Using Energy Budgets to Reach Lifetime Goals while Compensating Dynamic Effects

André Sieber, Jörg Nolte

Distributed Systems/Operating Systems Group
Brandenburg University of Technology Cottbus-Senftenberg
Cottbus, Germany
Email: {as, jon}@informatik.tu-cottbus.de

Reinhardt Karnapke

Communication and Operating Systems Group
Technische Universität Berlin
Berlin, Germany
Email: karnapke@tu-berlin.de

***Abstract*—Nodes within sensor networks often have tight bound goals for the lifetime while running from a non-renewable energy source. Variations within the hardware or induced by the software complicate the prediction of the energy consumption. Additionally, batteries are vulnerable to temperature and non-linear effects. To reach certain lifetime goals under these influences without sacrificing energy due to pessimistic estimations, online energy management is necessary. In this paper, we present policies to control the behavior of applications and devices using energy budgets. First experiments yield promising results, with nodes reaching their lifetime goals while maintaining a high application quality.**

Keywords–Wireless Sensor Networks; Energy Management; Energy Awareness; Lifetime Goals

I. INTRODUCTION

Wireless sensor networks are being used in a wide range of applications, with environmental monitoring and/or hazard detection among the most prominent ones. In many of these monitoring applications, the sensor networks should run for a certain time without direct human interaction. This requirement may result from the network being placed in a hardly reachable area, in a dangerous area (e.g., in a steelworks) or because human interference should be kept to an absolute minimum (e.g., when monitoring nesting animals). For these applications, the sensor networks are usually deployed with a certain time-to-live in mind, until either the next maintenance is due and batteries can be changed, or until the whole monitoring period (e.g., the breeding season) is over.

When application developers design networks with lifetime goals in mind, they carefully choose the appropriate sensing equipment, devise a communication pattern and select batteries that supply enough power. However, in some cases the usage of sufficient batteries is impossible, e.g., due to the form factor. If the sensor nodes need to fit in a certain area, the size of the batteries is limited. Even if there is no limiting form factor, there are a number of influences which can not be calculated easily. Sensor node hardware is subject to production variances, resulting in different power consumption characteristics even for components manufactured together. The same is true for batteries, only in reverse: The amount of energy they supply varies. Moreover, supply and consumption can both vary, depending on environmental conditions like temperature, the battery voltage level, or the dynamic voltage converter efficiency. Even more, batteries suffer from non-linear effects like rate-capacity and the recovery effect [1].

Sensor network applications that fail to take variances and run-time-effects into account may lead to early node failures due to depleted energy. To prevent this, the required energy is often overestimated on purpose. The amount of energy for reaching the lifetime goal is calculated and a safety margin (e.g., 20%) is added afterwards. Please note that these 20% are calculated using the nodes with the highest assumed consumption. When, for example, the communication represents the main energy drain, the nodes closest to the sink would be used as basis, as they need to forward messages more often than outer nodes. Even though this results in wasted energy, it is often thought of as being necessary. However, the quality of service supplied by the application can be improved at least on some nodes, if it is possible to use all available energy. In the example, some outer nodes could sample/sense more often to increase the quality of the data, e.g., by calculating averages or max values, without increasing the network load.

In order to avoid energy being wasted, dynamic energy management is necessary, with a manager that can use various handles to influence the energy consumption on each node individually, based on the available energy. The application duty cycle can be changed or the MAC/routing timings adapted. Sensors can be used with varying levels of detail or granularity. However, this requires the possibility of isolating tasks and devices. To enable the management to limit their consumption, we introduce fine grained energy budgets. These budgets are assigned to individual (sub-)tasks, provide a certain amount of energy, and enable the energy management to plan how to distribute the scarce resources.

Even though the main goal is to reach a certain lifetime with the whole network, the methods presented in this paper are primarily concerned with the options available on each individual node. The reasons for this are twofold:

1) Reaching the lifetime goal with the whole network requires each node to reach the lifetime goal.
2) Network wide strategies require communication, which in itself induces more energy consumption. However, the presented approach does not obstruct a global strategy, e.g., adaption of routes to distribute the communication load. The presented approach can even compensate an increase or decrease in communication load, as it only represents another dynamic influence.

Figure 1 shows an example network with a grid topology containing 11 times 11 nodes. Each node has a variable sensing interval and may need to forward messages from neighboring nodes. The routing topology is a tree, resulting in a higher communication load on nodes closer to the sink. Therefore, these nodes need to adjust their sensing interval in order to

 ISBN: 978-1-61208-425-1

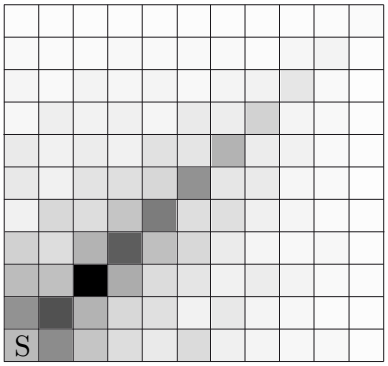

Figure 1. Difference of Sensing Interval due to balancing the load with a non-uniform network load. Sensing interval ranges from 1.71s (white) to 3.36s (black)

conserve energy and reach the lifetime goal. In the example, outer nodes sample every 1.71 seconds while the inner nodes sample less often, the longest interval is nearly twice as long (3.36 seconds).

The rest of this paper is structured as follows: In Section II related work is presented. Section III focuses on the energy budgets principles and our approaches to react on changing energy demands and extreme situations. In Section IV early evaluation results are shown. Finally, a conclusion is given in Section V.

II. Related Work

Energy management on sensor nodes traditionally faces three questions:

1) How much energy is available within the energy source (usually translating to the state-of-charge of the battery)?
2) How much energy is consumed by the (different parts of the) application?
3) How much of the available energy can be spent and how is it partitioned between consuming parts to aid the application goal?

There are three kinds of approaches to answer the first question. Hardware based approaches like fuel gauges, e.g., [2] or smart battery monitors like [3] supply accurate information about the remaining energy in batteries but in turn increase the system complexity and energy consumption. Model based approaches, e.g., [4], [5], use chemical or analytic models to predict the behavior of the energy source. They are very accurate but suffer from high complexity and computation costs, which renders them unusable for deeply embedded systems. They also often rely on complex parameters which have to be retrieved via experiments and thus depend on a certain type of battery. Measurement based approaches use easily observable parameters, often the voltage, to estimate the state of charge. They can be based on tables generated in advance [6] or observe the relative voltage decline [7]. Due to the limited resources and energy constraints in wireless sensor networks, we favor the approach presented in [7].

Tracking the consumed energy and thus answering the second question is possible both in hard- and software. Hardware approaches cover Coulomb counters and smart battery systems [3], *Sensor Node Management Devices* [8] and specifically designed measurement devices [9], [10]. All hardware approaches introduce an overhead in device costs and energy consumption. Software approaches rely on the observation of certain events to account for the consumption. To be able to observe these events, the code has to be modified with hooks to call the accounting functions. The events can either be based on functional application blocks [11], [12] or on device driver actions [13], [14]. The latter are based on taking time stamps when devices change their state to obtain the duration of each state and calculate the consumed energy. While the hardware based approaches deliver precise results, we favor a software based approach using time stamps [15], as it is capable of delivering not only the global (node wide) consumption but can also distinguish between individual system elements, including software components. In contrast to other approaches, it is capable of taking varying consumption due to voltage changes of the battery and varying efficiency of potentially used voltage converters into account.

There are various approaches for managing the energy consumption and answering the third question. Table I shows an overview over existing approaches and a simple categorization based on the platform they are ran on, the scope of the approach and the controlled entity/granularity. The management can be done either with a direct influence on system and application parameters (upper half) or indirect, using a limited resource distributed among application parts (lower half).

Providing the application with energy awareness by using an indirect approach enables the use of different adaption strategies. Instead of providing a handle to the management, the application itself can manage its consumption. Apart from using individual service levels and controlling timers, more complex schemes, like limiting the number of forwarded messages, are possible. This makes indirect approaches more flexible. Based on the concept of resource containers (RC) [27], energy capsules [28] and energy containers [29] provide information about the energy usage of (sub-)tasks, but do not take advantage for energy management from them. The indirect approaches of Cinder [21] and ECOSystem [22] utilize this concept to store information about energy availability.

Apart from the approaches that are based on setting the service level, EPOS [17], Nemesis [25] and SORA [26] also suffer from insufficient isolation between application parts, resulting in a potentially uncontrolled impact of a changed

TABLE I. OVERVIEW OF ENERGY MANAGEMENT APPROACHES. DIRECT APPROACHES IN UPPER HALF, INDIRECT IN LOWER HALF.

	platform	scope	controlled entity
Odyssey [16]	PC	local	Service Level
EPOS [17]	WSN	local	Task
Energy Levels [6]	WSN	local	Service Level
Eon [18]	WSN	local	Timer/Service Level
EMA [19]	WSN	local	Timer
IDEA [20]	WSN	neighborhood	Service Level
Cinder [21]	mobile phone	local	energy distribution
ECOSystem [22]	PC	local	energy distribution
Pixi OS [23]	WSN	local	energy distribution
Virtual Battery [24]	WSN	local	energy distribution
Nemesis OS [25]	PC	local	price
SORA [26]	WSN	global	price

 ISBN: 978-1-61208-425-1

demand, up to starvation. Only Virtual Battery [24] and Pixi OS [23] resolve short-term energy shortages by allowing debts. Most approaches lack the flexibility to resolve long-term changes of the energy demand of individual application parts. Cinder, EMA [19] and Pixi OS have a high computational overhead as they work continuously instead of periodically. Approaches that do not only consider local information but include neighborhood (IDEA [20]) and global information (SORA [26]) suffer from increased energy consumption due to the communication overhead. Additionally, the preparation effort depends on the used energy accounting and the parameters needed by the management. IDEA, SORA and Energy Levels have a relatively high demand.

All existing approaches have different benefits and drawbacks. Especially the reaction to changes in the energy demand of single application(-parts) is often inadequately addressed. However, the management must be capable of dynamically adjusting single application(-parts). Fine-grained management makes isolation between single applications or application parts necessary to prevent starvation. Indirect management enables energy-aware applications, but needs to provide mechanisms of direct management to aid the applications through functions for calculating duty cycles and service levels. Using a periodic approach reduces overhead and the necessary planning horizon of the applications.

III. Management using Energy Budgets

Using information about the energy consumption of the system, it is possible to enforce limits to meet desired discharge rates. Applications should react to reach lifetime goals, therefore they must be provided with information about the available energy. With different application parts and goals competing for the energy, a mechanism which separates them and provides a local view of the remaining energy for a single part is necessary. However, the allocated energy also needs to be associated with a time window. This information can be provided using energy budgets, which are also based on the concept of resource containers [27], but are only used to limit a single resource, namely energy, here.

An energy budget is an abstract reservoir for energy. It represents the right to a certain amount of energy. The system energy (or parts of it) is/are divided between the budgets. An abstract budget B is defined by its currently stored amount b and the validity interval $[t_{start}, t_{end}]$ of b. The demand of a budget is defined by the minimal energy min needed and the maximal energy max consumable by its associated consumers during $[t_{start}, t_{end}]$. They give the system a hint on reasonable values for filling Budget B at t_{start} (1).

$$B = (b, [t_{start}, t_{end}], min, max) \tag{1}$$

As most activities within a sensor node live rather long, the validity interval is more or less a constant refresh interval at whose begin the budget is refilled. This also divides the system consumption and reduces the prediction horizon for the application. The essential requirement for a reliable operation of the application is that the minimal demand for all budgets is satisfied in every interval. In scenarios using harvesting, this requirement may not be fulfilled due to the unsteady energy income. In times with no income, nodes must be able to perform only essential tasks and wait for more income. The remaining energy can be distributed among the budgets in various ways (discussed later). If no energy harvesting is available, the network's maintainer must be informed immediately if this demand can not be satisfied, as the functionality of the node or even the whole network can no longer be guaranteed.

Obtaining reasonable values for min and max can be hard, as it involves the determination of boundaries of the energy consumption of individual application parts. Apart from careful calculation, these values could be obtained by simulation or simply by experiments. This process could also be automated to a certain degree by including a learning phase. But if the demand fluctuates or is based on spontaneous events, min and max may represent the demand inadequately. Thus, the management must be able to adjust the distributed amount of energy and other forms of cooperation, e.g., lending energy, can be utilized by the budgets (see below).

Using multiple budgets in parallel is not a problem, as long as different devices are used. Shared resources like the CPU need to be accounted by using requests. Fairness, as called for in [29], can be achieved by logging the input, e.g., in the form of transmitted bytes, and then diving the total consumed energy by the total number of bytes transmitted. Then, each task can be charged with this value times the number of bytes it transmitted. If all requests always result in the same consumption, this approach can be simplified to counting the number of requests. It can also be applied to buffered operations, like writing onto an SD-card. Then, the energy would be subtracted from the budget before the actual operation takes place.

Energy budgets can be mapped to different entities. In some cases, they might be associated with a device or a certain device mode. This also enables task- or activity granularity. On a larger scale, they can also be applied to whole execution paths of a program, e.g., a message that is created, a routing decision made, the MAC involved and the radio hardware activated.

The basic concept of energy budgets and their behavior can be extended and policies implemented in numerous ways. Apart from the way the energy is distributed among the budgets, there are different ways to approach special situations when the energy is insufficient or not consumed. Which policy or combination should be used depends mainly on the requirements of the application. Our initial evaluation shows that with different policies, the behavior of the system varies widely in different situations.

A. Energy Allowance

The main goal of energy distribution is to reach the lifetime goal while distributing the energy in a fair way. In this case, fair means that for all budgets at least their minimum requirement is allocated and no budget is favored unintentionally. There may be user defined utilities (U_i) as well as boundary conditions (min_i and max_i) which have to be taken into account and can lead to an uneven distribution of energy. The energy distribution into budgets boils down to a resource allocation problem, where resources are allocated based on utilities (2).

$$max \sum U_i(E_i) \tag{2}$$

subject to:

$$min_i \leq E_i \leq max_i$$

$$\sum E_i \leq E$$

The utility function U_i represents the amount of useful work that can be gained by adding more energy to the budget.

This allocation problem can be solved with brute force or with linear optimization, but these approaches are hard to realize on resource constrained sensor nodes. Additionally, the formal problem description can lead to a solution where budgets with the highest utility get all the energy. Even though this is a valid solution, it is an unwanted one. An additional constraint is necessary which takes the utility as weight for a fair mix of all budgets. This is possible with a linear utility function. By distributing the minimum demand first, the problem can be eased. The additional energy available can then be scattered, e.g., in percent of the max requirement of each task (3):

$$b_i = b_i + \frac{(max_i - b_i) * U_i}{\sum (max - b) * U} * E \tag{3}$$

This heuristic can lead to situations where the share of a budget is higher than its max_i. Then, the amount exceeding the maximum remains in the global energy pool and equation 3 is applied again.

Since the available energy als well as the demand and utility of application parts can change, the allowance must be done frequently. The allowance frequency again depends on the application demand and scenario. With increased consumption, the frequency should be also increased, to be able to react to changes in the available and requested energy in a timely manner.

B. Extreme Situations

In some cases, the calculation of the remaining energy might reveal that there is not enough energy left to fulfill the requirements of all tasks until the end of the lifetime goal. Then, the choice remains to either divide the energy strictly and not fulfill the requirements in some intervals, or to continue as before, accepting a possible early node failure but informing the network maintainer. In other cases, a budget might have received more energy than it can consume. For the associated task, this is not a problem. However, the feedback to the accounting might lead to wrong decisions, as the observed state of charge (SoC) suddenly is higher than anticipated. Once the task starts consuming all allocated energy within its budget, the battery is drained faster than the management expects. In a third case, a task might suddenly need more energy than in the previous intervals, e.g., for additional message retransmissions. If this amount is higher than the budget allocated by the management, the task might steal energy from a different task if its priority is higher. It could also borrow energy from the system or another task that does not need all of its budget in this interval, but would have to return the favor later.

C. The Duty of the Application

All of the methods described above are used to distribute energy, and give the application incentives to chance its behavior if there is not enough or more than the required amount of energy available. The application itself needs to react to these incentives, for example by adapting its duty cycle. Devices that are not directly influenced must also be taken into account by the application (e.g., the radio). As described above, the easiest way to guarantee fairness is to calculate costs for shared resources based on the number of bytes or requests.

The number of budgets used within the system depends mainly on the application. While it is possible to use only a single budget for a full sensor network application, including sensing and communication, the isolation of application parts eases the adaption decision, as only the relevant parts must be considered.

As literature shows, there are two common ways to adapt to a changed energy availability. First, the application's duty cycle can be changed within a tolerable range. Second, the application can adapt by using distinct service levels. While the first is appropriate for sensing tasks, the latter allows more complex changes to the behavior of the application. As the necessary information is provided by the budgets, both can be implemented easily.

The duty cycle τ is computed based on the remaining time d before the budget is refilled, the consumption of the last execution c and the energy available b_i (4).

$$\tau = \lfloor \frac{d * c}{b_i} \rfloor \tag{4}$$

The service level σ is calculated as ratio of how much the budget is filled compared to min and max (5).

$$\sigma = \begin{cases} 1 & \text{if } b_i \leq min_i \\ N & \text{if } b_i \geq max_i \\ \frac{N*(b_i - min)}{max - min} & \text{else} \end{cases} \tag{5}$$

IV. Evaluation and Results

In order to evaluate and compare the different approaches, we used simulations and experiments with real sensor nodes. The experiments were realized using the so-called FeuerWhere nodes [30]. These nodes feature a MSP430 micro controller and three transceivers, one operating at 868 MHz and two operating at 2.4 GHz. We used Reflex [31], an operating system for deeply embedded systems, as basis for the integration of our energy budgets. As real experiments are better suited to prove that our approach works, we focus only on those in the following evaluations.

A. Cost of Energy Allowance

To evaluate the computational overhead of the allowance heuristic, the runtime of the algorithm with different options was measured. Figure 2 shows the results.

While using only the heuristic (equation 3) has the lowest runtime costs, it alone can not guarantee the minimal demand (A). Distributing the minimal demand of each budget first (B) increases the runtime only slightly but fulfills the requirements. To increase the dynamic between the budgets, it is feasible to take the consumed and requested energy of the budgets into account (C). Distributed between the minimal demand and the heuristic, the runtime increases considerably, as the additional step is a modified version of the heuristic itself.

 ISBN: 978-1-61208-425-1

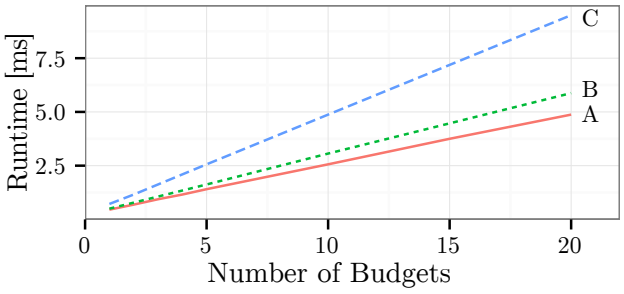

Figure 2. Runtime of the distribution algorithm for MSP430 running at 16MHz. (A) Based on percentage of maximal requested energy. (B) Additionally distribution of minimal requested energy. (C) Distribution of consumed/requested energy.

With increased number of budgets the runtime rises linearly. As the allocation of energy only happens every few hours in most scenarios, the overhead is negligible even when a lot of budgets are in use.

B. *Influence of different policies*

To evaluate the influence of the policies in situations where the energy demand changes, we used two scenarios. Two tasks were involved, one sampling and one transmitting messages. The experiments ran for three phases with 50 seconds per phase. The sending task wanted to transmit a message every ten seconds while the sampling interval was adjusted dynamically.

1) Scenario 1: In the first scenario the amount of energy allocated to the sending task in each phase was only sufficient to send four of the five desired messages, forcing it to adapt according to the chosen policy: Isolation, adaption, lending, or stealing.

Figure 3 shows the results. The vertical lines represent the transmitted messages, the line that starts horizontally is the sampling interval. When using isolation as policy, there is no way the sending task can obtain enough energy. Therefore, every fifth message is not transmitted. There is no influence on the sampling task.
When the adaption policy is used, the sending task receives the minimum energy it requires to fulfill its assignment, enabling it to transmit all five messages in each interval. However, the energy consumption is higher, the remaining energy lower, which results in an adjustment of the sampling interval for phases two and three. Sampling less frequent conserves the energy required for the sending task.
The third part of the figure shows the policy lending, where the sending task borrows energy from the system. However, what is borrowed needs to be paid back, resulting in no transmissions at the end of the third phase. The sampling is not influenced in any way.
Stealing, the fourth policy, takes the energy required for sending directly and at the time it is required from the sampling task, making huge changes in the sampling interval necessary near the end of each phase.

The choice between policies is up to the application programmer, each variant has its advantages and disadvantages. Adaption makes sure that all parts of the sensor node continue to function, albeit some of them are under the influence of others. Lending ensures that only the part that requires too much energy may fail early, while all others continue to function as planned. This may enable a timely response from the network maintainer. Lending and stealing might also be combined.

2) Scenario 2: In the second scenario the energy consumption of the sending task was lower, resulting in a too large budget. Figure 4 shows the results for four different policies: unlimited savings, isolation, adaption and no savings.

When a task may store an unlimited amount of energy, all the excess of energy remains with that task and there is no influence on the other tasks (upper left). When the isolation policy is used, a budget that has not been completely spent needs less energy to be filled again. Therefore, the remaining system energy is higher and more energy can be distributed among all other tasks, in this case the sampling task (upper right). When using adaption, the budget for the sending task is reduced to the amount it used, making an even smaller amount of energy necessary to fill it in phase two, leading to more system energy and more energy available for the sampling task in phase two. However, once the adaption is complete, the energy available for the sampling task is reduced again in phase 3 and would remain the same in future phases (lower

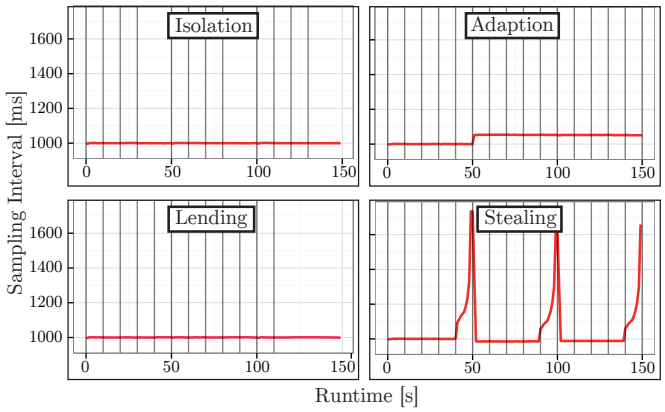

Figure 3. Influence of different policies when more energy is needed than allocated

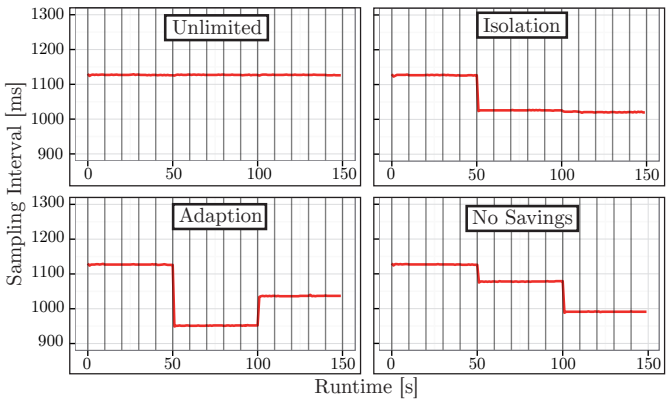

Figure 4. Influences of the choice of policy if a task needs less energy than assumed

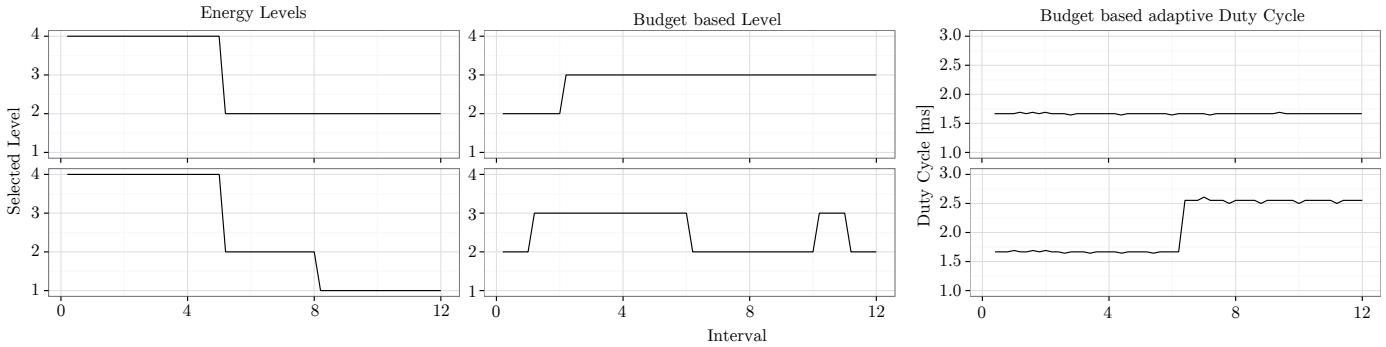

Figure 5. Experimental comparison of management approaches. *Energy Levels* computes feasible level assignment based on available energy and level utility. *Budget based Level* assigns level based on available energy in budget. Budget based duty cycle is computed based on available energy and runtime cost. Lower half shows experiment where in interval 6 the available energy was reduced by 20%.

left). The policy of no savings removes all remaining energy from a budget at the end of a phase, also resulting in a higher system energy and more energy available for the sampling task. As the too-high budget is filled at every phase, there is also some energy to ”reclaim” at the end of each phase (lower right).

The comparison shows that allowing a task to store an unlimited amount of energy in its budget is a bad idea, as the stored energy is effectively lost, unless there comes a time when the task needs a tremendous amount of energy. The other three policies differ mainly in the point in time at which they reclaim the unused energy, but they all lead to an improvement for the sampling task.

C. System behavior

To evaluate the management behavior, a comparison of our energy budgets and the *Energy Levels* approach was conducted. The application periodically sent messages to a sink and featured four service levels distinguished by the sensor duty cycle. The energy consumption of the different service levels varies between 100µA and 270µA, resulting in potential runtime between 32 and 11 months with conventional alkaline batteries. Periodically, one of the approaches is used to compute the service level for the following interval. To increase the comparability, both managers were based on the same energy accounting. *Energy Levels* uses a simplex algorithm to compute a feasible service level allocation based on the available energy, the consumption of each of the 4 used levels, and a utility ranging from one to four for each level (see [6] for the description of the optimization problem). As described in [6], the highest possible level is selected to provide the best QoS. The energy budget version (*Budget based Level*) used one budget and selected the service level based on the available energy within the budget (see 5). Additionally, a variant based on adaptive computation of the duty cycle based on equation 4 was tested using two budgets to isolate the sensing expenses from the communication.

The top half of Figure 5 shows the results achieved by each variant with a steady declining energy reserve. As *Energy Levels* selects the highest possible level, it starts with the highest level and later falls down to second last one. In contrast, the energy budget variant starts with the second last level, but shortly increases to and maintains the next higher level. While this approach is more conservative, it does not depend on future energy availability. This is also true for the adaptive duty cycle, which is based on the energy within the associated budget and its consumption. The computed sampling interval is steady around 1.66s and thus slightly higher than level 3 (fixed 1.5s interval).

The lower half of Figure 5 shows the impact of a varying energy availability on the three variants. In the middle of the experiment, the available energy is reduced by 20% (e.g., due to reduced battery capacity). As the *Energy Levels* starts with the highest level, it must use the lowest level in the last third, because the optimistic approach used more energy at the beginning. The *Budget based Level* consumes the energy balanced with the runtime and thus reduces the service level only by one, resulting in less drastic changes. The duty cycle of the third variant adapts after the changed energy availability to 2.55s and thus is lower than level 2 (fixed 3s interval).

Both variants based on the energy budgets not only deliver a more homogeneous application quality, but the battery may also benefit from a reduced rate-capacity effect due to the the overall lower load of the conservative approach.

D. Interaction with the battery

The results of an experiment which included feedback from the battery are shown in Figure 6. It shows how the energy is divided among three budgets associated with three application parts. Sensing covers all energy expenses involved with the

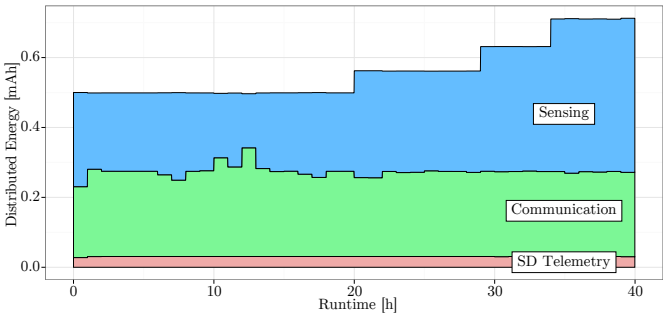

Figure 6. Distributed energy to different application parts. Battery Manager increases available energy over time due to higher capacity than expected.

data acquisition based on a dynamically calculated duty cycle. Communication includes a fixed communication interval every 2 minutes and SD Telemetry writes system information onto an SD-Card periodically.

In the course of the experiment, the communication effort varies and influences the energy distributed to sensing and its possible duty cycle. Additionally, the battery contains more energy than expected, resulting in an increased availability. As Communication and SD telemetry need a nearly fixed amount, sensing and, thus, the application duty cycle profits.

V. Conclusion

In this paper, we have presented our approach of using fine grained energy budgets to reach lifetime goals in wireless sensor networks. *Energy budgets* present a framework for dealing with varying demands in sensor networks. We evaluated the approach using FeuerWhere sensor nodes and presented promising first results for different kinds of policies. Additionally, we compared our approach with an existing approach and showed how the battery feedback influences the energy management.

In the future we plan to continue the evaluation of our approach by using different policies in a real deployment.

References

[1] T. Reddy, Linden's Handbook of Batteries, 4th Edition. Mcgraw-hill, 2010.

[2] Texas Instruments, "Datasheet bq26500 single cell li-ion and li-pol battery gas gauge." [Online]. Available: http://www.ti.com (Accessed July 10, 2015)

[3] Maxim Integrated, "Datasheet ds2438 smart battery monitor." [Online]. Available: http://www.maximintegrated.com (Accessed July 10, 2015)

[4] M. Doyle, T. F. Fuller, and J. Newman, "Modeling of galvanostatic charge and discharge of the lithium/polymer/insertion cell," Journal of the Electrochemical Society, vol. 140, no. 6, 1993, pp. 1526–1533.

[5] C.-F. Chiasserini and R. R. Rao, "Energy efficient battery management," Selected Areas in Communications, IEEE Journal on, vol. 19, no. 7, 2001, pp. 1235–1245.

[6] A. Lachenmann, P. J. Marrón, D. Minder, and K. Rothermel, "Meeting lifetime goals with energy levels." in SenSys, vol. 7, 2007, pp. 131–144.

[7] A. Sieber and J. Nolte, "Utilizing voltage decline for reaching lifetime goals," Paderborn, Germany, Tech. Rep., 2011.

[8] A. Hergenröder, J. Horneber, D. Meier, P. Armbruster, and M. Zitterbart, "Distributed energy measurements in wireless sensor networks," in Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems. ACM, 2009, pp. 299–300.

[9] X. Jiang, P. Dutta, D. Culler, and I. Stoica, "Micro power meter for energy monitoring of wireless sensor networks at scale," in Proceedings of the 6th international conference on Information processing in sensor networks. ACM, 2007, pp. 186–195.

[10] P. Dutta, M. Feldmeier, J. Paradiso, and D. Culler, "Energy metering for free: Augmenting switching regulators for real-time monitoring," in Information Processing in Sensor Networks, 2008. IPSN'08. International Conference on. IEEE, 2008, pp. 283–294.

[11] A. Lachenmann, P. J. Marrón, D. Minder, and K. Rothermel, "Meeting lifetime goals with energy levels," in Proceedings of the 5th international conference on Embedded networked sensor systems. ACM, 2007, pp. 131–144.

[12] A. Castagnetti, A. Pegatoquet, C. Belleudy, and M. Auguin, "An efficient state of charge prediction model for solar harvesting wsn platforms," in Systems, Signals and Image Processing (IWSSIP), 2012 19th International Conference on. IEEE, 2012, pp. 122–125.

[13] S. Kellner and F. Bellosa, "Energy accounting support in tinyos," PIK-Praxis der Informationsverarbeitung und Kommunikation, vol. 32, no. 2, 2009, pp. 105–109.

[14] A. Dunkels, F. Osterlind, N. Tsiftes, and Z. He, "Software-based on-line energy estimation for sensor nodes," in Proceedings of the 4th workshop on Embedded networked sensors. ACM, 2007, pp. 28–32.

[15] A. Sieber and J. Nolte, "Online device-level energy accounting for wireless sensor nodes," in Proceedings of the 10th European conference on Wireless Sensor Networks, 2013, pp. 149–164.

[16] J. Flinn and M. Satyanarayanan, "Energy-aware adaptation for mobile applications," in Proceedings of the Seventeenth ACM Symposium on Operating Systems Principles, ser. SOSP '99. New York, NY, USA: ACM, 1999, pp. 48–63.

[17] G. R. Wiedenhoft, L. F. Wanner, G. Gracioli, and A. A. Fröhlich, "Power management in the epos system," ACM SIGOPS Operating Systems Review, vol. 42, no. 6, 2008, pp. 71–80.

[18] J. Sorber, A. Kostadinov, M. Garber, M. Brennan, M. D. Corner, and E. D. Berger, "Eon: a language and runtime system for perpetual systems," in Proceedings of the 5th international conference on Embedded networked sensor systems. ACM, 2007, pp. 161–174.

[19] X. Jiang and J. Taneja, "Energy management for wireless sensor networks," CS270 Project Report, Spring 2007.

[20] G. W. Challen, J. Waterman, and M. Welsh, "Idea: Integrated distributed energy awareness for wireless sensor networks," in Proceedings of the 8th international conference on Mobile systems, applications, and services. ACM, 2010, pp. 35–48.

[21] A. Roy, S. M. Rumble, R. Stutsman, P. Levis, D. Mazières, and N. Zeldovich, "Energy management in mobile devices with the cinder operating system," in Proceedings of the sixth conference on Computer systems. ACM, 2011, pp. 139–152.

[22] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat, "Ecosystem: Managing energy as a first class operating system resource," in ACM SIGPLAN Notices, vol. 37, no. 10. ACM, 2002, pp. 123–132.

[23] K. Lorincz, B.-r. Chen, J. Waterman, G. Werner-Allen, and M. Welsh, "Resource aware programming in the pixie os," in Proceedings of the 6th ACM conference on Embedded network sensor systems. ACM, 2008, pp. 211–224.

[24] Q. Cao, D. Fesehaye, N. Pham, Y. Sarwar, and T. Abdelzaher, "Virtual battery: An energy reserve abstraction for embedded sensor networks," in Real-Time Systems Symposium, 2008. IEEE, 2008, pp. 123–133.

[25] R. Neugebauer and D. McAuley, "Energy is just another resource: Energy accounting and energy pricing in the nemesis os," in Hot Topics in Operating Systems, 2001. Proceedings of the Eighth Workshop on. IEEE, 2001, pp. 67–72.

[26] G. Mainland, D. C. Parkes, and M. Welsh, "Decentralized, adaptive resource allocation for sensor networks," in Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2. USENIX Association, 2005, pp. 315–328.

[27] G. Banga, P. Druschel, and J. C. Mogul, "Resource containers: A new facility for resource management in server systems," in OSDI, vol. 99, 1999, pp. 45–58.

[28] A. Dunkels, J. Eriksson, N. Finne, and N. Tsiftes, "Powertrace: Network-level power profiling for low-power wireless networks," Swedish Institute of Computer Science, 2011.

[29] S. Kellner, "Flexible online energy accounting in tinyos," in Real-World Wireless Sensor Networks. Springer, 2010, pp. 62–73.

[30] K. Piotrowski, S. Ortmann, and P. Langendörfer, "Multi-radio wireless sensor node for mobile biomedical monitoring," Biomed Tech, vol. 57, 2012, p. 1.

[31] K. Walther, R. Karnapke, and J. Nolte, "An existing complete house control system based on the reflex operating system: Implementation and experiences over a period of 4 years," in Proceedings of 13th IEEE Conference on Emerging Technologies and Factory Automation, 2008, pp. 40–45.

 ISBN: 978-1-61208-425-1