consumption $\epsilon_{k,n}$. For the rest of this chapter, we will only discuss the amount of energy consumption $\epsilon_{k,n}$, while the required computation time to complete the service in a frame with the specified energy consumption can be derived by simple calculation. In some cases, the maximum time to finish a service within a frame may be specified by a deadline. This translates into a minimum energy consumption $\epsilon_{k,n}$, which can also be handled by our algorithms presented in the next sections. For a discussion on minimum and maximum energy consumptions $\epsilon_{k,n}$ of a service, the reader is referred to Section 4.5.4.

Rate-based application model

For the *rate-based* application model, the reward of a service depends on the rate to activate the application, while the computation demand and the energy consumption are fixed for one activation. For example, a sensor application which observes an environmental phenomenon may be more precise when the sensing rate is higher, where each sensing activity takes the same time and consumes the same energy.

In contrast to the computation-based model, for the rate-based model, the rates of applications can not be set independently but need to satisfy some constraints. For instance, a sensor node may sense some data, and then decide whether to compress and store the data in some local memory or to directly transmit the data via the radio. Clearly, the sum of stored and transmitted data has to equal the amount of originally sensed data. In this chapter, we explore rate-conserving applications that can be described by rate-based graphs (RBGs) defined as follows: A rate-based graph is a directed tree, denoted by $G = (\mathcal{V}, \mathcal{E})$, in which a vertex in $\mathcal{V}$ is an application and a directed edge $(u \rightarrow v)$ in $\mathcal{E}$ is the relationship between the rates of two applications. Suppose that $\rho(v)$ is the rate of an application in vertex v in $\mathcal{V}$. If $(u \rightarrow v)$ in $\mathcal{E}$ is the only incoming edge to

application v and is the only outgoing edge from application u, the rates of applications u and v are the same. If there are more than one incoming edge to an application v, the rate of the application v must be the sum of the rates of the applications that have directed outgoing edges to vertex v, i.e., $\rho(v) = \sum_{(u \to v) \in \mathcal{E}} \rho(u)$. Moreover, if there are more than one outgoing edge from an application v, the rate of the application v must be the sum of the rates of the applications that have directed incoming edges from vertex v, i.e., $\rho(v) = \sum_{(v \to u) \in \mathcal{E}} \rho(u)$. Figure 35 depicts an example for 9 applications and their rate relationships.[1]

For the rate-based model, once we assign an energy consumption to an application, the corresponding rate of the application is known. Furthermore, we assume the reward $r_n(\epsilon_{k,n})$ to be a strictly concave and increasing function of the energy consumption $\epsilon_{k,n}$. This assumption is justified since, e.g., for many audio or video applications in sensor networks the reward saturates for high transmission rates, i.e., for high rates the additional perceived reward is getting smaller.

For each frame, the scheduler has to determine how to provide the services of different applications. As already stated at the beginning of this section, we assume the reward $r_n(\epsilon_{k,n})$ for executing a service n in frame k to be concave over the energy consumption $\epsilon_{k,n}$. For the rest of this chapter, we will only discuss the amount of energy consumption $\epsilon_{k,n}$ of an application in each frame, while the corresponding computation times and rates can be derived by simple calculation.

4.3.4 Problem Definition

We are given a predictor for K frames at time 0. The energy in the energy storage at time 0 is specified as $E_C(0)$ and the energy harvested in the k-th frame is $E_S(k)$. The k-th frame starts from time $k-1$ to time k. A set of N applications is going to be executed in each frame, in which each application has its reward function $r_n(\epsilon_{k,n})$ of the energy consumption.

The objective is to determine an *inter-frame energy assignment*

$$\vec{e} = (e_1, e_2, \ldots, e_K)$$

of energy consumption for these K frames and *intra-frame energy assignments*

$$\vec{\epsilon_k} = (\epsilon_{k,1}, \epsilon_{k,2}, \ldots, \epsilon_{k,N})$$

for the N applications within the k-th frame such that the sum of rewards $\sum_{k=1}^{K} \sum_{n=1}^{N} r_n(\epsilon_{k,n})$ is maximized without violating the required energy constraints.

[1] The approaches in this chapter also work for rate-based graphs with scaling factors on the edges. For example, it could be $0.5\rho(v_1) + 0.3\rho(v_2) = \rho(v_3)$.

According to the rate-based application model, the energy consumptions of some applications in a frame might be related. For example, suppose the energy consumption of application v_n is σ_n when its rate is 1. The relationship $\rho(v_1) + \rho(v_2) = \rho(v_3)$ in Figure 35 leads to $\frac{\epsilon_{k,1}}{\sigma_1} + \frac{\epsilon_{k,2}}{\sigma_2} = \frac{\epsilon_{k,3}}{\sigma_3}$, where $\epsilon_{k,n}$ is the energy consumption of application v_n in the respective frame k. By bringing these equations in a left-hand side form, we write the constraints from the rate-based graphs (RBG) in the form $c_m(\vec{\epsilon_k}) = 0$, where $m = 1, \ldots, M$. The constraints $c_m(\vec{\epsilon_k})$ can be arbitrary linear combinations of the assignment $\vec{\epsilon_k}$ which conserve the rates of the application.

Let $E_C(k, \vec{e})$ be the energy in the energy storage at time k by applying the assignment of energy consumption $\vec{e}$. After completing the last frame, we would like to reserve some amount E_ℓ of energy in the energy storage for future use, and, hence, a feasible assignment $\vec{e}$ must satisfy $E_C(K, \vec{e}) \geq E_\ell$. We denote the studied problem as the *general reward maximization on energy harvesting* problem. Without loss of generality, we only explore the case that $E_C(0) - E_\ell + \sum_{i=1}^{K} E_S(i) \geq 0$ in this chapter since there is no feasible solution for the other case.

We formally define the feasibility of an assignment for the general reward maximization on energy harvesting problem as follows:

Def. 1: *[Feasible Assignment] An energy vector* $\vec{e} = (e_1, \ldots, e_K)$ *along with* $\vec{\epsilon_k} = (\epsilon_{k,1}, \epsilon_{k,2}, \ldots, \epsilon_{k,N})$ *for all these K frames is feasible if*

(a) $E_C(k, \vec{e}) = \wedge E_{\max}, E_C(k-1, \vec{e}) + E_S(k) - e_k$,
where $E_C(0, \vec{e})$ *is* $E_C(0)$,

(b) $E_C(k, \vec{e}) \geq 0, \forall 1 \leq k < K$,

(c) $E_C(K, \vec{e}) \geq E_\ell$,

(d) $\sum_{n=1}^{N} \epsilon_{k,n} = e_k$, *and*

(e) $c_m(\vec{\epsilon_k}) = 0, \forall 1 \leq m \leq M$.

For simplicity of presentation, in property *(b)* of Definition 1, we assume that the energy storage can be emptied completely for all frames $1 \leq k < K$. Note, however, that the algorithms presented in the next section can be extended easily to cases where a minimum energy $E_{\min}$ has to be stored at all times, i.e. $E_C(k, \vec{e}) \geq E_{\min}, \forall 1 \leq k < K$.

An assignment is said *optimal* for the general reward maximization on energy harvesting problem if its accumulated reward $\sum_{k=1}^{K} \sum_{n=1}^{N} r_n(\epsilon_{k,n})$ is the maximum among all feasible assignments. We say there exists an *energy underflow* for an assignment $\vec{e}$ if there exists $E_C(k, \vec{e}) < 0$ for some $1 \leq k \leq K-1$ or $E_C(K, \vec{e}) < E_\ell$. On the other hand, an assignment $\vec{e}$ is said with *energy overflow* if there exists some k with $E_C(k-1, \vec{e}) + E_S(k) - e_k > E_{\max}$.

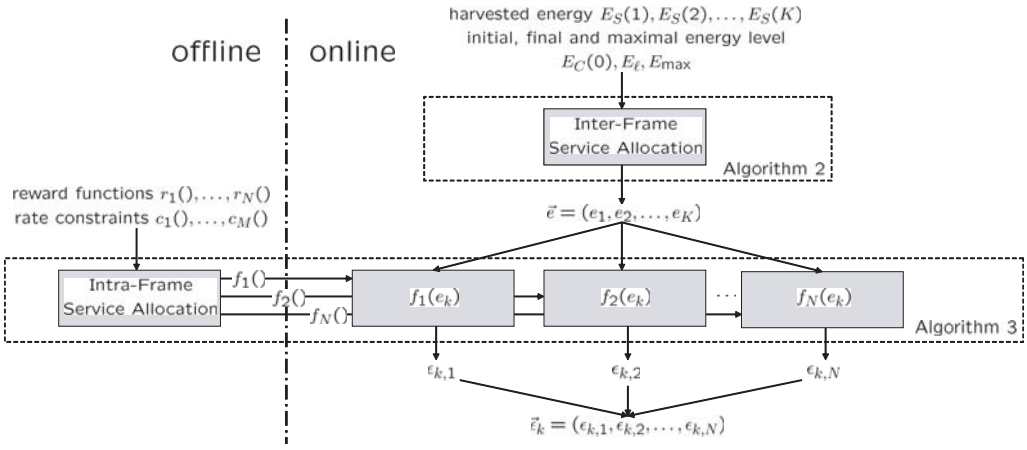


Fig. 36: Overview of the two-stage service level allocation framework.

4.4 Proposed Service Allocation Framework

In this section, we will show how to optimally allocate services subject to the available energy given by the environment and the energy storage. Thereby, we exploit the nature of the general reward maximization on energy harvesting problem and split it into two subproblems, namely the inter-frame problem and the intra-frame problem, which can be solved independently. As illustrated in Figure 36, an Inter-Frame Service Allocation Algorithm focuses on finding inter-frame energy assignments $\vec{e} = (e_1, e_2, \ldots, e_K)$. The respective algorithm will be presented in Section 4.4.1 and has to be executed once per prediction horizon. As input data, it takes a vector of the harvested energies $E_S(1), E_S(2), \ldots, E_S(K)$, the initially stored energy $E_C(0)$, the capacity $E_{\max}$ as well as the final energy E_ℓ. As a matter of fact, the Inter-Frame Service Allocation Algorithm solves the fundamental problem of handling energy underflows and overflows.

The intra-frame problem copes with computing intra-frame energy assignment $\vec{\epsilon}_k = (\epsilon_{k,1}, \epsilon_{k,2}, \ldots, \epsilon_{k,N})$ for every k-th frame. The energy consumption of a frame in assignment $\vec{e}$ can be treated as the energy budget which has to be distributed among all running applications. Depending on the reward functions $r_1(), \ldots, r_N()$ of the applications and the constraints $c_1(), \ldots, c_M()$ from possible rate-based graphs, the intra-frame energy assignment can be calculated directly using the Intra-Frame Service Allocation Algorithm presented in Section 4.4.2. Alternatively, as we will see in Section 4.6.2, it is much more efficient to compute explicit solutions $f_1(), \ldots, f_N()$ offline and compute the energy consumptions $\epsilon_{k,1}, \epsilon_{k,2}, \ldots, \epsilon_{k,N}$ of the applications in the online case by simply evaluating the functions $f_1(), \ldots, f_N()$. In any case, the Intra-Frame Service Allocation has to be repeated for every frame k in the prediction horizon.

This section is organized as follows: In Section 4.4.1, we present an

optimal algorithm for Inter-Frame Service Allocation. For the sake of clearness, we start by presenting an algorithm for energy storages with unlimited capacity in Section 4.4.1.1. In Section 4.4.1.2, the algorithm is then extended to general cases by considering a limited energy storage capacity and we denote this algorithm as Inter-Frame Service Allocation. Section 4.4.1.3 provides the proofs for the optimality of the Inter-Frame Service Allocation Algorithm. Finally, the Intra-Frame Service Allocation and proofs for the optimality of the overall two-staged service allocation concept are presented in Section 4.4.2.

4.4.1 Inter-Frame Service Allocation

We will now present how to derive an optimal inter-frame energy assignment $\vec{e}$. The proposed algorithm is presented in Algorithm **??**, denoted by Algorithm Inter-Frame Service Allocation. It can be proved to derive optimal solutions for systems with only one application with any concave reward function. Moreover, if there is more than one application, Section 4.4.2 will show that these applications can be treated as one application with a *joint reward function*, which is concave, too. Therefore, for the sake of simplicity, we will now consider only one application with a concave reward function. For the whole Section 4.4.1, we will denote e_k the energy consumption of this single application in the k-th frame (instead of $\epsilon_{k,1}$). Similarly, we will simply call the reward function $r()$ instead of $r_1()$.

4.4.1.1 A Greedy Algorithm for Unlimited Energy Storage Capacity

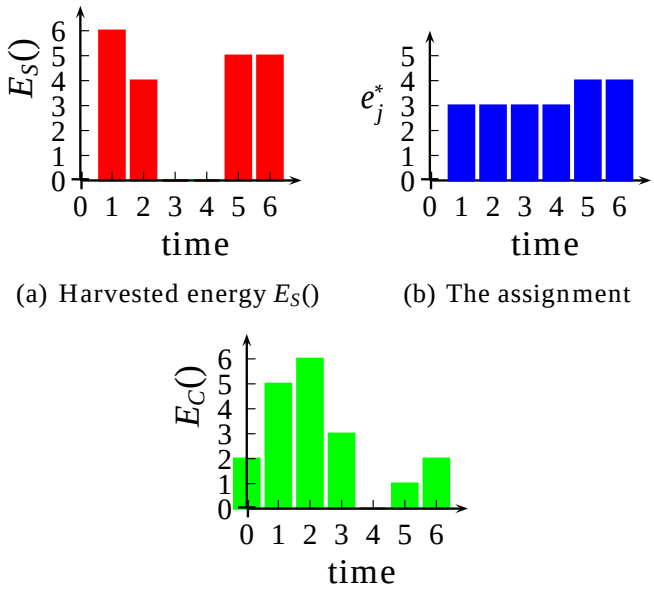
Based on the concavity of the reward function, the following lemma holds.

Lem. 1: *If* $e_{k_1}+e_{k_2} = e_{k_3}+e_{k_4}$ *with* $0 \leq e_{k_1} < e_{k_3}, e_{k_4} < e_{k_2}$, *then* $r(e_{k_1})+r(e_{k_2}) < r(e_{k_3})+r(e_{k_4})$.

Proof. Let α_3 be $\frac{e_{k_2}-e_{k_3}}{e_{k_2}-e_{k_1}}$ and α_4 be $\frac{e_{k_2}-e_{k_4}}{e_{k_2}-e_{k_1}}$. Since $e_{k_1} < e_{k_3}, e_{k_4} < e_{k_2}$, we have $0 < \alpha_3 < 1$ and $0 < \alpha_4 < 1$. Because $e_{k_1} + e_{k_2} = e_{k_3} + e_{k_4}$, we know that $\alpha_3 + \alpha_4 = 1$. Therefore, by the concavity, we conclude $r(e_{k_1}) + r(e_{k_2}) = (\alpha_3+\alpha_4)r(e_{k_1})+(2-\alpha_3-\alpha_4)r(e_{k_2}) < r(\alpha_3 e_{k_1}+(1-\alpha_3)e_{k_2})+r(\alpha_4 e_{k_1}+(1-\alpha_4)e_{k_2}) = r(e_{k_3}) + r(e_{k_4})$.

□

Lemma 1 tells us, that to optimally exploit the concavity of the reward function and maximize the sum of rewards, one would like to consume a constant, average amount of energy for all K frames of the prediction horizon. However, due to the burst of the given energy source E_S, this is often not possible. The key idea to obtain an optimal assignment is to



(a) Harvested energy $E_S()$ (b) The assignment

(c) Stored energy $E_C()$

Fig. 37: The solution derived by Algorithm Greedy-Incremental for $K = 6$, $E_{\max} = +\infty$.

equate energies of an assignment as much as possible subject to feasibility constraints.

For the case of an unlimited energy storage with $E_{\max} = +\infty$, we only have to deal with energy underflows since energy overflows are not possible. Here, the harvested energy might not be enough to support the energy consumption. For example, as shown in Figure 38(a), if K is 6 with $E_\ell = E_C(0) = 2, E_S(1) = 6, E_S(2) = 4, E_S(3) = 0, E_S(4) = 0, E_S(5) = 5, E_S(6) = 5$, Lemma 1 suggests to have an assignment $\vec{e}$ with $\frac{6+4+5+5}{6} = \frac{10}{3}$ unit of energy consumption for all these six frames. However, according to Definition 1, the resulting assignment is not feasible since there is an energy underflow with $E_C(4, \vec{e}) = -\frac{4}{3}$. Therefore, an optimal assignment should try to consume some constant amounts of energy without leading to an energy underflow.

Let k be the index of the last frame that has been assigned so far, in which k is initialized as 0. For each j with $j = k+1, k+2, \ldots, K$, the maximum amount of energy that is allowed to consume from time k to time j is $E_C(k) + \sum_{i=k+1}^{j} E_S(i)$. If we decide to consume $E_C(k) + \sum_{i=k+1}^{j} E_S(i)$ amount of energy from time k to time j, an assignment, ignoring feasibility constraints, should consume $\frac{E_C(k)+\sum_{i=k+1}^{j} E_S(i)}{j-k}$ amount of energy for each of the frames from the $(k+1)$-th frame to the j-th frame. Let $\widetilde{e}_j$ be $\frac{E_C(k)+\sum_{i=k+1}^{j} E_S(i)}{j-k}$, $\forall j = k+1, k+2, \ldots, K$. Clearly, when $\widetilde{e}_j \geq \widetilde{e}_{k^*}$ for every index $k < j < k^*$, a partial assignment for the $k+1$-th frame to the k^*-th frame with a constant

Algorithm 4 Greedy-Incremental (GI)

Input: K, $E_S(k)$ for $k = 1, 2, \ldots, K$, $E_C(0)$, E_ℓ;
Output: a feasible assignment $\vec{e}^*$ of energy consumption for the K frames;

1: $k \Leftarrow 0$;
2: **while** $k < K$ **do**
3: let $\widetilde{e_j}$ be $\frac{E_C(k)+\sum_{i=k+1}^{j} E_S(i)}{j-k}$, $\forall j = k+1, k+2, \ldots, K-1$;
4: let $\widetilde{e_K}$ be $\frac{E_C(k)-E_\ell+\sum_{i=k+1}^{K} E_S(i)}{K-k}$;
5: $k^* \Leftarrow \max\{\arg\min_{k<\hat{k}\leq K}\{\widetilde{e_{\hat{k}}}\}\}$;
6: $e_j^* \Leftarrow \widetilde{e_{k^*}}$, $\forall k+1 \leq j \leq k^*$;
7: **if** $k^* = K$ **then**
8: $E_C(k^*) \Leftarrow E_\ell$;
9: **else**
10: $E_C(k^*) \Leftarrow 0$;
11: $k \Leftarrow k^*$;
12: return $\vec{e}^*$ as the solution;

amount of energy consumption $\widetilde{e_{k^*}}$ will lead to a solution with $E_C(j) \geq 0$ for any $k < j \leq k^*$. Therefore, we find the maximum index k^* in which $\widetilde{e_j} \geq \widetilde{e_{k^*}}$ for every index $k < j < k^*$, and then assign energy consumption $\widetilde{e_{k^*}}$ to any j-th frame with $j = k+1, k+2, \ldots, k^*$. Then, we can update the index k as k^* and repeat the above procedure. However, since we have a constraint on the residual energy in the energy storage after completing the K-th frame, $\widetilde{e_K}$ should be revised as $\frac{E_C(k)-E_\ell+\sum_{i=k+1}^{K} E_S(i)}{K-k}$ to guarantee that the residual energy at time K is E_ℓ.

The proposed algorithm is presented in Algorithm 4, denoted by Algorithm Greedy-Incremental (GI) for the rest of this chapter. With the initialization of k as 0, we calculate the values $\widetilde{e_j}$ in Step 3 and Step 4 for every $j = k+1, k+2, \ldots, K$. Then, we find the maximum index k^*, in which $\widetilde{e_j} \geq \widetilde{e_{k^*}}$ for every index $k < j < k^*$, in Step 5 with the assignment of energy consumption $\widetilde{e_{k^*}}$ for every of the frames from the $k+1$-th frame to the k^*-th frame. Clearly, $E_C(k^*)$ is 0 (E_ℓ, respectively) when k^* is less than K (k^* is equal to K, respectively). Then, the algorithm goes to the next loop by updating k as k^*. The time complexity of the algorithm is $O(K^2)$ with a proper implementation of the summation in Step 3 and Step 4 in Algorithm 4. In section 4.4.1.3 we will provide the proof for the optimality of the assignment $\vec{e}^*$ for the reward maximization problem with a single application and unlimited energy capacity $E_{\max} = +\infty$.

Applying Algorithm GI to the example in the first paragraph in this subsection would lead to a solution as shown in Figure 37(b). When k is 0, we have $\widetilde{e_1} = 8, \widetilde{e_2} = 6, \widetilde{e_3} = 4, \widetilde{e_4} = 3, \widetilde{e_5} = 3.4, \widetilde{e_6} = \frac{10}{3}$, and, hence, e_1^*, e_2^*, e_3^*, and e_4^* are set to 3 since k^* is 4. Then, when k is 4, we have $\widetilde{e_5} = 5, \widetilde{e_6} = 4$. Therefore, e_5^* and e_6^* are set to 4. Figure 37(c) shows the stored energy $E_C()$

over time.

We have the following lemmas for the derived solution.

Lem. 2: *The derived solution from Algorithm Greedy-Incremental consumes energy non-decreasingly in these K frames.*

Proof. Suppose that $e_i^* > e_{i+1}^* = e_{i+2}^* = \cdots = e_j^* \neq e_{j+1}^*$ for some $1 \leq i \leq K$ for contradiction. (For brevity, we assume e_{K+1}^* is ∞.) Then, when determining the energy consumption e_i^*, we have $\widetilde{e_i} > \widetilde{e_j}$, which contradicts the selection of k^* in Algorithm Greedy-Incremental.

□

Lem. 3: *The derived solution from Algorithm Greedy-Incremental is feasible.*

Lem. 4: *The derived solution from Algorithm Greedy-Incremental consumes energy* $E_C(0) - E_\ell + \sum_{i=1}^{K} E_S(i)$ *in these K frames.*

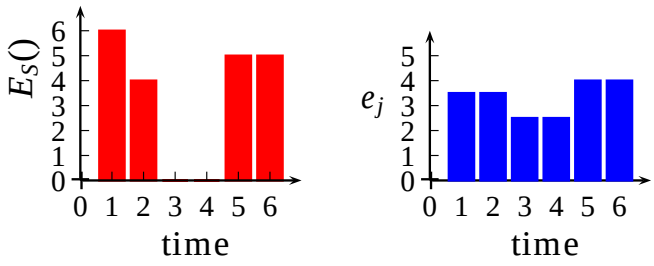
Proof. The above lemmas come directly from the construction.

□

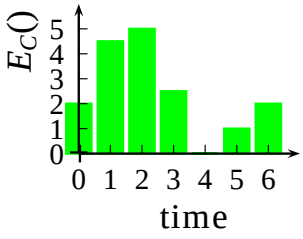
4.4.1.2 An Algorithm for Limited Energy Storage Capacity

We now deal with systems with energy storage capacity constraints, i.e., $E_{\max} \neq \infty$. The derived solution $\vec{e}^*$ is guaranteed to have no energy underflow if there is no energy overflow. But, when $E_{\max} \neq \infty$, there might be some energy overflows in assignment $\vec{e}$, and will lead to some energy underflows since the derived solution tries to consume all the harvested energy. Consider the setting of $E_{\max} = 5$ for the example in the first paragraph of Section 4.4.1.1. The derived solution of Algorithm GI has energy overflow at time 2. Then, the derived solution $\vec{e}$ will later have energy underflow, e.g., at time 4 in this example. Therefore, if the resulting schedule has energy overflow, we should consume more energy to avoid energy overflow so that we can get more reward.

Beginning from the solution $\vec{e}^*$ derived from Algorithm GI, we revise the solution to prevent from energy overflow. The assignment $\vec{e}^*$ can be segmented into *N segments*, in which, in assignment $\vec{e}^*$, all the frames in a segment are with the same energy consumption and any two frames in different segments are with different energy consumptions. Suppose that k_n is the index of the frame at the end of the n-th segment for every $n = 1, 2, \ldots, N$, where k_0 is initialized as 0 for notational brevity. The proposed algorithm, denoted by Algorithm Inter-Frame Service Allocation as shown in Algorithm 5, then revises $\vec{e}^*$ individually in these N segments by calling a recursive procedure subSeg().



(a) Harvested energy $E_S()$ (b) The assignment



(c) Stored energy $E_C()$

Fig. 38: The solution derived by Algorithm Inter-Frame Service Allocation for $K = 6$, $E_{\max} = 5$.

The SUBSEG() procedure takes four parameters k', K', E'_C, and E'_ℓ as its input to derive a feasible assignment between the $(k'+1)$-th frame and the K'-th frame (from time k' to time K') with energy in the energy storage E'_C at time k' and E'_ℓ at time K'. When $K'-k'$ is 1, it is clear to execute the K'-th frame with energy consumption $E_S(K')+E'_C-E'_\ell$. When $K'-k'$ is more than 1, we have to check whether there will be energy underflow or energy overflow. Firstly, let $e^\flat$ be the average energy consumption for executing these $K'-k'$ frames, i.e., $e^\flat \Leftarrow \frac{E'_C-E'_\ell+\sum_{i=k'+1}^{K'} E_S(i)}{K'-k'}$ in Step 3 in Procedure SUBSEG in Algorithm 5. For every $k' < j < K'$, let $e^\sharp_j$ be the maximum average energy consumption from time j to time K', where $e^\sharp_j \Leftarrow \frac{E_{\max}-E'_\ell+\sum_{i=j+1}^{K'} E_S(i)}{K'-j}$ in Step 4 in Procedure SUBSEG in Algorithm 5. Similarly, let $\hat{e}_j$ be the maximum average energy consumption from time k' to time j, where $\hat{e}_j \Leftarrow \frac{E'_C+\sum_{i=k'+1}^{j} E_S(i)}{j-k'}$ in Step 5 in Procedure SUBSEG in Algorithm 5. Let $k^\sharp$ be the index j with the minimum $e^\sharp_j$ and $\hat{k}$ be the index j with the minimum $\hat{e}_j$, where ties are broken arbitrarily. If both $e^\sharp_{k^\sharp}$ and $\hat{e}_{\hat{k}}$ are no less than $e^\flat$, the procedure returns the assignment to consume energy $e^\flat$ from the $k'+1$-th frame to the K'-th frame. Otherwise these $K'-k'$ frames are divided into two sub-segments. If $e^\sharp_{k^\sharp}$ is less than $e^\flat$, the frames from the k'-th frame to the $k^\sharp$ frame are restricted to leave energy $E_{\max}$ in the energy storage at time

$k^\sharp$ by calling subSeg($k', k^\sharp, E'_C, E_{\max}$) and subSeg($k^\sharp, K', E_{\max}, E'_\ell$). Otherwise, if $\hat{e}_{\hat{k}}$ is less than $e^\flat$, the frames from the k'-th frame to the $\hat{k}$ frame are restricted to leave no energy in the energy storage at time $\hat{k}$ by calling subSeg($k', \hat{k}, E'_C, 0$) and subSeg($\hat{k}, K', 0, E'_\ell$).

Algorithm 5 Inter-Frame Service Allocation

Input: K, $E_S(k)$ for $k = 1, 2, \ldots, K$, $E_C(0)$, E_ℓ, $E_{\max}$;
Output: a feasible assignment $\vec{e}$ of energy consumption for the K frames;
1: let $\vec{e}^*$ be the solution derived from Algorithm 4;
2: divide the K frames into N segments and let k_n be the index of the frame at the end of the n-th segment for every $n = 1, 2, \ldots, N$, where k_0 is 0;
3: **for** $n = 1; n \le N; n \Leftarrow n + 1$ **do**
4: let $e_{(k_{n-1}+1)}, e_{(k_{n-1}+2)}, \ldots, e_{k_n}$ be the resulting assignment by calling subSeg(k_{n-1}, k_n, E'_C, E'_ℓ), where E'_C is $E_C(0)$ when $n = 1$, E'_C is 0 for any $n > 1$, E'_ℓ is 0 for any $n < N$, and E'_ℓ is E_ℓ when $n = N$;
5: return $\vec{e}$ as the solution;

Procedure: subSeg()
Input: (k', K', E'_C, E'_ℓ);
Output: a feasible assignment of energy consumption for the frames from the $(k'+1)$-th frame to the K'-th frame;
1: **if** $K' - k' = 1$ **then**
2: return the assignment by consuming $E'_C + E_S(K') - E'_\ell$ for the K'-th frame;
3: $e^\flat \Leftarrow \frac{E'_C - E'_\ell + \sum_{i=k'+1}^{K'} E_S(i)}{K' - k'}$;
4: $e^\sharp_j \Leftarrow \frac{E_{\max} - E'_\ell + \sum_{i=j+1}^{K'} E_S(i)}{K' - j}$ for every $k' < j < K'$;
5: $\hat{e}_j \Leftarrow \frac{E'_C + \sum_{i=k'+1}^{j} E_S(i)}{j - k'}$ for every $k' < j < K'$;
6: $k^\sharp \Leftarrow \arg_{k'<j<K'} \wedge e^\sharp_j$;
7: $\hat{k} \Leftarrow \arg_{k'<j<K'} \wedge \hat{e}_j$;
8: **if** $e^\sharp_{k^\sharp} < e^\flat$ **then**
9: subSeg($k', k^\sharp, E'_C, E_{\max}$);
10: subSeg($k^\sharp, K', E_{\max}, E'_\ell$);
11: **else if** $\hat{e}_{\hat{k}} < e^\flat$ **then**
12: subSeg($k', \hat{k}, E'_C, 0$);
13: subSeg($\hat{k}, K', 0, E'_\ell$);
14: **else**
15: return the assignment to consume $e^\flat$ from the $k'+1$-th frame to the K'-th frame;

Applying Algorithm Inter-Frame Service Allocation on the example in Figure 38(a) with $E_{\max} = 5$, and $E_\ell = E_C(0) = 2$, the first segment with $k_1 = 4$ will be divided into two segments by taking $k^\sharp$ as 2 and $e^\sharp_2 = 2.5$. As a result, $\vec{e}$ is $(3.5, 3.5, 2.5, 2.5, 4, 4)$. In Figure 38, the energies $\vec{e}$ and the

resulting stored energies $E_C()$ are displayed.

The time complexity of Algorithm Inter-Frame Service Allocation is $O(K^2)$ since the time complexity for the n-th segment is $O((k_n - k_{n-1})^2)$, while $\sum_{n=1}^{N} O((k_n - k_{n-1})^2) = O(K^2)$. In section 4.4.1.3 we will provide the proof for the optimality of the assignment $\vec{e}$ for the reward maximization problem with a single application.

We have the following lemma for the derived solution.

Lem. 5: *The derived solution from Algorithm Inter-Frame Service Allocation is feasible.*

Proof. Since the assignment is determined in Step 2 and Step 15 in Procedure subSeg in Algorithm 5, it is clear that there is no energy overflow or energy underflow in the derived solution.

□

4.4.1.3 Proof for the Optimality of the Inter-Frame Energy Allocation Algorithm

This section provides the optimality of Algorithm GI and Algorithm Inter-Frame Service Allocation. Due to space limitation, some proofs are only sketched. The following property from the concavity of the reward function must hold for any optimal solution:

Lem. 6: **(Jumping of Energy Consumption)** *For an optimal energy vector $\vec{e}$, (1) if $e_i < e_{i+1}$, $E_C(i, \vec{e})$ is 0; (2) if $e_i > e_{i+1}$, $E_C(i, \vec{e})$ is $E_{\max}$.*

Proof. Let $E_C(i, \vec{e})$ be γ. We prove this lemma by contradiction. Suppose that $e_i < e_{i+1}$ and $\gamma > 0$. Let $e_i^{\perp}$ be $\wedge e_i + \gamma, \frac{e_i + e_{i+1}}{2}$, while $e_{i+1}^{\perp}$ is $e_i + e_{i+1} - e_{i^{\perp}}$. Let $e_j^{\perp}$ be e_j for any $j \neq i, i+1$. It is clear that $E_C(j, \vec{e})$ and $E_C(j, \vec{e^{\perp}})$ are the same for any $j \neq i$. By the definition of $e_i^{\perp}$, $E_C(i, \vec{e^{\perp}})$ must be no less than 0, and hence $\vec{e^{\perp}}$ is feasible. Moreover, we know that $e_i < e_i^{\perp}, e_{i+1}^{\perp} < e_{i+1}$ with $e_i + e_{i+1} = e_i^{\perp} + e_{i+1}^{\perp}$. By Lemma 1, assignment $\vec{e^{\perp}}$ has more reward than assignment $\vec{e}$, which contradicts the optimality of $\vec{e}$.

Suppose that $e_i > e_{i+1}$ and $\gamma < E_{\max}$. Let $e_i^{\perp}$ be $\vee e_i - E_{\max} + \gamma, \frac{e_i + e_{i+1}}{2}$, while $e_{i+1}^{\perp}$ is $e_i + e_{i+1} - e_i^{\perp}$. Similar to the argument of the first case, we will have contradiction to the optimality of assignment $\vec{e}$.

□

Moreover, the following lemma shows that there is no energy waste for an optimal assignment.

Lem. 7: **(Total Energy)** *For an optimal energy vector $\vec{e}$, $\sum_{i=1}^{K} e_i = E_C(0) + \sum_{i=1}^{K} E_S(i) - E_\ell$.*

Proof. If $\sum_{i=1}^{K} e_i < E_C(0) + \sum_{i=1}^{K} E_S(i) - E_\ell$, assignment $\vec{e}$ is clearly sub-optimal since the reward function is an increasing function. On the other hand, if $\sum_{i=1}^{K} e_i > E_C(0) + \sum_{i=1}^{K} E_S(i) - E_\ell$, $\vec{e}$ is not feasible. Both contradict the optimality of $\vec{e}$.

□

Optimality of Algorithm Greedy-Incremental

Based on Lemmas 2, 3, 6, and 7, we can prove the optimality of Algorithm GI when there is no limitation on the energy storage capacity.

Thm. 8: *Algorithm Greedy-Incremental derives optimal assignments for the general reward maximization on energy harvesting problem for one application* ($N = 1$) *when* $E_{\max} = \infty$.

Proof. Based on Lemma 6 and Lemma 7, an optimal assignment $\vec{e}^{\perp}$ must have non-decreasing energy consumption with $\sum_{i=1}^{K} e_i^{\perp} = E_C(0) + \sum_{i=1}^{K} E_S(i) - E_\ell$. Suppose that $\vec{e}^{\perp}$ is different from the assignment $\vec{e}^{*}$ derived by Algorithm Greedy-Incremental. By adopting the segmentation terminology for $\vec{e}^{*}$ at the beginning of Section 4.4.1.2. Suppose that n' is the first segment that $\vec{e}^{*}$ and $\vec{e}^{\perp}$ differs from each other. By definition, $e^*_{(k_{n'-1}+1)} = e^*_{(k_{n'-1}+2)} = \cdots = e^*_{k_{n'}}$. Let κ be the index of the first frame that $\vec{e}^{*}$ and $\vec{e}^{\perp}$ differs from each other.

If $e^*_\kappa < e^{\perp}_\kappa$, we have $\sum_{i=k_{n'-1}+1}^{k_{n'}} e_i^* < \sum_{i=k_{n'-1}+1}^{k_{n'}} e_i^{\perp}$, since $e^{\perp}_\kappa \le e^{\perp}_{\kappa+1} \le \cdots \le e^{\perp}_{k_{n'}}$ and, by Lemma 2, $e^*_\kappa \le e^*_{\kappa+1} \le \cdots \le e^*_{k_{n'}}$. Therefore, assignment $\vec{e}^{\perp}$ has some energy underflow, which contradicts the feasibility of $\vec{e}^{\perp}$.

If $e^*_\kappa > e^{\perp}_\kappa$, because $e^*_\kappa \le e^*_{\kappa+1} \le \cdots \le e^*_K$, we know that $E_C(k_{n'-1} + 1, \vec{e}^{*}) + \sum_{i=k_{n'-1}+1}^{k} e_i^{\perp} < \sum_{i=k_{n'-1}+1}^{k} e_i^* <^1 E_C(k_{n'-1} + 1, \vec{e}^{*}) + \sum_{i=k_{n'-1}+1}^{k} E_S(i)$ for any $k_{n'-1} + 1 \le k < K$ and $E_C(k_{n'-1} + 1, \vec{e}^{*}) + \sum_{i=k_{n'-1}+1}^{K} e_i^{\perp} < \sum_{i=k_{n'-1}+1}^{K} e_i^* <^2 E_C(k_{n'-1} + 1, \vec{e}^{*}) - E_\ell + \sum_{i=k_{n'-1}+1}^{K} E_S(i)$, where $<^1$ and $<^2$ come from Lemma 3. Therefore, there is energy waste in assignment $\vec{e}^{\perp}$, which contradicts the optimality of $\vec{e}^{\perp}$.

As a result, assignment $\vec{e}^{*}$ derived by Algorithm Greedy-Incremental is optimal for the general reward maximization on energy harvesting problem if a single application is executed on a system with $E_{\max} = \infty$.

□

Optimality of Algorithm Inter-Frame Energy Allocation

We now show the optimality of Algorithm Inter-Frame Service Allocation for the general reward maximization on energy harvesting problem when

there is some limitation on the energy storage capacity. It is obvious that the derived assignment $\vec{e}$ does not have energy waste, i.e., $\sum_{i=1}^{K} e_i = E_C(0) + \sum_{i=1}^{K} E_S(i) - E_\ell$. The following lemma is needed to prove the optimality property in Lemma 6.

Lem. 8: *For procedure* subSeg*() with specified parameters k', K', E'_C, and E'_ℓ in Algorithm 5, (1) if $e^\sharp_{k^\sharp} \geq e^\flat$ and $\hat{e}_{\hat{k}} < e^\flat$, then, $e_{\hat{k}} \leq e_{\hat{k}+1}$; (2) if $e^\sharp_{k^\sharp} < e^\flat$, $e_{k^\sharp} \geq e_{k^\sharp+1}$.*

Proof. For the first case, we will reach the contradiction by $\hat{e}_j < \hat{e}_{\hat{k}}$ for some $k < j < K'$ or $e^\sharp_j < e^\flat$ for some $k < j < K'$. For the second case, if $e^\sharp_{k^\sharp} < e^\flat$ and $e_{k^\sharp} < e_{k^\sharp+1}$, we will have $e^\sharp_j < e^\sharp_{k^\sharp}$ for some $k < j < K'$.

□

Based on Lemma 8, it is not difficult to see that the derived solution satisfies the optimality properties in Lemma 6. The following theorem states the optimality of Algorithm Inter-Frame Service Allocation.

Thm. 9: *Algorithm Inter-Frame Service Allocation derives optimal assignments for the general reward maximization on energy harvesting problem with a single application* ($N = 1$).

Proof. Based on Lemma 6 and Lemma 7, an optimal assignment $\vec{e}^\perp$ can only decrease (respectively, increase) energy consumption at time k when $E_C(k, \vec{e}^\perp)$ is $E_{\max}$ (respectively, 0). Suppose that $\vec{e}^\perp$ is different from the assignment $\vec{e}$ derived by Algorithm Inter-Frame Service Allocation. Let κ be the smallest index in which $e^\perp_\kappa$ is different from e_κ. Let m be the index with $e_{m-1} \neq e_m = e_{m+1} = \cdots = e_\kappa$, where e_0 is defined as ∞ for boundary condition. By definition, $E_C(m, \vec{e}^\perp)$ is equal to $E_C(m, \vec{e})$. Let β_1 (respectively, β_2) be the earliest index after κ with $E_C(\beta_1, \vec{e}) = E_{\max}$ (respectively, $E_C(\beta_2, \vec{e}) = 0$). If there does not exist β_1 (respectively, β_2), let β_1 (respectively, β_2) be K.

If $e_\kappa < e^\perp_\kappa$ and $\beta_1 < \beta_2$, we know that $e_i \geq e_j$ for any $m \leq i < j \leq \beta_2$ due to Lemma 6. Moreover, for any $\kappa \leq i \leq \beta_2$, we have $e^\perp_i > e_\kappa$ since it is impossible to have $E_C(i, \vec{e}) = E_{\max}$. Clearly, we reach the conclusion that there is an energy underflow of $\vec{e}^\perp$. The proof is similar for the case that $e_\kappa < e^\perp_\kappa$ and $\beta_1 \geq \beta_2$.

If $e_\kappa > e^\perp_\kappa$ and $\beta_1 \geq \beta_2$, similarly, there will be some energy overflow of $\vec{e}^\perp$ or $\sum_{i=1}^{K} e^\perp_i < E_C(0) - E_\ell + \sum_{i=1}^{K} E_S(i)$, which contradicts the optimality of $\vec{e}^\perp$. If $e_\kappa > e^\perp_\kappa$ and $\beta_1 < \beta_2$, there will be an energy overflow at time β_1 in $\vec{e}^\perp$.

As a result, assignment $\vec{e}$ derived by Algorithm Inter-Frame Service Allocation is optimal for the general reward maximization on energy harvesting problem with a single application.

□

Note that Algorithm 5 does not have to know the exact reward function. The only requirement for guaranteeing the optimality is that the (joint) reward function has to be concave.

4.4.2 Intra-Frame Service Allocation

In contrast to the inter-frame energy allocation, the intra-frame energy allocation requires the precise knowledge of the reward functions $r_n()$ to distribute the energy among all running applications $n = 1, \ldots, N$. Given the energy e_k as the total energy which can be consumed in the k-th frame, we can formally write the intra-frame energy assignment problem as follows:

$$\text{maximize } \sum_{n=1}^{N} r_n(\epsilon_{k,n}) \text{ subject to:} \tag{4.1}$$

$$\begin{aligned} \sum_{n=1}^{N} \epsilon_{k,n} &= e_k \\ c_m(\vec{\epsilon}_k) &= 0 \qquad \forall 1 \leq m \leq M \\ \epsilon_{k,n} &\geq 0 \qquad \forall 1 \leq n \leq N \end{aligned}$$

In other words, we want to maximize the accumulated reward subject to the energy budget e_k and the constraints $c_m(\vec{\epsilon}_k)$ from possible rate-based graphs. We denote the optimal objective value of the above optimization problem in Equation (4.1) as function $R(e_k) = \max\{\sum_{n=1}^{N} r_n(\epsilon_{k,n})\}$. The following theorem establishes the concavity of $R(e_k)$ over the energy consumption e_k.

Thm. 10: *If r(x,y) is concave in (x,y) and C is a convex nonempty set, then* $R(x) = \max_{y \in C} r(x, y)$ *is concave in x, provided* $R(x) > -\infty$ *for some x.*

Proof. A proof can be found in standard text books on convex optimization, e.g., in [BV04, Chapter 3].

□

As a result, based on the concavity of $R(e_k)$ in Theorem 10 and the optimality of Algorithm Inter-Frame Service Allocation in Theorem 9, we can divide the general reward maximization on energy harvesting problem into two subproblems, namely finding inter-frame and intra-frame energy assignments. We can now determine intra-frame service assignments by using standard techniques from constrained optimization. By

Algorithm 6 (Intra-Frame Service Allocation)

Input: e_k, $r_n()$ for $n = 1, \ldots, N$, $c_m(\vec{\epsilon_k})$ for $m = 1, \ldots, M$;

Output: a feasible assignment $\vec{\epsilon_k}$ of energy consumption for the N applications in frame k;

1: let $L(\epsilon_{k,n}, \ldots, \epsilon_{k,N}, \lambda_0, \lambda_1, \ldots, \lambda_M)$ be $\sum_{n=1}^{N} r_n(\epsilon_{k,n}) - \lambda_0 \cdot \left(\sum_{n=1}^{N} \epsilon_{k,n} - e_k \right) - \sum_{m=1}^{M} \lambda_m \cdot c_m(\vec{\epsilon_k})$;

2: solve $\frac{\partial L()}{\partial \epsilon_{k,1}} = \cdots = \frac{\partial L()}{\partial \epsilon_{k,N}} = 0$, $\forall 1 \leq n \leq N$ under the energy constraints $\sum_{n=1}^{N} \epsilon_{k,n} = e_k$ and $c_m(\vec{\epsilon_k}) = 0$;

3: return $\vec{\epsilon_k}$ as the solution;

applying the Lagrange Multiplier method, we can solve the problem in Equation (4.1) optimally, as shown in Algorithm 6.

Certainly, solving the equations in Step 2 is a critical operation in Algorithm 6. As already indicated in the section overview and depicted in Figure 36, one would preferably determine explicit functions $f_1(), \ldots, f_N()$ to compute the energy consumptions of the applications $\epsilon_{k,n} = f_n(e_k), \forall 1 \leq n \leq N$ efficiently in the online case. In Section 4.6, such techniques for a practical implementation of the algorithm will be discussed.

Due to the concavity of the reward functions, the solution found by using the Lagrange Multipliers $\lambda_0, \lambda_1, \ldots, \lambda_M$ is guaranteed to be the *global* optimum for the intra-frame energy assignment problem. As a result, it is not difficult to see the optimality of the derived solutions, which is a direct consequence of Theorems 9 and 10.

Cor. 1: *The energy assignment obtained by successive application of Algorithm 5 and 6 constitutes an optimal assignment for the general reward maximization on energy harvesting problem.*

To demonstrate how Algorithm 6 works, we use the following example for illustration. Suppose that an embedded system has to execute two independent applications v_1 and v_2 with the energy budget e_k in a certain frame. The respective reward functions are given by $r_1(\epsilon_{k,1}) = \ln \epsilon_{k,1}$ and $r_2(\epsilon_{k,2}) = \sqrt{\epsilon_{k,2}}$. By setting the partial derivatives of the Lagrangian $L()$ equal to 0, we get $\frac{\partial r(\epsilon_{k,1})}{\partial \epsilon_{k,1}} = \frac{\partial r(\epsilon_{k,2})}{\partial \epsilon_{k,2}} = \lambda_0$ under the energy constraint $\epsilon_{k,1} + \epsilon_{k,2} = e_k$. Thus, we have to solve

$$\frac{1}{\epsilon_{k,1}} = \frac{1}{2\sqrt{\epsilon_{k,2}}} = \lambda_0$$

$$\epsilon_{k,1} + \epsilon_{k,2} = e_k.$$

In this case, we know that $\epsilon_{k,2} = \frac{\epsilon_{k,1}^2}{4}$. Therefore, $\frac{\epsilon_{k,1}^2}{4} + \epsilon_{k,1} - e_k = 0$, in which $\epsilon_{k,1} = -2 + 2\sqrt{1 + e_k}$. If we continue on the example from the previous subsection in Figure 38, the energy consumptions in the first frame are, e.g., $\epsilon_{1,1} = -2 + 2\sqrt{4.5}$ and $\epsilon_{1,2} = 5.5 - 2\sqrt{4.5}$. In Figure 39, the normalized functions $\frac{\epsilon_{k,1}}{e_k} = \frac{2\sqrt{e_k+1}-2}{e_k}$ and $\frac{\epsilon_{k,2}}{e_k} = 1 - \frac{\epsilon_{k,1}}{e_k}$ are displayed over the energy budget e_k. For values $e_k < 8$, application v_1 receives the larger percentage of e_k, whereas for $e_k > 8$, it is advantageous to assign more energy to application v_2. At $e_k = 8$, the energy budget e_k has to be shared equally among both applications.

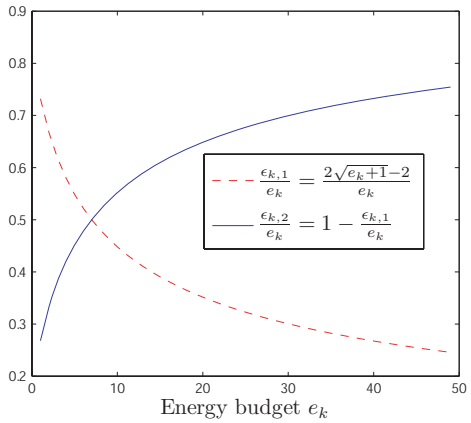


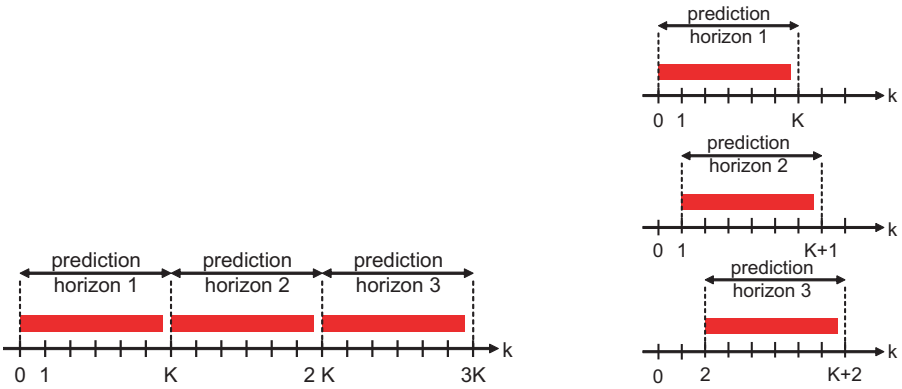
Fig. 39: Intra-frame energy assignment for the example with $r_1(\epsilon_{k,1}) = \ln \epsilon_{k,1}$ and $r_2(\epsilon_{k,2}) = \sqrt{\epsilon_{k,2}}$.

4.5 Design Considerations

This section provides some remarks for designing embedded systems with energy harvesting devices using the techniques described in this chapter.

4.5.1 Energy Prediction Techniques

The energy prediction algorithm should depend on the type of the energy source and the system environment. In general, standard techniques known from automatic control and signal processing can be applied here. For photovoltaic cells harvesting solar energy, several energy prediction algorithms have been presented in the literature [KPS04, MTBB07, MTBB08]. Of course, one could even use external information from the weather forecast to predict solar radiation, but in many cases, simple techniques turn out to be sufficient. By keeping a his-



(a) Sliding horizon operation

(b) Sliding frame operation

Fig. 40: Different sliding operation for Algorithm 5 (Inter-Frame Service Allocation).

tory of harvested energies of the past, simple algorithms based on exponential decay turn out to give quite accurate predictions [KPS04]. In Section 3.6.1.1, a worst-case energy prediction algorithm is presented which guarantees sustainable operation for solar powered devices. Specifically, the proposed algorithm in accounts for the fact that short-term energy prediction can be considerably lower than average expectations.

In any case, some kind of energy prediction algorithm is necessary for serious performance optimization of energy harvesting embedded systems. In this chapter, we study a basic problem by using a perfect energy prediction and gain fundamental insight in optimization problems underlying energy harvesting systems. Certainly, a system receiving imprecise or wrong energy prediction information will experience a performance degradation. However, an investigation of the impact of prediction mistakes is beyond the scope of this chapter.

4.5.2 Sliding Horizon Operation

For the online usage of Algorithm Inter-Frame Service Allocation, one has two principal alternatives, as illustrated in Figure 40, *sliding horizon* or *sliding frame* operations. While the sliding horizon operation repeatedly predicts and optimizes the energy assignments for independent horizons, the sliding frame operation solves the problem for overlapping horizons for every frame. The latter operation is also known as receding horizon control in the literature [ML99]. As energy prediction is assumed perfect for the study in this chapter, the sliding horizon operation is adopted and will be used for presenting the simulation results in Section 4.6. Certainly, if there are prediction mistakes, one would preferably choose the sliding frame operation since prediction mistakes can be counteracted more efficiently. On the other hand, the sliding frame operation imposes

a higher computational load for the embedded system, since Algorithm 5 is executed K times more often then in the sliding horizon operation.

4.5.3 The Minimum Energy Storage Capacity $E_{\max,\min}$

To design the energy supply of an embedded system, it is important to estimate how to dimension the energy storage device. Given an initial energy $E_C(0)$, an energy source $E_S(k), 1 \leq k \leq K$ and a final energy constraint E_ℓ, we are interested in the minimum storage capacity which is needed to achieve the maximum possible reward. We denote

$$E_{\max,\min} = \min_{E_{\max}} \{r(\vec{e}^*, E_{\max}) = r(\vec{e}^*, +\infty)\}$$

the minimum capacity $E_{\max}$ for which the optimal reward equals the reward for an unconstrained system which $E_{\max} = +\infty$. For a single horizon, we can now determine

$$E_{\max,\min} = \max_{k=1,2,\ldots,K} \{E_C(0) + \sum_{j=1}^{k} (E_S(j) - e_j^*)\},$$

where $\vec{e}^*$ is an assignment computed by Algorithm 4 for specified $E_S(k)$ for $k = 1, 2, \ldots, K$, $E_C(0)$ and E_ℓ. For sliding horizon operation, this calculation has to be repeated for representative data samples $E_S(k)$ for the energy source. This can be done by using, e.g., sufficiently large traces of solar energy which have been recorded beforehand. By choosing the maximum of all capacities $E_{\max,\min}$, it is possible to determine the minimum capacity $E_{\max}$ to optimally exploit a given environmental source.

4.5.4 Energy Buffering, Limited Energy Consumption and Discrete Service Levels

As already mentioned, a frame may last several hours in physical time. If now the energy storage is full, i.e., $E_C(k-1) = E_{\max}$ and $e_k = E_S(k)$ for some k, we may still have an energy overflow if $E_S(k)$ arrives *before* it is consumed by e_k. This conflict can be resolved by assuming the existence of small energy buffers (capacitors or supercapacitors) or a lower capacity $E'_{\max}$. In a similar way, problems can be resolved for the underflow problem.

In some systems, the applications might have the requirement to consume at least some amount of energy consumption in a frame, and there might also be a constraint on the maximum energy consumption in a frame since there is no improvement in quality/reward for more energy consumption. We can revise the solution $\vec{e}$ derived by Algorithm Inter-Frame Service Allocation by setting the energy consumption of the k-th

frame as the maximum energy consumption if e_k is greater than the maximum energy consumption constraint. The proofs in Section 4.4.1.3 can be extended to prove the optimality of the revised solution. If there exists e_k which is less than the minimum energy consumption constraint, it is not difficult to see that the input does not admit a feasible assignment.

In some embedded systems, one may have to choose between a limited number of discrete service levels with corresponding energy consumptions $\epsilon_{k,n}$. We found that a straightforward extension of the work presented in this chapter is not trivial for these cases. Specifically, it seems not trivial to derive optimal service level assignments since the nature of the optimization problem changes fundamentally if one moves from continuous to discrete energy consumptions $\epsilon_{k,n}$. Nevertheless, approximations based on the algorithms presented in this chapter may provide good (but suboptimal) results. This holds in particular if the number of service levels is getting large.

4.6 Simulative Evaluation

4.6.1 Simulation Environment and Setup

For the experiments, we use long-term measurements of solar light intensity recorded during 5 years from a photovoltaic plant at Mont Soleil, Switzerland [sol07]. The data measured in $\left[\frac{W}{m^2}\right]$ serves as harvested energy $E_S()$. Of course, to simulate a concrete system, one would have to *scale the measured power profile* with the size, number and efficiency of the actually used solar panels. The data is sampled every 5 minutes, so we have a maximum of 288 samples per day.

Since solar energy shows a similar pattern every 24 hours, multiples of a day are reasonable choices for the prediction horizon. The number of frames per day does not affect the results too much. The presented results in this section are conducted by setting 16 frames per day, which means a frame lasts for 1.5 hours. The results for different numbers of frames per day are quite similar. When investigating different values for the number of frames of the prediction horizon $K \in \{16, 32, 48, \ldots\}$, we use the shorter notation $K \in \{1d, 2d, 3d, \ldots\}$ but keep in mind that we have 16 frames per day. At the end of the prediction horizon, we want the remaining energy $E_C(K)$ to be at least equal to the initial energy $E_C(0)$, i.e., $E_\ell = E_C(0)$.

Concerning the online complexity of Algorithm 5 we experienced that even if the number of frames K is growing, the average number of computation steps is by far less than the worst-case time complexity $O(K^2)$. The worst-case scenario only happens if Algorithm 5 has to prevent an

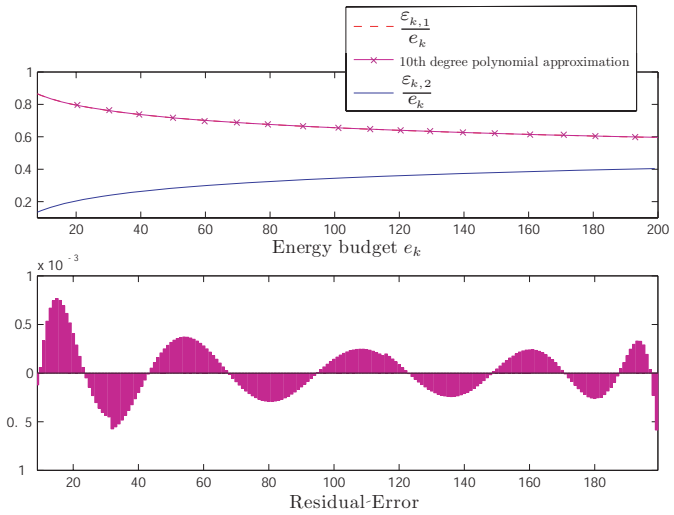


Fig. 41: Approximated inter-frame energy assignment using polynomial functions.

energy overflow or underflow in every frame k. However, if the energy storage capacity $E_{\max}$ is chosen close to capacity $E_{\max,\min}$ as presented in Section 4.5.3, the average number of calls of procedure SUBSEG and hence the online computation demand of Algorithm 5 remains low.

4.6.2 Efficient Implementation of the Intra-Frame Service Allocation

As illustrated in Figure 36, the intra-frame energy assignments can be computed online by Algorithm 6. However, since the reward functions $r_1(), \ldots, r_N()$ and the rate constraints $c_1(), \ldots, c_M()$ are usually given at design time and remain unchanged during the operation of the embedded system, explicit functions $f_1(), \ldots, f_N()$ can be pre-computed offline and evaluated online instead of running Algorithm 6. For this algorithm, we already mentioned that solving the set of equations in Step 2 may not give an explicit solution. In such situations, numerical root-finding algorithms like Newton's method can be applied to find an approximation for the energy assignment $\epsilon_{k,n} \forall 1 \leq n \leq N$. Newton's method is known to have a fast, quadratic convergence ratio and the precision of the result is only limited by the finite precision of the machine which is used to carry out the arithmetic. Like that, Algorithm 6 is performed offline for a large number of energy budgets e_k and approximation functions $\epsilon_{k,n} = f_n(e_k), \forall 1 \leq n \leq N$ are computed.

We investigated systems consisting of several applications and discuss a representative problem next. Assume we have two independent applications v_1 and v_2 with the reward functions $r_1(\epsilon_{k,1}) = \ln \epsilon_{k,1}$ and

$r_2(\epsilon_{k,2}) = \sqrt{\sqrt{\epsilon_{k,2}} + 2}$. To our best knowledge, there exists no explicit function to distribute the budget e_k in a certain frame. Using Newton's method, we approximated $\epsilon_{k,1}$ and $\epsilon_{k,2}$ for integer values $e_k \in \{1, 2, \ldots, 200\}$. Next, we determine the coefficients of, e.g. , a 10th degree polynomial by least-square fitting. In Figure 41, the top plot shows the approximated values of $\frac{\epsilon_{k,1}}{e_k} = 3.7 \cdot 10^{-21} \cdot e_k^{10} - 10^{-18} \cdot e_k^9 + 1.9 \cdot 10^{-15} \cdot e_k^8 - 4.9 \cdot 10^{-13} \cdot e_k^7 + 810^{-11} \cdot e_k^6 - 8.6 \cdot 10^{-9} \cdot e_k^5 + 6 \cdot 10^{-7} \cdot e_k^4 - 2.8 \cdot 10^{-5} \cdot e_k^3 + 0.00083 \cdot e_k^2 - 0.018 \cdot e_k + 0.97$. In this plot, it is not possible to distinguish the approximated from the real values of $\frac{\epsilon_{k,1}}{e_k}$. The approximation mistake is expressed in terms of residual error and can be negligible in practice, see the bottom plot in Figure 41. The coefficients of the approximation functions have to be stored and evaluated for every application in the online case, causing neglectable overhead in terms of computation and memory requirement. Note that these approximations can be performed for miscellaneous computation-based or rate-based application models with arbitrary reward functions, as presented in Section 4.3.3.

4.6.3 Choosing Sufficient Parameters K and $E_{\max}$

A fundamental question one might ask when designing a system in a given environment is: How many days should the horizon span to obtain reasonable rewards? For this purpose, we simulate Algorithm 5 (Inter-Frame Service Allocation) for different parameters $K \in \{1d, 2d, 3d, 5d, 10d, 15d, 30d, 70d, 105d, 210d\}$. To obtain a total simulated time of 210 days for each experiment, $\{210, 105, 70, 42, 21, 14, 7, 3, 2, 1\}$ horizons are executed in the sliding horizon operation, respectively. The resulting energy assignments $\vec{e}$ are evaluated by applying the joint reward function $R(e_k) = \ln(0.01 + \frac{e_k}{1000})$ which assigns negative rewards (i.e., penalties) to energies $e_k < 990$, where $R(e_k)$ is the summation of reward functions $r_n(\epsilon_{k,n})$ of the N applications under energy constraint e_k. In particular, setting the services to 0 is punished with an penalty of $\ln(0.01) \approx -4.61$. For each experiment, we calculate the accumulated reward for 210 days.

As a matter of fact, the accumulated reward depends both on the number d of days of the prediction horizon *and* the energy storage capacity $E_{\max}$. In Figure 42 we see that the accumulated reward increases quickly with the parameters d and $E_{\max}$. The minimum energy capacity $E_{\max}$ required to optimally exploit this energy source is $E_{\max,\min} = 759450$. Using this value for the battery, a horizon of $d = 15$ days is sufficient to achieve 93.4% of the maximum possible reward (i.e., the reward for $d, E_{\max} = \infty$). For this particular reward function, however, also smaller capacities $E_{\max}$ are possible to achieve a similar reward.

In Figure 43, the accumulated rewards were normalized by the reward

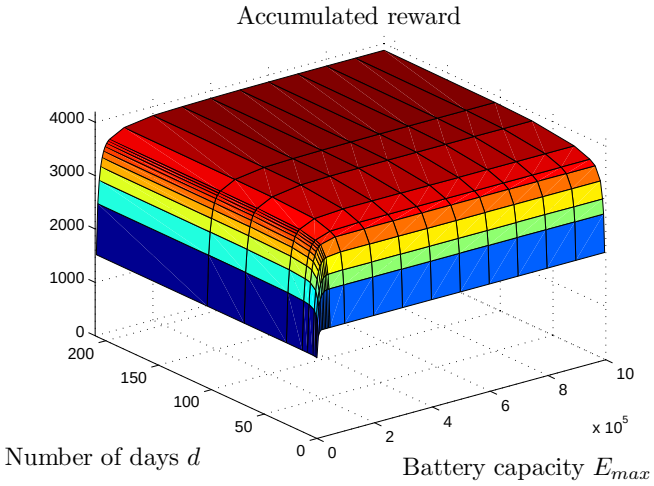


Fig. 42: Accumulated reward $\sum_{k=1}^{K}\sum_{n=1}^{N} r_n(\epsilon_{k,n})$ for 210 days for the optimal assignment, $E_C(0) = E_\ell = 3000$.

obtained by the experiment with the longest horizon, namely 210 days. If one chooses a smaller $E_{\max}$, it turns out that the reward converges faster towards its longterm average with increasing d. For a horizon of $d = 15$ days, a capacity of $E_{\max} = 500000$ will result in a reward of 93,7% of the reward for the same capacity with $d = 210$ days. For smaller capacities $E_{\max} = 24000$ and 4000, the ratio increases to 97,7 % and 99.9 %, respectively. The reason for this behaviour is that Algorithm 5 is getting more and more nearsighted with smaller capacity $E_{\max}$: Due to the capacity constraint, local maxima of $\vec{e}$ are computed to avoid energy overflows. Hence, for small energy storage capacities $E_{\max}$, the total reward can hardly be improved by increasing the prediction horizon d.

In summary, using our techniques, one can easily determine or tradeoff suitable parameters K and $E_{\max}$ for a given environment. Given a representative trace of the harvested energy, the reward functions of the applications as well as possible rate constraints, one can easily determine suitable parameters for the energy storage $E_{\max}$ and the length of the prediction horizon K. Since the reward converges quickly for both parameters, one can simply select values of $E_{\max}$ and K which result in a reward only a few percent below the optimal reward for $E_{\max} = E_{\max,\min}$ and $K = +\infty$. For our example, the energy storage should be large enough to store the incoming energy E_S during approximately 5 days and the prediction horizon should be at least 2 weeks to achieve a reward close to the optimal one.

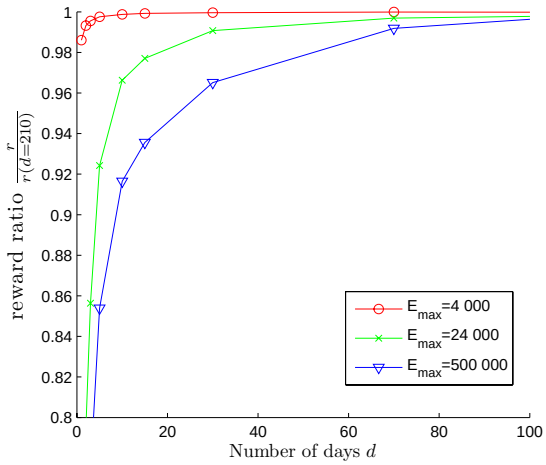


Fig. 43: Convergence of the normalized reward over the number of days d, $E_C(0) = E_\ell = 3000$.

4.6.4 Comparison to an Adversary Algorithm

Since there are no other algorithms available for the general reward maximization on energy harvesting problem, we designed an adversary algorithm to Algorithm Inter-Frame Service Allocation which can be found in Appendix A (Algorithm 7). What makes finding adversary algorithms tricky is that one has to find algorithms which are *feasible* and *competitive* at the same time. To this end, the constructed adversary algorithm constitutes the smartest solution an engineer would probably implement on, e.g, a sensor node not being aware of the techniques described in this chapter. Algorithm 7 averages the energy consumptions for the remaining frames. Only if energy overflows or underflows happen, recalculations of the energies are performed. The time complexity of the adversary algorithm is the same as that of Algorithm 5.

Figure 44 displays a comparison of the assignments generated by the adversary algorithm and Algorithm Inter-Frame Service Allocation for $d = 5$ days. Both assignments start with an initial energy $E_C(0) = 3000$, the energy storage capacity is $E_{\max} = 20000$. Obviously, the optimal assignment $\vec{e}$ manages to balance the energy consumption much better than the assignment $\vec{e}^{\,a}$ derived from the adversary algorithm. The latter has to suspend the service completely during the first four nights, which is clearly an unacceptable behaviour. Around frame 60, a burst of energy E_S is forcing $\vec{e}^{\,a}$ to increase the service to 10000, whereas $\vec{e}$ shows only a moderate increase to 6000. Using the same reward function as in Section 4.6.3, the total reward for assignment $\vec{e}$ amounts to 34.6; assignment $\vec{e}^{\,a}$ achieves a negative total reward of -57.1.

For Figure 45, we repeated the experiment for 20 horizons in sliding

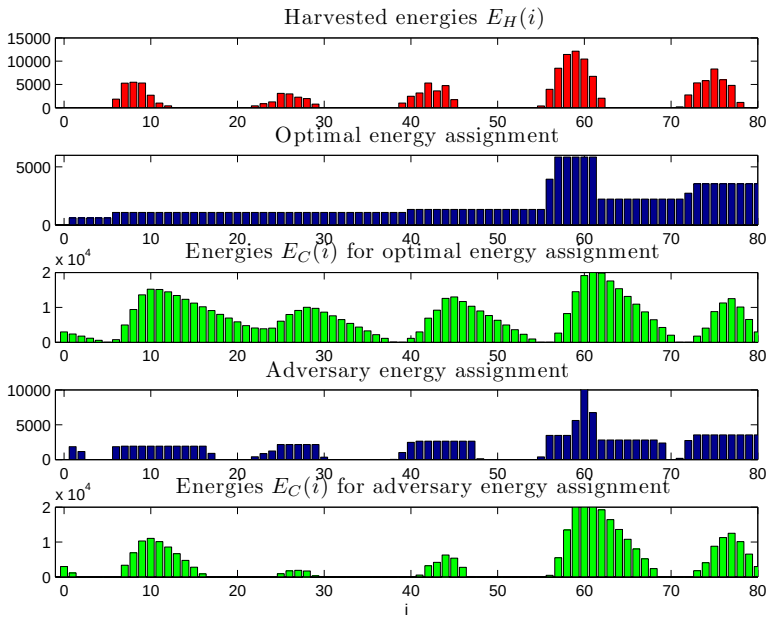


Fig. 44: Comparison of assignments $\vec{e}$ and $\vec{e}^{\,\prime\prime}$ for $E_C(0) = E_\ell = 3000, E_{\max} = 20000, d = 5$.

horizon operation, i.e. for 100 consecutive days in total. The average reward during this time was 68.4 for the optimal and 12.2 for the adversary algorithm. So also in terms of average reward, the optimal assignment outperforms $\vec{e}^{\,\prime\prime}$ significantly.

The primary results of the comparison with the adversary algorithm are as follows: Firstly, we demonstrate that energy underflows and overflows happen also for real energy sources (in our case we apply a typical trace of solar energy) and not only for the constructed examples in Section 4.4.1. And secondly, we show that significant performance improvements can be realized using our algorithms compared to naive approaches.

4.7 Chapter Summary

This chapter copes with embedded systems equipped with energy harvesting devices, such as solar panels. By applying prediction techniques, we explore how to globally optimize the system performance of diverse applications in terms of system rewards. The reward functions of applications are assumed to be concave and increasing with respect to the energy consumption. For a given specified prediction horizon composed of multiple frames, we develop a two-stage mechanism for energy assignments. First, we determine how much energy can be consumed in

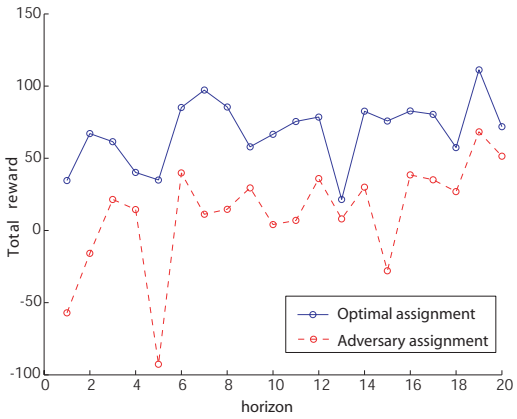


Fig. 45: Rewards of assignments $\vec{e}$ and $\vec{e}^{\prime\prime}$ for 20 horizons, $E_C(0) = E_\ell = 3000, E_{\max} = 20000, d = 5$.

one frame without violating the energy capacity constraint. Then, an algorithm based on the Lagrange Multiplier method is applied to distribute the energy budget in one frame to the applications in the system. The effectiveness of the proposed algorithms is supported by simulations based on long-term measurements of the power generated by real solar cells. Moreover, we also demonstrate how to dimension the embedded system, e.g., the battery capacity and elaborate on implementation details for efficiency issues.

5

Conclusions

5.1 Main Results

This thesis addresses power management in energy harvesting embedded systems. As example scenario, we focus on wireless sensor nodes which are powered by solar cells. We demonstrate that classical power management solutions have to be reconceived and/or new problems arise if perpetual operation of the system is required. In particular, we provide a set of algorithms and methods for different application scenarios, including real-time scheduling, application rate control as well as reward maximization. The purpose of the latter two approaches is to decide which and how many tasks are executed in a long-term perspective. Based on an estimation of the energy harvested in the future, long-term decisions on the use of the available energy are made.

On a task level, we provide real-time scheduling algorithms to assign energy to upcoming tasks in a short-term perspective. By taking into account both available time and energy, an optimal task ordering is computed to avoid deadline violations.

While most conventional power management solutions aim to save energy subject to given performance constraints, performance constraints are not given a priori for the energy harvesting systems discussed in this thesis. Rather, the performance is adapted in a best effort manner according to the availability of environmental energy. Goal is to optimize the performance of the application subject to given energy constraints. Compared to state of the art approaches, our methods achieve a better performance, or achieve the same performance requiring, e.g., smaller

solar cells and batteries. Furthermore, we show how to dimension important system parameters like the minimum battery capacity or a sufficient prediction horizon.

Our theoretical results are supported by simulations using long-term measurements of solar energy in an outdoor environment. In this way, we are able to simulate the behaviour of wireless sensor nodes for time scales which are usually desired (i.e. several months up to several years). Furthermore, we measured the implementation overhead of some algorithms on real sensor nodes to demonstrate the practical relevance of our approach.

5.2 Future Perspectives

The application scenarios investigated in this thesis give fundamental insight in the challenges of energy harvesting systems. Beyond these basic scenarios, however, we identified several topics which deserve further investigation. It follows a short discussion of potential future work.

Combination of Approaches

As already indicated, the single approaches in this thesis could be combined as follows: As illustrated in Figure 46, there may be two hierarchically structured schedulers which control the application. In a first step, an Application Rate Controller or a Service Level Allocator decides which and how many tasks are executed on the long run. In a second step, a Real-Time Task Scheduler decides about the short-term task ordering. To realize this combination of approaches, however, adaptations of the methods in this thesis become necessary.

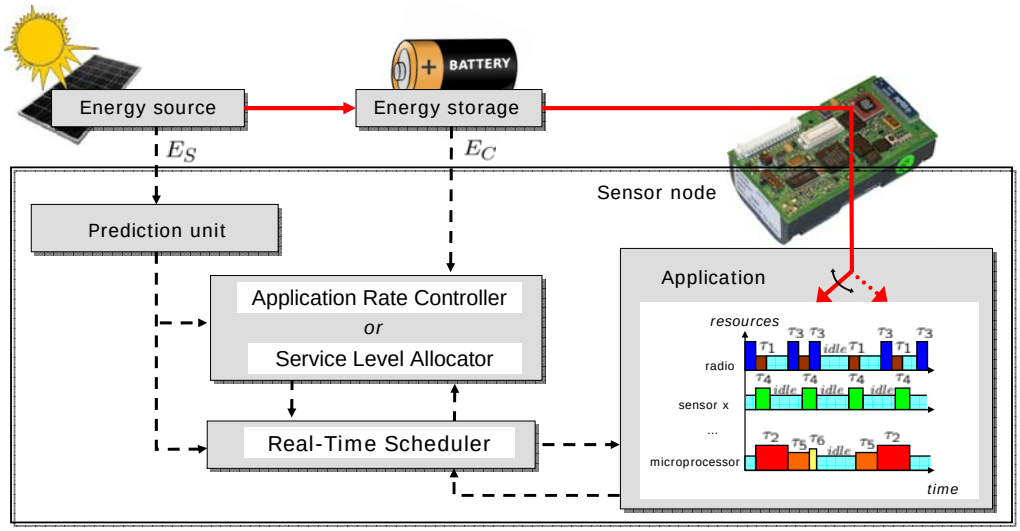


Fig. 46: A Possible Combination of the Approaches presented in Chapters 2, 3 and 4.

Energy Prediction Uncertainty

Provided that precise information about the future energy generation is available, most methods in this thesis guarantee an *optimal* system performance. To account for prediction mistakes, both a worst-case energy prediction algorithm and an approximate control design have been presented in Chapter 3. We pointed out that overly precise optimization of the application parameters turns out to be useless if major prediction errors occur. However, for the different approaches presented in Chapters 2, 3 and 4, it remains unclear to what extent prediction mistakes actually degrade the performance of the system.

Hardware Characteristics

Practical experiments revealed that charging a storage device like e.g. a supercapacitor or a battery with a solar cell remains a highly non-linear process. As a consequence, the amount of harvested energy depends on factors like, e.g., the current charging level, the charging history, the ambient temperature and the impinging light intensity.

What is clearly missing is a performance optimization approach that takes into account the characteristics of the underlying hardware. In other words, the optimization software running on the sensor node not only has to estimate how much *solar energy* will be available in the future but also to what extent usable *electrical energy* can be generated and consumed by the sensor node. Non-linearities of the conversion and storage process are expected to play a dominating role in the overall performance optimization. Hence, measurements of a prototype photovoltaic energy scavenger would shad light on how solar energy can be used effectively.

Distributed Application Control

All scenarios discussed in this thesis investigate optimization of the temporal performance of individual embedded systems. A potential extension of our work could deal with clusters of embedded systems running a distributed application. Therefore, the spatial variations in environmental energy between different nodes must be exploited. As in multihop wireless sensor networks, nodes have joint objective functions (e.g. joint sensing areas), or may have joint constraints (e.g. wireless communication protocols).

Discrete Application Parameters

The work presented in this thesis considers the adaptation of continuous application parameters to maximize the utility of energy harvesting

systems. These continuous parameters represent, e.g., the instantiation rates of tasks, the amount of computation or simply the duty cycle of a sensor node. In practice, however, one sometimes has the choice between a finite number of possible modes of operation. For this purpose, a more realistic application model consisting of a finite set of discrete modes is of importance. While some methods in this thesis can still approximately find this parameters (e.g. by rounding to the next level), an optimal solution is not any longer guaranteed. For the application scenario reward maximization in Chapter 4, we designed efficient algorithms for systems with discrete service levels. The respective work is described in [MCT09].

Bibliography

[ABM+05] Yasser Ammar, Aurlien Buhrig, Marcin Marzencki, Benot Charlot, Skandar Basrour, Karine Matou, and Marc Renaudin. Wireless sensor network node with asynchronous architecture and vibration harvesting micro power generator. In *sOcEUSAI '05: Proceedings of the 2005 joint conference on Smart objects and ambient intelligence*, pages 287–292, New York, NY, USA, 2005. ACM Press.

[AM01] Andre Allavena and Daniel Mosse. Scheduling of frame-based embedded systems with rechargeable batteries. In *Workshop on Power Management for Real-Time and Embedded Systems (in conjunction with RTAS 2001)*, 2001.

[AMMA99] H. Aydin, R. Melhem, D. Mosse, and P.M. Alvarez. Optimal reward-based scheduling for periodic real-time tasks. In *Proceedings of the 20th IEEE Real-Time Systems Symposium (RTSS'99)*, pages 79–89, 1999.

[Bar03] Sanjoy K. Baruah. Dynamic- and static-priority scheduling of recurring real-time tasks. *Real-Time Systems*, 24(1):93–128, 2003.

[BBBM01] F. Borrelli, M. Baotic, A. Bemporad, and M. Morari. Efficient On-Line Computation of Constrained Optimal Control. In *IEEE Conference on Decision and Control*, pages 1187–1192, Orlando, Florida, December 2001.

[BBM00] A. Bemporad, F. Borrelli, and M. Morari. Piecewise linear optimal controllers for hybrid systems. In *American Control Conference*, pages 1190–1194, Chicago, USA, June 2000.

[BBM02] Alberto Bemporad, Francesco Borrelli, and Manfred Morari. Model Predictive Control Based on Linear Programming - The Explicit Solution. *IEEE Transactions on Automatic Control*, 47(12):1974–1985, December 2002.

[BBM03] F. Borrelli, A. Bemporad, and M. Morari. A Geometric Algorithm for Multi-Parametric Linear Programming. *Journal of Optimization Theory and Applications*, 118(3):515–540, September 2003.

[BBMT08] Davide Brunelli, Luca Benini, Clemens Moser, and Lothar Thiele. An Efficient Solar Energy Harvester for Wireless Sensor Nodes. In *Design, Automation and Test in Europe (DATE 08)*, Munich, Germany, March 10-14 2008.

[BDH+04] J. Beutel, M. Dyer, M. Hinz, L. Meier, and M. Ringwald. Next-generation prototyping of sensor networks. In *Proc. 2nd ACM Conf. Embedded Networked Sensor Systems (SenSys 2004)*, pages 291–292. ACM Press, New York, November 2004.

[BMDP02] A. Bemporad, M. Morari, V. Dua, and E.N. Pistikopoulos. The Explicit Linear Quadratic Regulator for Constrained Systems. *Automatica*, 38(1):3–20, January 2002.

[BR03] Philippe Barrade and Alfred Rufer. Current capability and power density of supercapacitors: Considerations on energy efficiency. In *European Conference on Power Electronics and Applications (EPE), 2003*, Toulouse, France, September 2-4 2003.

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, March 2004.

[CK05] Jian-Jia Chen and Tei-Wei Kuo. Voltage-scaling scheduling for periodic real-time tasks in reward maximization. In *the 26th IEEE Real-Time Systems Symposium (RTSS)*, pages 345–355, 2005.

[CK07a] Jian-Jia Chen and Chin-Fu Kuo. Energy-efficient scheduling for real-time systems on dynamic voltage scaling (DVS) platforms. In *RTCSA*, pages 28–38, 2007.

[CK07b] Jian-Jia Chen and Tei-Wei Kuo. Procrastination determination for periodic real-time tasks in leakage-aware dynamic voltage scaling systems. In *ICCAD*, pages 289–294, 2007.

[CKY04] Jian-Jia Chen, Tei-Wei Kuo, and Chia-Lin Yang. Profit-driven uniprocessor scheduling with energy and timing constraints. In *ACM Symposium on Applied Computing*, pages 834–840, 2004.

[Cor06] Moteiv Corporation. Tmote sky - ultra low power ieee 802.15.4 compliant wireless sensor module, datasheet. June, 2006.

[DKT96] Jayanta K. Dey, James F. Kurose, and Donald F. Towsley. On-line scheduling policies for a class of IRIS (increasing reward with increasing service) real-time tasks. *IEEE Transactions on Computers*, 45(7):802–813, 1996.

[EC07] T. Esram and P.L. Chapman. Comparison of photovoltaic array maximum power point tracking techniques. *Energy Conversion, IEEE Transaction on*, 2:439–339, 2007.

[Fil04] Carlo Filippi. An algorithm for approximate multiparametric linear programming. *Journal of Optimization Theory and Applications*, 120(1):73–95(23), January 2004.

[Gal95] Thomas Gal. Postoptimal Analyzes, Parametric Programming, and Related Topics. 2nd ed., Berlin, Germany,de Gruyter 1995.

[HHKK04] Jason Hill, Mike Horton, Ralph Kling, and Lakshman Krishnamurthy. The platforms enabling wireless sensor networks. *Commun. ACM*, 47(6):41–46, 2004.

[HKF+05] Jason Hsu, Aman Kansal, Jonathan Friedman, Vijay Raghunathan, and Mani Srivastava. Energy harvesting support for sensor networks. In *SPOTS track at IPSN 2005*, 2005.

[HZK+06] Jason Hsu, Sadaf Zahedi, Aman Kansal, Mani Srivastava, and Vijay Raghunathan. Adaptive duty cycling for energy harvesting systems. In *ISLPED '06: Proceedings of the 2006 international symposium on Low power electronics and design*, pages 180–185, New York, NY, USA, 2006. ACM Press.

[JBM07] C.N. Jones, M. Baric, and M. Morari. Multiparametric Linear Programming with Applications to Control. *European Journal of Control*, 13(2-3):152–170, March 2007.

[JPC05a] Xiaofan Jiang, Joseph Polastre, and David E. Culler. Perpetual environmentally powered sensor networks. In *Proceedings of the Fourth International Symposium on Information Processing in Sensor Networks, IPSN 2005*, pages 463–468, UCLA, Los Angeles, California, USA, April 25-27 2005.

[JPC05b] Xiaofan Jiang, Joseph Polastre, and David E. Culler. Perpetual environmentally powered sensor networks. In *Proceedings of the Fourth International Symposium on Information Processing in Sensor Networks, IPSN 2005*, pages 463–468, UCLA, Los Angeles, California, USA, April 25-27 2005.

[JPG04] R. Jejurikar, C. Pereira, and R. Gupta. Leakage aware dynamic voltage scaling for real-time embedded systems. In *Proceedings of the Design Automation Conference*, pages 275–280, 2004.

[KC00] R. Koetz and M. Carlen. Principles and applications of electrochemical capacitors. In *Electrochimica Acta 45*, pages 2483–2498. Elsevier Science Ltd., 2000.

[KGB04] M. Kvasnica, P. Grieder, and M. Baotić. Multi-Parametric Toolbox (MPT), 2004.

[KHZS07] Aman Kansal, Jason Hsu, Sadaf Zahedi, and Mani B. Srivastava. Power management in energy harvesting sensor networks. *Transactions on Embedded Computing Systems*, 6(4):32, 2007.

[KPS04] Aman Kansal, Dunny Potter, and Mani B. Srivastava. Performance aware tasking for environmentally powered sensor networks. In *Proceedings of the International Conference on Measurements and Modeling of Computer Systems, SIGMETRICS 2004*, pages 223–234, New York, NY, USA, June 10-14 2004. ACM Press.

[LLS+91] J. W.-S. Liu, K.-J. Lin, W.-K. Shih, A. C.-S. Yu, C. Chung, J. Yao, and W. Zhao. Algorithms for scheduling imprecise computations. *IEEE Computer*, 24(5):58–68, May 1991.

[LSR05] Longbi Lin, Ness B. Shroff, and R.Srikant. Asymptotically optimal power-aware routing for multihop wireless networks with renewable energy sources. In *Proceedings of IEEE INFOCOM 2005*, pages 1262 – 1272, Miami, USA, March 13-17 2005.

[MB06] A. Magnani and S. Boyd. Convex piecewise-linear fitting. In *Optimization and Engineering (submitted)*, `http://www.stanford.edu/~boyd/reports/cvx_pwl_fit.pdf`, April 2006.

[MBTB06] Clemens Moser, Davide Brunelli, Lothar Thiele, and Luca Benini. Lazy scheduling for energy-harvesting sensor nodes.

In *Fifth Working Conference on Distributed and Parallel Embedded Systems, DIPES 2006*, pages 125–134, Braga, Portugal, October 11-13 2006.

[MCT08] Clemens Moser, Jian-Jia Chen, and Lothar Thiele. Reward maximization for embedded systems with renewable energies. *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA08)*, 0:247–256, 2008.

[MCT09] Clemens Moser, Jian-Jia Chen, and Lothar Thiele. Power management in energy harvesting embedded systems with discrete service levels. In *Submitted to The International Symposium on Low Power Electronics and Design (ISLPED)*, August 19-21, 2009.

[ML99] M. Morari and J.H. Lee. Model predictive control: Past, present and future. *Computers & Chemical Engineering*, 23(4):667–682, 1999.

[MOH04] Kirk Martinez, Royan Ong, and Jane Hart. Glacsweb: a sensor network for hostile environments. In *The First IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks*, 2004.

[Mt06] Inc. Maxwell technologies. Boostcap ultracapacitor - pc series data sheet. June, 2006.

[MTBB07] Clemens Moser, Lothar Thiele, Davide Brunelli, and Luca Benini. Adaptive power management in energy harvesting systems. In *DATE '07: Proceedings of the Conference on Design, Automation and Test in Europe*, pages 773–778, NY, USA, 2007. ACM Press.

[MTBB08] Clemens Moser, Lothar Thiele, Davide Brunelli, and Luca Benini. Robust and Low Complexity Rate Control for Solar Powered Sensors. In *Design, Automation and Test in Europe (DATE 08)*, Munich, Germany, March 10-14 2008.

[NKA+05] Lama Nachman, Ralph Kling, Robert Adler, Jonathan Huang, and Vincent Hummel. The intel®mote platform: a bluetooth-based sensor network for industrial monitoring. In *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*, page 61, Piscataway, NJ, USA, 2005. IEEE Press.

[PC06] Chulsung Park and P.H. Chou. Ambimax: Autonomous energy harvesting platform for multi-supply wireless sensor nodes. In *Proceedings of the Sensor and Ad Hoc Communications and Networks. SECON '06.*, volume 1, 2006.

[PCFZ05] Shashank Priya, Chih-Ta Chen, Darren Fye, and Jeff Zahnd. Piezoelectric windmill: A novel solution to remote sensing. *Japanese Journal of Applied Physics*, 44(3):L104–L107, 2005.

[PSC05] J. Polastre, R. Szewczyk, and D. Culler. Telos: Enabling ultra-low power wireless research. In *4th International Conference on Information Processing in Sensor Networks (IPSN05)*, pages pp. 364–369, Piscataway, NJ, April 2005.

[PUBG01] B. Prabhakar, Elif Uysal-Biyikoglu, and A. El Gamal. Energy-efficient transmission over a wireless link via lazy packet scheduling. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, pages 386–394, Anchorage, AK, USA, April 2001.

[RKH+05] Vijay Raghunathan, Aman Kansal, Jason Hsu, Jonathan Friedman, and Mani B. Srivastava. Design considerations for solar energy harvesting wireless embedded systems. In *Proceedings of the Fourth International Symposium on Information Processing in Sensor Networks, IPSN 2005*, pages 457–462, UCLA, Los Angeles, California, USA, April 25-27 2005.

[RM04] Kay Roemer and Friedemann Mattern. The design space of wireless sensor networks. In *IEEE Wireless Communications*, volume 11, pages 54–61, December 2004.

[RMM03] Cosmin Rusu, Rami G. Melhem, and Daniel Mosse. Multi-version scheduling in rechargeable energy-aware real-time systems. In *15th Euromicro Conference on Real-Time Systems, ECRTS 2003*, pages 95–104, Porto, Portugal, July 2-4 2003.

[RSF+04] Shad Roundy, Dan Steingart, Luc Frechette, Paul K. Wright, and Jan M. Rabaey. Power sources for wireless sensor networks. In *Wireless Sensor Networks, First European Workshop, EWSN 2004, Proceedings*, Lecture Notes in Computer Science, pages 1–17, Berlin, Germany, January 19-21 2004. Springer.

[RWR03] Shad Roundy, Paul K. Wright, and Jan M. Rabaey. *Energy Scavenging for Wireless Sensor Networks with Special Focus on Vibrations*. ISBN: 978-1-4020-7663-3. Springer, 2003.

[SAL+03] J. Stankovic, T. Abdelzaher, C. Lu, L. Sha, and J. Hou. Real-time communication and coordination in embedded sensor networks. In *Proceedings of the IEEE*, vol.91, no.7pp. 1002-1022, July 2003.

[SC06] Farhan Simjee and Pai H. Chou. Everlast: long-life, supercapacitor-operated wireless sensor node. In *ISLPED '06: Proceedings of the 2006 international symposium on Low power electronics and design*, pages 197–202, New York, NY, USA, 2006. ACM Press.

[Sch87] M. Schechter. Polyhedral functions and multiparametric linear programming. *Journal of Optimization Theory and Applications*, 53(2):269–280, 1987.

[SLC91] W.-K. Shih, J. W.-S. Liu, and J.-Y. Chung. Algorithms for scheduling imprecise computations with timing constraints. *SIAM J. Computing*, 20(3):537–552, June 1991.

[sol07] Bern university of applied sciences, engineering and information technologies, photovoltaic lab: Recordings of solar light intensity at Mont Soleil from 01/01/2002 to 31/09/2006. www.pvtest.ch, March, 2007.

[SP01] N.S. Shenck and J.A. Paradiso. Energy scavenging with shoe-mounted piezoelectrics. *Micro, IEEE*, 21(3):30–42, May/Jun 2001.

[SPMC04] Robert Szewczyk, Joseph Polastre, Alan Mainwaring, and David Culler. Lessons from a sensor network expedition. In *In Proceedings of the First European Workshop on Sensor Networks (EWSN*, pages 307–322, 2004.

[Sta06] I. Stark. Invited talk: Thermal energy harvesting with thermo life. In *Wearable and Implantable Body Sensor Networks, 2006. BSN 2006. International Workshop on*, pages 19–22, 2006.

[Sun06] Sunergy. Recordings of solar light intensity from 08/08/2005 to 04/09/2005, datasheet. http://sunergy.dmclf.org/cms/, June, 2006.

[VGB07] Christopher M. Vigorito, Deepak Ganesan, and Andrew G. Barto. Adaptive control of duty cycling in energy-harvesting wireless sensor networks. In *4th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON 2007)*, pages 21–30, 2007.

[VRS+04] Thiemo Voigt, Hartmut Ritter, Jochen Schiller, Adam Dunkels, and Juan Alonso. Solar-aware clustering in wireless sensor networks. In *Proceedings of the Ninth IEEE Symposium on Computers and Communications*, June 2004.

[WALJ+06] Geoff Werner-Allen, Konrad Lorincz, Jeff Johnson, Jonathan Lees, and Matt Welsh. Fidelity and yield in a volcano monitoring sensor network. In *OSDI '06: Proceedings of the 7th symposium on Operating systems design and implementation*, pages 381–396, Berkeley, CA, USA, 2006. USENIX Association.

[WMT05] Ernesto Wandeler, Alexander Maxiaguine, and Lothar Thiele. Quantitative characterization of event streams in analysis of hard real-time applications. *Real-Time Systems, Springer Science+Business Media B.V.*, 9(2):205–225, 2005.

[YDS95] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, pages 374–382, 1995.

[Yea04] Eric Yeatman. Advances in power sources for wireless sensor nodes. In *In Proc. 1st International Workshop on Body Sensor Networks*, pages 21–22, 2004.

[YK04a] Han-Saem Yun and Jihong Kim. Reward-based voltage scheduling for fixed-priority hard real-time systems. In *International Workshop on Power-Aware Real-Time Computing*, 2004.

[YK04b] Han-Saem Yun and Jihong Kim. Reward-based voltage scheduling for hard real-time systems with energy constraints. In *International Conference on Real-Time and Embedded Computing Systems and Applications (RTCSA)*, pages 416–435, 2004.

[YL04] Chia-Lin Yang and Chien-Hao Lee. Hotspot cache: joint temporal and spatial locality exploitation for i-cache energy reduction. In *ISLPED*, pages 114–119, 2004.

A

Adversary Inter-Frame Service Allocation Algorithm

In the following an adversary algorithm which tries to average the energy consumption is described. It respects the energy constraints given by the system and the environment, i.e., it is feasible according to Definition **??**. A straightforward approach would be to calculate an average energy consumption $\frac{E_C(0)+\sum_{i=1}^{K} E_H(i)-E_\ell}{K}$ for all frames. However, since the power source E_H is not constant, energy overflows and underflows may happen. As shown in Algorithm 7 in Steps 4 and 5, possible energy underflows for the next frame $k+1$ are avoided by reducing the energy consumption to $E_C(k) + E_H(k+1)$. Analogously, the energy consumption is increased in Step 11 to prevent the stored energy $E_C(k+1)$ from overflowing. If such an overflow is avoided, however, we consume more energy than initially planned and we might end up with an infeasible schedule. Hence, a recalculation for the remaining frames becomes necessary to obtain a feasible schedule. This is done in Steps 12-14 by again averaging the remaining energy. Finally, the reward of the schedule can be improved by recalculating the energies also for energy underflows (Steps 6-8). It is easy to verify that schedule $\vec{e}^a$ calculated by Algorithm 7 is feasible and uses all available energy ($E_C(0) + \sum_{i=1}^{K} E_H(i) - E_\ell$).

Algorithm 7 Adversary (Inter-Frame Service Allocation)

Input: K, $E_H(k)$ for $k = 1, 2, \ldots, K$, $E_C(0)$, E_ℓ, $E_{\max}$;
Output: a feasible assignment $\vec{e}^a$ of energy consumption for the K frames;

$k \Leftarrow 0$;
$e^a_j = \frac{E_C(0)+\sum_{i=1}^{K} E_H(i)-E_\ell}{K}$, $\forall j = 1, \ldots, K$;
while $k < K$ **do**
  if $E_C(k) + E_H(k+1) - e^a_{k+1} < 0$ **then**
    $e^a_{k+1} \Leftarrow E_C(k) + E_H(k+1)$;
    for $i = k+2; i \leq K; i \Leftarrow i+1$ **do**
      $e^a_j \Leftarrow \frac{\sum_{i=k+2}^{K} E_H(i)-E_\ell}{K-k-1}$; $\forall j = k+2, .., K$;
  if $E_C(k) + E_H(k+1) - e^a_{k+1} > E_{\max}$ **then**
    $e^a_{k+1} \Leftarrow E_C(k) + E_H(k+1) - E_{\max}$;
    for $i = k+2; i \leq K; i \Leftarrow i+1$ **do**
      $e^a_j \Leftarrow \frac{E_{\max}+\sum_{i=k+2}^{K} E_H(i)-E_\ell}{K-k-1}$; $\forall j = k+2, .., K$;
  $E_C(k+1) = E_C(k) + E_H(k+1) - e^a_{k+1}$;
  $k \Leftarrow k+1$;
return $\vec{e}^a$ as the solution;

List of Publications

The following list summarizes the publications which constitute the basis of this thesis. The pertinent chapters of this thesis are given in brackets.

C. Moser, D. Brunelli, L. Thiele, and L. Benini. Real-Time Scheduling with Regenerative Energy. In *The 18th Euromicro Conference on Real-Time Systems (ECRTS 06)*, July 2006.
(Chapter 2)

C. Moser, D. Brunelli, L. Thiele, and L. Benini. Lazy Scheduling for Energy Harvesting Sensor Nodes. In *The Fifth IFIP Working Conference on Distributed and Parallel Embedded Systems (DIPES 06)*, October 2006.
(Chapter 2)

C. Moser, D. Brunelli, L. Thiele, and L. Benini. Real-Time Scheduling for Energy Harvesting Sensor Nodes. In *Real-Time Systems, Special Issue on Real-Time Wireless Sensor Networks, Volume 37, Number 3, pages 233-260*, December 2007.
(Chapter 2)

C. Moser, L. Thiele, D. Brunelli, and L. Benini. Adaptive Power Management in Energy Harvesting Systems. In *Design, Automation and Test in Europe (DATE 07)*, April 2007.
(Chapter 3)

C. Moser, L. Thiele, D. Brunelli, and L. Benini. Robust and Low Complexity Rate Control for Solar Powered Sensors. In *Design, Automation and Test in Europe (DATE 08)*, March 2008.
(Chapter 3)

C. Moser, L. Thiele, D. Brunelli, and L. Benini. Approximate Control Design for Solar Driven Sensor Nodes. In *The 11th International Conference on Hybrid Systems: Computation and Control (HSCC 08)*, April 2008.
(Chapter 3)

C. Moser, L. Thiele, D. Brunelli, and L. Benini. Approximate Control Design for Solar Driven Sensor Nodes. In *TIK-Report-282, Computer Engineering and Networks Laboratory, ETH Zürich , (long version of the HSCC 08 paper)*, January 2008.
(Chapter 3)

C. Moser, L. Thiele, D. Brunelli, and L. Benini. Adaptive Power Management for Environmentally Powered Systems. Submitted to *IEEE Transactions on Computers*, December 2008.
(Chapter 3)

C. Moser, J.-J. Chen, and L. Thiele. Reward Maximization for Embedded Systems with Renewable Energies. In *The 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 08)*, August 2008.
(Chapter 4)

C. Moser, J.-J. Chen, and L. Thiele. Optimal Service Level Allocation in Environmentally Powered Embedded Systems. In *The 24th ACM Symposium on Applied Computing (SAC 09)*, March 2009.
(Chapter 4)

C. Moser, J.-J. Chen, and L. Thiele. Performance Optimization in Energy Harvesting Embedded Systems. Submitted to *ACM Transactions in Embedded Computing Systems (TECS)*, November 2008.
(Chapter 4)

It follows a list of publications that are not covered in this thesis.

C. Moser, J.-J. Chen, and L. Thiele. Power Management in Energy Harvesting Embedded Systems with Discrete Service Levels. Submitted to *The International Symposium on Low Power Electronics and Design (ISLPED)*, August 2009.

D. Brunelli, L. Benini, C. Moser, and L. Thiele. An Efficient Solar Energy Harvester for Wireless Sensor Nodes. In *Design, Automation and Test in Europe (DATE 08)*, March 2008.

D. Brunelli, C. Moser, L. Thiele, and L. Benini,. Design of a Solar Harvesting Circuit for Battery-less Embedded Systems. Accepted for publication in *IEEE Transactions on Circuits and Systems I*, XX 2009.

Curriculum Vitae

Name	Clemens Moser
Date of Birth	13 February 1979
Place of Birth	Friedrichshafen (Germany)
Citizen of	Altstätten (SG)
Nationality	Swiss and German

Education:

2005–2009	ETH Zurich, Computer Engineering and Networks Laboratory Ph.D. student under the supervision of Prof. Dr. L. Thiele
1999–2004	TU Munich, Germany Studies in Electrical Engineering and Information Technology Graduation as Dipl.-Ing. TUM in December 2004 Graduation as B.Sc. TUM in April 2003
1998–1999	Maltheser Hilfsdienst, Augsburg, Germany 13 Months Alternative Civilian Service (Zivildienst)
1989–1998	Rudolf-Diesel-Gymnasium Augsburg, Germany High School Graduation in June 1998
1985–1989	Elementary School, Mering, Germany

Professional Experience:

2005–2009	Research & Teaching Assistant at ETH Zurich
2000	Internship at Clariant GmbH, Augsburg, Germany
1999	Internship at Osram GmbH, Augsburg, Germany