Energy-Efficient Voltage Scheduling of Peripheral Components on Wireless Sensor Nodes

Stephan Friedrichs, Ulf Kulau and Lars Wolf
Technische Universität Braunschweig
Institute of Operating Systems and Computer Networks
Mühlenpfordtstraße 23
D-38106 Braunschweig
Email: {sfriedr,kulau,wolf}@ibr.cs.tu-bs.de

***Abstract*—Much effort has been put into optimizing the energy-efficiency of wireless sensor nodes, but existing work exclusively focuses on the transceiver and the processing unit. Nevertheless, the peripheral energy consumption may dominate that of the entire node. We introduce the concept of a dynamically scalable peripheral voltage supply: Even for peripheral devices, a lower voltage level leads to a lower energy consumption. Each peripheral requires a different minimum operating voltage, but switching the voltage level consumes energy as well. We combine theory and practice to present an algorithm weighing off the benefits of a downscaled voltage level against the switching overhead, i. e., for calculating an optimal peripheral voltage schedule. Our approach is capable of self-parameterization and has been implemented and tested on a prototype, saving up to 47 % of peripheral energy as compared to existing solutions.**

I. Introduction

Compared to the growth of computational capabilities (Moore's law), the evolution of adequate mobile energy sources is limping. Hence, a significant amount of research aims at improving the lifetime of mobile applications like wireless sensor networks (WSNs). In this context, Dynamic Voltage Scaling (DVS) is a well-known technique, which depends on the ability of a system to adapt the voltage level to its workload. As the dynamic power consumption of CMOS gates shows a quadratic dependency on the voltage level, DVS helps to significantly improve the energy efficiency of micro-electronic systems [1], which has been exploited in several existing approaches [2], [3], [4], [5], leading to an increase of WSN lifetime. Nevertheless, existing work exclusively focuses on the micro-controller unit and does not consider peripherals like memory devices, sensors, or actuators. However, for some applications, a set of various sensors is needed [6]. Thus, the active power consumption of the sensing unit can easily exceed that of the micro-controller unit. For example, the current consumption of a typical gyroscope [7] is about three times higher than those of the msp430 [8] micro-controller.

Hence, the reduction of peripheral power consumption has a significant impact on the overall energy efficiency of a sensor node. Typical sensor node implementations lose much energy to a fixed, predefined, global peripheral voltage level. In this paper, we seize the concept of scalable peripheral voltage to optimize the overall power consumption and thus battery life.

For this purpose, we assume an a-priori known usage sequence of the peripheral devices on one sensor node, e. g. one that is periodically repeated, for instance: "read sensor 1, read sensor 2, write to external memory, read sensor 1, . . . ". Every peripheral device requires a minimum peripheral voltage level and a lower voltage implies lower energy consumption [5]; but switching the voltage level requires energy as well.

The current best practice is to statically set the voltage level to the maximum of the minimum required voltage level of all peripherals and to not change it dynamically.

Our novel approach addresses the trade-off between the energy gain of a temporarily reduced peripheral voltage and the corresponding switching overhead. We have developed and implemented an algorithm calculating optimal voltage schedules for arbitrary peripheral usage sequences. Yet, the overhead for calculating the schedule is negligible, as the optimal sequence of switching the voltage level is calculated only once during startup.

We address the rather practical issue of measuring the parameters of the involved peripheral components in our implementation as well. The developed prototype is able to self-parameterize every needed variable to run the algorithm. Hence, a fully self-optimizing and generic solution is presented.

The outline of the paper is as follows: Related work is discussed in the next section. Section III formalizes the peripheral voltage scheduling problem and introduces notation. Sections IV and V present our scheduling algorithm and its practical implementation. Our approach is evaluated in Section VI, while Section VII concludes this paper.

II. Related Work

Typical DVS approaches focus on the processing unit only [9], [3]. However, our previous work [5] considers the effects of various voltage levels on the peripherals as well. Even peripherals like Micro-Electro-Mechanical Systems (MEMS), which mainly consist of none-CMOS parts [10], have a reduced energy consumption when exposed to a downscaled voltage level. This result leads to the idea of adapting the peripheral voltage level to the actual sensing requests instead of the system load.

[5] introduces an exclusive scalable voltage path for the peripheral components. Thus, together with a non-negligible switching overhead, the voltage level of the sensors and memory devices can be adapted. In this context, the scheduling problem is to find the best trade-off between reducing the

energy consumption by temporarily lowering the voltage level and the inferred switching costs.

Existing algorithms [11], [12] for voltage scheduling on low-performance sensor nodes are not sufficient for this purpose. In conclusion, they focus on the processing unit only or neglect the energy costs for adapting the voltage level.

To the best of our knowledge, there is no related work addressing the scheduling problem of finding an optimal voltage adaption scheme.

III. The Peripheral Voltage Scheduling Problem

We consider a wireless sensor node with a set of peripheral hardware, typically sensors and external memory. Each peripheral hardware device requires a minimum voltage to be properly operated. To the best of our knowledge, the best practice up to date is to statically configure the lowest peripheral voltage conform to all peripheral devices' voltage requirements. This can be very inefficient, because most hardware consumes more energy when exposed to higher voltage. In this work, we seek to exploit a sensor node's mechanism to dynamically switch the peripheral voltage.

The crux is that switching the voltage does not come for free. If it would, one could simply operate every peripheral device with its minimum required voltage. But switching the voltage consumes energy as well: The additional time interfacing a scalable voltage supply prolongs the duty-cycle of a processing unit, leading to a higher energy consumption. We present a measurement for the switching overhead in Section V.

In this work, we provide an algorithm for peripheral voltage scheduling that weighs off the energetic benefits of switching to a lower peripheral voltage against the switching overhead without violating the minimum voltage requirements of active hardware.

We achieve this by exploiting two observations due to preliminary measurements.

1) Most peripheral hardware devices can be safely undervolted while they are inactive.
2) The energy consumption of each peripheral device increases when the peripheral voltage is increased.

Both observations have been verified in experiments for a wide variety of hardware.

A. Peripheral Energy Consumption

Consider a sensor node with a set S of peripheral hardware devices. In order to assess the benefits of switching to a lower peripheral voltage before using $s \in S$, we need to know how much energy is consumed when querying s using the peripheral voltage v. We denote that relation by the function $e_s(v)$, which may be arbitrarily complicated, see Figure 1. $e_s(v)$ depends on the time t_s necessary to query s, the peripheral voltage v, and the accumulated current $I_s(v,t)$ flowing through s as well as through the inactive peripheral hardware $S \setminus \{s\}$:

$$e_s(v) = v \int_0^{t_s} I_s(v,t)\,\mathrm{d}t \tag{1}$$

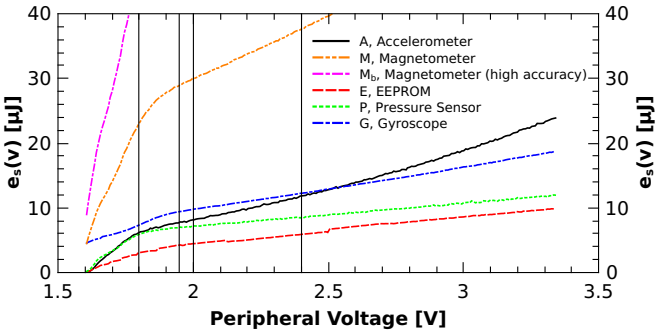

Figure 1. Active and passive energy consumption of the peripheral set of our prototype, compare Table I. The vertical bars at 1.800 [V], 1.950 [V], 2.000 [V], and 2.400 [V] denote the minimum voltages of EEPROM and Pressure Sensor, Magnetometer, Accelerometer, and Gyroscope, respectively.

Moreover, some peripherals support various operating modes, resulting in a different energy consumption as well. As an example in Figure 1 the magnetometer can be used in standard mode M, and with higher accuracy M_b, leading to higher energy consumption $e_{M_b}(v)$. We simply model such varieties as different peripherals $s \in S$.

Instead of attempting to explicitly model $e_s(v)$, we consider it a black box and only assume that it is a monotonically increasing function. In the following subsection, we argue that only a very small set of values for v is of interest for our scheduling problem (only the minimal voltages for the peripherals). That allows us to (1) measure all relevant values of $e_s(v)$, of which there are no more than $|S|^2$, in a self-parameterization phase, compare Section V-A, (2) then run the scheduling algorithm using those values, and (3) run the schedule.

B. Notation

We formalize the peripheral voltage scheduling problem motivated above, using the following model.

Consider a sensor node with a set S of peripheral hardware devices. Each $s \in S$ has two attributes: (1) a minimum voltage $v_{\min}(s)$ required to properly operate s, and (2) the energy consumption $e_s(v)$ of all peripherals while only s is active, depending on the peripheral voltage v, see above. Throughout this work, we assume $e_s(v)$ to be a monotonically increasing function, i. e., that a reduction of the peripheral voltage never results in an increased energy consumption. For a constant amount C of energy, the *switching overhead*, the sensor node can adapt its peripheral voltage.

The sensor node is presented a sequence of *queries* denoted by $[1,\ldots,n]$ to fulfill its sensing and operation tasks. Query i operates the peripheral device $s_i \in S$, thus requiring a minimum peripheral voltage of $v_{\min}(s_i)$ and resulting in an energy consumption of $e_{s_i}(v)$. Note that $s_i = s_j$ for $i \neq j$ is quite common, because the typical application repeatedly operates a handful of sensors over a very long running time; in other words, we usually have $n \gg |S|$. A sequence of peripheral voltages $v(1),\ldots,v(n)$ is called *voltage schedule* for the queries $[1,\ldots,n]$, and we call a pair of query and voltage $(i, v(i))$ a *configuration*. $(i, v(i))$ is *feasible* if $v(i) \geq v_{\min}(s_i)$ and a voltage schedule is feasible if all its configurations are

feasible. The energy consumption E of a voltage schedule is:

$$E = \sum_{i=1}^{n} e_{s_i}(v(i)) + \sum_{i=2}^{n} \begin{cases} C & \text{if } v(i-1) \neq v(i), \\ 0 & \text{otherwise.} \end{cases} \quad (2)$$

Our goal is to minimize E. We call a voltage schedule *optimal* if E is minimal. It follows from the monotonicity of $e_s(v)$ that an optimal schedule only uses $v(i) \in \{v_{\min}(s) \mid s \in S\} = \{V_1, \ldots, V_m\}$ with $V_1 < \ldots < V_m$. As motivated above, there are at most $|S|^2$ device-voltage combinations whose energy consumption can be measured in a self-parameterization phase, compare Section V-A.

Note that the current best practice, i. e., not adapting the peripheral voltage, is equivalent to a trivial schedule that simply keeps the minimum peripheral voltage high enough for all peripheral devices: $v(1) = \ldots = v(n) = \max_{s \in S} v_{\min}(s)$.

IV. Algorithm

Let us, for the configuration (i, V_j), determine the minimum amount of energy $E_{i,j}$ necessary to reach it using a feasible schedule $v(1), \ldots, v(i)$ while assuming an infinite energy consumption for infeasible configurations. For the first query, we have:

$$E_{1,j} = \begin{cases} \infty & \text{if } V_j < v_{\min}(s_1), \\ e_{s_1}(V_j) & \text{otherwise.} \end{cases} \quad (3)$$

For $2 \leq i \leq n$, there is the mandatory energy consumption $e_{s_i}(V_j)$ to answer the query i itself, as well as the accumulated costs for traversing $i-1$ preceding configurations. There are two ways to reach the configuration (i, V_j) with an optimal energy consumption: Either the peripheral voltage from the previous query is kept, or it is changed. The former case yields an additional energy consumption of $E_{i-1,j}$. In the latter case we require the minimum amount of energy $\hat{E}_{i-1}$ to reach the cheapest feasible predecessor configuration and the additional costs C for switching the voltage, where $\hat{E}_{i-1} = E_{i-1,\hat{\jmath}}$ with $\hat{\jmath} := \arg\min_j E_{i-1,j}$. This yields, for $2 \leq i \leq n$:

$$E_{i,j} = \begin{cases} \infty & (i, V_j) \text{ is infeasible,} \\ e_{s_i}(V_j) + E_{i-1,j} & E_{i-1,j} < \hat{E}_{i-1} + C, \\ e_{s_i}(V_j) + \hat{E}_{i-1} + C & \text{otherwise.} \end{cases} \quad (4)$$

Equations (3) and (4) directly translate into Algorithm 1, see Figure 3. It uses dynamic programming to efficiently solve the recursion by determining $E_{i,\cdot}$ before $E_{i+1,\cdot}$; the optimal overall schedule is that ending in the configuration (n, V_j), where $E_{n,j} = \hat{E}_n$ is minimal and the schedule $v(1), \ldots, v(n)$ is determined by following the backward references stored in P, compare Figure 2.

A. Algorithm Extensions

Algorithm 1 processes finite query sequences of fixed order. In the remaining part of this section, we outline how to extend it to determine optimal schedules for periodically repeating queries, how the order of queries can be modified to find more energy-efficient schedules, and how to integrate global voltage limits into Algorithm 1.

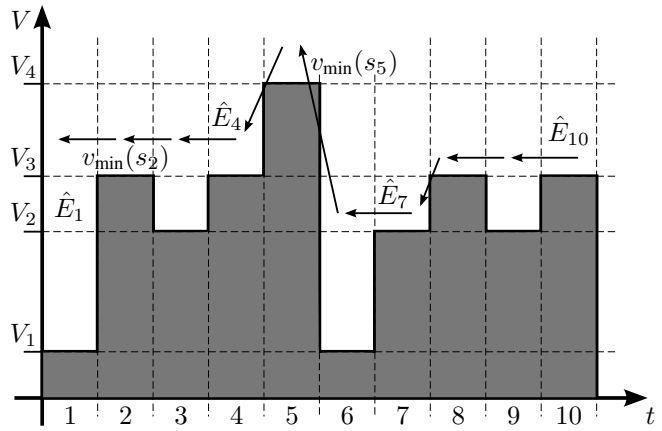

Figure 2. Using dynamic programming to determine an optimal schedule. Here: $(v(1), \ldots, v(10)) = (V_3, V_3, V_3, V_3, V_4, V_2, V_2, V_3, V_3, V_3)$. Note that different queries with the same $v_{\min}(s_i)$, like 3, 7, and 9 may originate from the same sensor $s \in S$.

for $j \leftarrow 1, \ldots, m$ **do**
  $E_{1,j} \leftarrow \begin{cases} \infty & \text{if } V_j < v_{\min}(s_1) \\ e_{s_1}(V_j) & \text{otherwise} \end{cases}$
  $P_{1,j} \leftarrow \perp$
end for
for $i \leftarrow 2, \ldots, n$ **do**
  $\hat{\jmath} \leftarrow \arg\min_{j=1,\ldots,m} E_{i-1,j}$
  for $j \leftarrow 1, \ldots, m$ **do**
    if $V_j < v_{\min}(s_i)$ **then**
      $E_{i,j} \leftarrow \infty$
      $P_{i,j} \leftarrow \perp$
    else if $E_{i-1,j} < E_{i-1,\hat{\jmath}} + C$ **then** is a maximum of
      $E_{i,j} \leftarrow e_{s_i}(V_j) + E_{i-1,j}$
      $P_{i,j} \leftarrow j$
    else
      $E_{i,j} \leftarrow e_{s_i}(V_j) + E_{i-1,\hat{\jmath}} + C$
      $P_{i,j} \leftarrow \hat{\jmath}$
    end if
  end for
end for

Figure 3. Algorithm 1. Determining an optimal schedule using dynamic programming.

1) Periodic Schedules: The most common scenario is not to have a finite sequence of queries, but one that is repeated indefinitely, i. e., until battery depletion. So instead of a query sequence $[1, \ldots, n]$, we have $[1, \ldots, n, 1 \ldots, n, \ldots]$. Algorithm 1 can calculate an optimal schedule for that case as well: Suppose that query i has the strongest voltage requirement of all queries, i. e., it maximizes $v_{\min}(s_i) =: V$. Then any feasible schedule must enter the configuration (i, V) whenever i occurs in the query sequence. We use (i, V) as a fixed point. It is easy to check that an optimal voltage schedule is obtained by providing Algorithm 1 with the query sequence $[1, \ldots, \underline{i, \ldots, n, 1, \ldots, i-1, i}]$ and then periodically repeating the schedule for the underlined queries.

2) Optimal Ordering of Queries: Occasionally, queries do not have to be served in a specific order, e. g. in case of independent data aggregation. This scenario allows us to

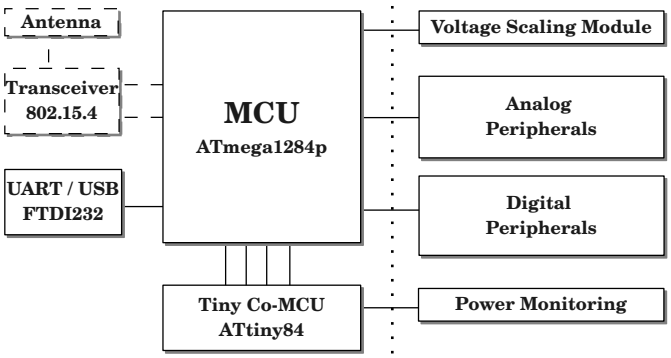

Figure 4. Block diagram of the prototype implementation.

reorder queries and the question arises, which ordering results in the cheapest voltage schedule. We claim that a cheapest optimal schedule is achieved when the queries $[1, \ldots, n]$ are ordered such that, w. l. o. g., $v_{\min}(s_1) \geq \ldots \geq v_{\min}(s_n)$ and outline a proof:

Suppose we know some ordering resulting in a cheapest schedule. Group its queries into segments $S_1, \ldots, S_k$ of maximal length, such that all queries in segment S_i are served with the same voltage $V(S_i)$. For all $i \neq j$ we have $V(S_i) \neq V(S_j)$, because otherwise merging S_i and S_j would result in a cheaper schedule, contradicting optimality. Sorting queries within a segment doesn't change the energy consumption and neither does sorting the segments themselves. Note that every query j belongs to the segment S_i with minimum $V(S_i) \geq v_{\min}(s_j)$: If it didn't, the schedule would not be optimal due to the monotonicity of $e_{s_j}(v)$. Putting it all together proves that ordering the queries by descending minimum voltage yields a schedule with the same, i. e., optimal, costs.

3) Voltage Limits: There might be some lower bound $V_{\min}$ to the voltage of some inactive peripheral that is violated by $v_{\min}(s)$ for some other peripheral $s \in S$. Then Algorithm 1 is easily extended to only use voltages of $V_{\min}$ or higher: Recall that we only use $V_1 < \ldots < V_m$ as voltage candidates and let $V_i = \max\{V_i \mid V_i < V_{\min}\}$. Now Algorithm 1 can just use the voltage candidates $V_{\min} < V_{i+1} < \ldots < V_m$ instead. This even speeds up the process, especially the self-parameterization phase, because less voltages need to be considered, see Section V-A.

V. Implementation

The demand of energy-efficiency leads to limited resources like computational performance. Hence, typical wireless sensor nodes are equipped with ultra low-power (ULP) micro-controllers of small complexity. We use an 8-bit Atmel ATmega1284p micro-controller as processing unit, because it is wide-spread [13], [14], i. e., has practical relevance, and due to its scarce (CPU-) resources, a common limitation in WSNs. So it demonstrates that our approach is sufficiently lightweight for such requirements.

Related to the DVS capable node of [5], we borrow the design of its voltage-scaling module to achieve the ability of a software-defined voltage level for the peripherals: The voltage scaling module is connected to the processing unit via I2C-bus and provides a voltage level of $1.8\,[\mathrm{V}] \leq v \leq 3.3\,[\mathrm{V}]$ with an 8-bit resolution. In this case, the overhead of switching to an arbitrary voltage level is $C \approx 7.76\,[\mu\mathrm{J}]$, which could be further reduced using more sophisticated hardware.

Table I. Prototype peripherals.

Peripheral s	Device	Description	$v_{\min}(s)$ [V]
A	ADXL345	Accelerometer	2.000
E	AT24C08C	EEPROM	1.800
P	BMP085	Pressure Sensor	1.800
G	L3G4200D	Gyroscope	2.400
M	MAG3110	Magnetometer	1.950
M_b	MAG3110	Magnetometer (high accuracy)	1.950

Table II. Energy consumption $e_s(v_{\min}(s'))$ per query for each sensor-voltage pair $(s, v_{\min}(s'))$.

Reference Voltage		Energy Consumption $e_s(v_{\min}(s'))$ [μJ] per Query				
s'	$v_{\min}(s')$ [V]	$s = E$	$s = P$	$s = M$	$s = A$	$s = G$
E, P	$V_1 = 1.800$	4.943	13.140	—	—	—
M	$V_2 = 1.950$	6.106	15.072	282.827	—	—
A	$V_3 = 2.000$	6.363	15.466	292.033	18.962	—
G	$V_4 = 2.400$	8.019	18.915	365.285	25.585	14.391

Our prototype's sensing unit is divided into an analog and a digital section. The analog section offers the ability of connecting fully analog sensors to the ADC channels of the ATmega1284p, while the digital section includes the devices of Table I. All of them are connected via I2C bus.

The basic sensor set is inspired by motion-sensing applications like gait analysis [15] or flight control of quadrocopters [6]. These applications observe various sensors, so our scheduling approach promises to increase their energy efficiency. Nevertheless, in order to expand the set of peripherals for further experiments, the I2C-bus is accessible through a pin header. Figure 4 shows a block diagram of the prototype.

A. Self-Parameterization of $e_s(v)$

As described above, in order to calculate an optimal schedule, we need information describing the overall peripheral energy consumption, which depends on the active peripheral device s and the peripheral voltage v. We denote that by the function $e_s(v)$, compare Equation (1). For this reason, we added a tiny co-micro-controller (co-MCU) to the prototype, which is able to concurrently sample the current consumption of the peripherals (a shunt is used in connection with current sense amplifiers) and to measure the time (the co-MCU can be triggered by the ATmega1284p via digital GPIOs). That design allows us to measure $e_s(v)$ for any given values of s and v.

Together with the predefined minimum voltage level $v_{\min}(s)$ of each peripheral s, the co-MCU is able to determine the energy consumption of all components as depicted in Equation (1): For each $(s, s') \in S \times S$, we measure $e_s(v_{\min}(s'))$, see Table II.

In connection with the subsequent calculation of the schedule, it is sufficient to use raw values instead of the physical units for time, current, and voltage. With regard to limited processing power of wireless sensor nodes, this simplification replaces expensive floating-point arithmetic with cheap integer operations.

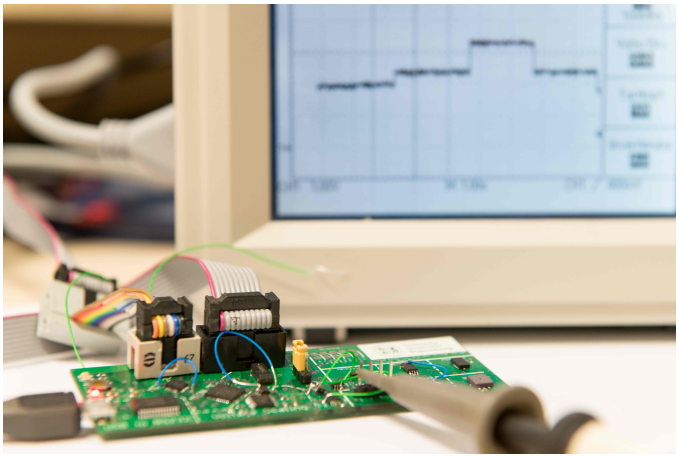

Figure 5. Test-setup of the evaluation.

VI. Evaluation

Figure 5 shows the prototype implementation and the test-setup for evaluating our voltage scheduling concept.

We compare four peripheral voltage handling strategies, three heuristics and Algorithm 1:

1) ConstDefault is what happens when a sensor node has no mechanism to adapt the peripheral voltage. A constant peripheral voltage of 3.3 [V] is kept – a wide-spread default sufficient for most peripheral hardware.
2) ConstMaxMin is the trivial strategy that uses the maximum minimum voltage, i. e., $\max_{s \in S} v_{\min}(s)$, for all queries. It never changes the peripheral voltage. Note that this strategy is, to our best knowledge, the current best practice. It is motivated by the scenario of a statically configured voltage that cannot be adjusted dynamically.
3) AlwaysSwitch, another trivial strategy, which *always*, i. e., for every query with a different voltage requirement than its predecessor, switches the voltage to its minimum requirement. It ignores the switching overhead.
4) Scheduled refers to a schedule computed by Algorithm 1, carefully weighing off the benefits of a lower peripheral voltage against the switching overhead.

In order to test the strategies, we devise various test cases, i. e., query sequences. Each of the above strategies is applied to every test case. We run the resulting voltage schedules 10 times each and measure their average energy consumption. Consult Section IV-A1 on how to use Algorithm 1 to efficiently deal with periodically repeating query sequences.

All test sequences, the resulting peripheral energy consumption of each strategy, as well as the percentage of energy saved by the Scheduled strategy as compared to the other three are presented in Table III. Which character refers to which sensor in a test sequence is specified in Table I, e. g., sequence 9 alternates between the gyroscope and the pressure sensor.

The average peripheral energy consumption for a query sequence is depicted in the center block of Table III. Clearly, the Scheduled strategy, i. e., Algorithm 1, never consumes more energy than any other strategy.

We present the percentage of energy saved by the Scheduled strategy compared to ConstDefault, ConstMaxMin, and AlwaysSwitch in the right block of Table III.

The improvement of Scheduled over ConstDefault is between 31.54 % (sequence 9) and 47.90 % (sequence 10), clearly a significant improvement.

Compared to ConstMaxMin, Scheduled saves up to 20.41 % (sequence 10), the majority of the improvements is comfortably above 17.00 %. As ConstMaxMin is a much better approach than ConstDefault, these still are outstanding improvements. On the other hand, the improvement regarding sequences 5 and 9 is zero within measuring noise. This is explained as follows: For sequences 5 and 9, the optimal schedule is to constantly keep the minimum maximum voltage. The ConstMaxMin strategy does that by chance, Scheduled does it, because it is optimal.

AlwaysSwitch performs better than ConstDefault and, in most cases, better than ConstMaxMin, but Scheduled still saves more energy. Despite the median improvement of Scheduled compared to AlwaysSwitch being between 1 % and 2 %, there are some cases where it is significant: For sequences 8, 5, and 9, Scheduled saves 8.45 %, 10.29 %, and 20.29 %, respectively. There are cases where there is little to no improvement of Scheduled compared to AlwaysSwitch. As above, this is owed to the fact that the AlwaysSwitch strategy happens to provide an optimal or near-optimal schedule.

Concluding, there are three important observations:

1) Much energy is saved by Scheduled compared to the three other strategies.
2) Scheduled never uses *more* energy than any of the other strategies. I. e., there is no situation where using it wastes energy. This is owed to the fact that Algorithm 1 finds an *optimal* schedule within our model, while any other strategy is bound to fail for some cases.
3) Occasionally, Scheduled is just as good as ConstMaxMin or AlwaysSwitch. These are the scenarios where the heuristics ConstMaxMin or AlwaysSwitch happen to find an optimal solution. This does not impair the quality of Scheduled, it rather demonstrates that ConstMaxMin and AlwaysSwitch are worthy competitors.

A final insight of our experiments is the following. In our prototype, the switching overhead is $7.76\,[\mu\mathrm{J}]$, which is quite high compared to peripheral queries requiring between $4.94\,[\mu\mathrm{J}]$ and $365.29\,[\mu\mathrm{J}]$. It could be improved using more sophisticated hardware, compare Section V. But even under those adverse conditions, dynamically switching the voltage saves energy, even with a simple strategy as AlwaysSwitch, even more so with the optimal Scheduled strategy.

Our concluding observation is that the good experimental results stem from the fact that dynamic voltage scheduling

Table III. IMPACT OF VOLTAGE SCHEDULING ON PERIPHERAL ENERGY-EFFICIENCY COMPARED TO CLASSICAL APPROACHES.

		Average Peripheral Energy Consumption [μJ]				Energy saved by SCHEDULED compared to		
#	Query Sequence	CONSTDEFAULT	CONSTMAXMIN	ALWAYSSWITCH	SCHEDULED	CONSTDEFAULT	CONSTMAXMIN	ALWAYSSWITCH
1	AEPGMAEPGM	1321	864	723	716	45.80 %	17.13 %	0.97 %
2	GAMGAMGAM	1872	1215	1013	1008	46.15 %	17.04 %	0.49 %
3	GAMPE	663	432	357	352	46.91 %	18.52 %	1.40 %
4	GAMPEGAMPE	1325	863	725	712	46.26 %	17.50 %	1.79 %
5	GEPGEPGEP	183	123	136	122	33.33 %	0.81 %	10.29 %
6	PEMAG	663	432	358	352	46.91 %	18.52 %	1.68 %
7	PEMAGPEMAG	1328	864	725	712	46.39 %	17.59 %	1.79 %
8	GAPEGAPE	209	133	142	130	37.80 %	2.26 %	8.45 %
9	GPGPGPGPGP	241	165	207	165	31.54 %	0.00 %	20.29 %
10	PAMPE	666	436	356	347	47.90 %	20.41 %	2.53 %
11	APEGAME	720	465	398	385	46.53 %	17.20 %	3.27 %

can save much energy, that we use a good model, and that SCHEDULED, i. e., Algorithm 1, is not a heuristic but provides provably optimal voltage schedules.

A. Limitations

Our method guarantees that active peripherals never are undervolted, but relies on undervolting the inactive ones – otherwise CONSTMAXMIN would be optimal. So a potential drawback of our method is that the measurements of an active peripheral might be impaired by the fact that it has been undervolted in the recent past. We tried to find evidence for such an impairment, but none of our peripherals, see Table I, was in any way compromised by the undervolting.

Should, however, such an effect be observed in an application, e. g., that one sensor must not be undervolted below a certain limit, Section IV-A3 shows how to easily integrate such a limit into Algorithm 1.

VII. CONCLUSION

In this paper, we address sensor nodes capable of dynamically adjusting the peripheral voltage level. We optimize peripheral energy-efficiency by (1) carefully modeling and (2) optimally solving the following optimization problem: Peripheral hardware consumes more energy when exposed to a higher voltage level, each peripheral hardware device requires some minimum voltage to be properly operated, and adjusting the voltage level requires energy as well, the *switching overhead*. Our approach weighs off the benefits of a downscaled voltage level against the switching overhead.

We formulate a mathematical model of the above optimization problem and provide an algorithm finding optimal voltage schedules. The optimality of our schedules is verified with an implementation and a dedicated prototype sensor node.

In experiments we observe energy savings of up to 47 % as compared to a fixed voltage level. The experimental evaluation also confirms the practical performance of our approach, thus verifying both our model and our algorithm. Additionally, due to its optimality, our approach never requires more energy than its alternatives.

Further work includes the generalization of our approach to related scheduling problems, as well as the distribution of the problem: Neighboring sensor nodes may collaborate when observing the environment and thus avoid the switching overhead by sharing their sensor set.

REFERENCES

[1] U. Tietze and C. Schenk, *Electronic Circuits: Handbook for Design and Application*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007.

[2] T. Hamachiyo, Y. Yokota, and E. Okubo, "A cooperative power-saving technique using dvs and dms based on load prediction in sensor networks," in *Sensor Technologies and Applications (SENSORCOMM), 2010 Fourth International Conference on*, july 2010, pp. 7 –12.

[3] L. B. Hoermann, P. M. Glatz, C. Steger, and R. Weiss, "Energy efficient supply of wsn nodes using component-aware dynamic voltage scaling," *Wireless Conference 2011 - Sustainable Wireless Technologies (European Wireless), 11th European*, pp. 1 –8, april 2011.

[4] W. Tuming, Y. Sijia, and W. Hailong, "A dynamic voltage scaling algorithm for wireless sensor networks," in *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on*, vol. 1, aug. 2010, pp. V1–554 –V1–557.

[5] U. Kulau, F. Büsching, and L. C. Wolf, "A node's life: Increasing WSN lifetime by dynamic voltage scaling," in *The 9th IEEE International Conference on Distributed Computing in Sensor Systems 2013 (IEEE DCoSS 2013)*, Cambridge, USA, May 2013. [Online]. Available: http://www.ibr.cs.tu-bs.de/papers/kulau-dcoss2013.pdf

[6] M. Muller, S. Lupashin, and R. D'Andrea, "Quadrocopter ball juggling," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, 2011, pp. 5113–5120.

[7] ST Microelectronic, "L3g4200d mems motion sensor: ultra-stable three-axis digital output gyroscope," 2010. [Online]. Available: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/datasheet/CD00265057.pdf

[8] Texas Instruments, "Msp430f15x, msp430f16x, msp430f161x mixed signal microcontroller," 2002. [Online]. Available: http://www.ti.com/lit/ds/symlink/msp430f1611.pdf

[9] W. Dargie, "Dynamic power management in wireless sensor networks: State-of-the-art," *Sensors Journal, IEEE*, vol. 12, no. 5, pp. 1518 –1528, may 2012.

[10] F. Khoshnoud and C. de Silva, "Recent advances in mems sensor technology x2013; biomedical applications," *Instrumentation Measurement Magazine, IEEE*, vol. 15, no. 1, pp. 8–14, 2012.

[11] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," in *Low Power Electronics and Design, 1998. Proceedings. 1998 International Symposium on*, 1998, pp. 197–202.

[12] Y. Cho, Y. Kim, Y. Joo, K. Lee, and N. Chang, "Simultaneous optimization of battery-aware voltage regulator scheduling with dynamic voltage and frequency scaling," in *Low Power Electronics and Design (ISLPED), 2008 ACM/IEEE International Symposium on*, 2008, pp. 309–314.

[13] Crossbow, "Micaz," 2012. [Online]. Available: http://www.openautomation.net/uploadsproductos/micaz_datasheet.pdf

[14] Atmel, "Avr raven," 2012. [Online]. Available: http://www.atmel.com/Images/doc8117.pdf

[15] J. Chen, K. Kwong, D. Chang, J. Luk, and R. Bajcsy, "Wearable sensors for reliable fall detection," in *Engineering in Medicine and Biology Society, 2005. IEEE-EMBS 2005. 27th Annual International Conference of the*, 2005, pp. 3551 –3554.