Variability-Aware Duty Cycle Scheduling in Long Running Embedded Sensing Systems

Lucas Wanner, Rahul Balani, Sadaf Zahedi, Charwak Apte, Puneet Gupta, and Mani Srivastava
University of California, Los Angeles
wanner@cs.ucla.edu, {rahulb, szahedi, charwak, puneet, mbs}@ee.ucla.edu

Abstract—**Instance and temperature-dependent leakage power variability is already a significant issue in contemporary embedded processors, and one which is expected to increase in importance with scaling of semiconductor technology. We measure and characterize this leakage power variability in current microprocessors, and show that variability aware duty cycle scheduling produces 7.1x improvement in sensing quality for a desired lifetime. In contrast, pessimistic estimations of power consumption leave 61% of the energy untapped, and datasheet power specifications fail to meet required lifetimes by 14%. Finally, we introduce a duty cycle abstraction for TinyOS that allows applications to explicitly specify lifetime and minimum duty cycle requirements for individual tasks, and dynamically adjusts duty cycle rates so that overall quality of service is maximized in the presence of power variability.**

I. Introduction

A common energy management strategy in embedded sensing systems is to operate in a duty-cycled fashion: the system stays in sleep mode for most of the time, and becomes active as a result of user interaction, incoming events, or periodic sensing of their environment [30]. A higher duty cycle rate typically translates into higher quality of service [37]. A system with higher duty cycling may, for example, sample sensors for longer intervals or at higher rates, increasing data quality. A typical application-level goal is to maximize quality of data through higher duty cycles, while meeting a lifetime goal. Typical duty cycles in embedded sensing applications range from below 1% in Car-Park management [1] and CargoNet [22], to greater than 50% in VigilNet [14].

Maximizing duty cycling rates requires knowledge of available energy capacity and power consumption of the hardware in its various modes of operation. Energy and power specifications are typically obtained from the datasheets. Due to the increasing variability in power consumption [4] [16], power specifications are heavely guard-banded [13], leaving a lot of energy potential or sensing quality on the table.

Sources of variability include instance-dependent variations due to scaling of physical dimensions in the semiconductor manufacturing process, environment (e.g. temperature-dependent variation), and aging. In previous work [34], we measured and characterized instance and temperature-dependent power variation for the Atmel SAM3U, a contemporary embedded processor based on an ARM Cortex M3 core. The model for temperature dependence of leakage current is based on the analytical relationship between leakage current and temperature as per BSIM3 [5], and thermal dynamics of a packaged chip [15]. We extend the model to quantify the dependence of leakage current on ambient temperature as well.

This material is based upon work supported by the NSF under awards # CNS-0905580 and CCF-1029030. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF. Lucas Wanner is supported in part by CAPES/Fulbright grant #1892/07-0.

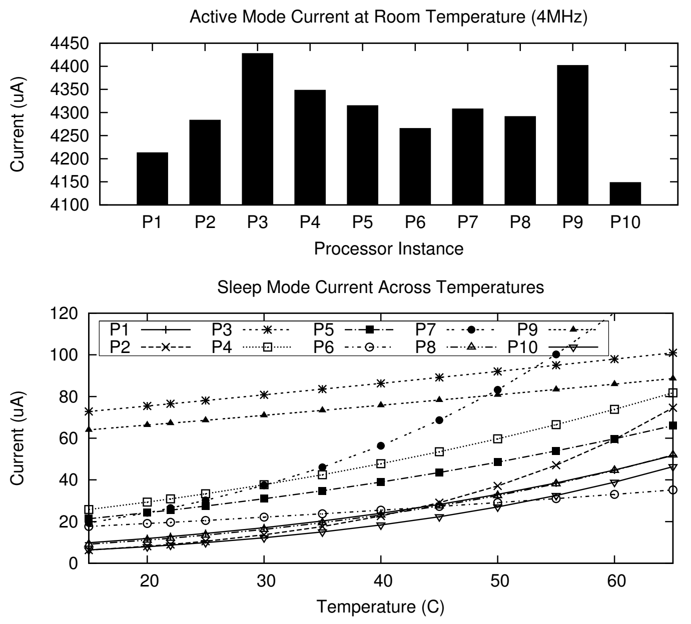

Fig. 1. Active and sleep current for the SAM3U

TABLE I
Variation of current in different modes

Mode	Min	Max	Mean	Std. Dev.	Datasheet
Active (22°C) (mA)	4.1	4.4	4.3	0.1	4.8
Sleep (22°C) (μA)	9	76	28	24	15
Sleep (22–60°C) (μA)	9	120	44	26	-

Figure 1 shows the large variation in sleep power across ten instances of SAM3U, and over a temperature range of 22–60° C that is representative of the temperatures often faced by embedded sensors deployed under unregulated and extreme ambient conditions. While sleep power for any individual processor is monotonic, the magnitudes of temperature based variations differ such that relative rankings of different processors change over temperature. Figure 1 also shows the measured active power, but as expected for this processor, which is in an older technology, variation is less than 10%. Table I summarizes the variation found in the measurements.

In hardware with no power variation, duty cycle rates can be trivially obtained from the power characteristics of the hardware and available energy. With instance and temperature-dependent variability, however, trivially uniform duty cycle schedules based on "one specification fits all" datasheet power characteristics will be unable to meet lifetime requirements due to unforeseen increases in power consumption or, if taking worst-case characteristics into consideration, leave significant energy potential untapped. In this paper, we discuss variability-aware duty cycling methods that handle instance and temperature-dependent variability.

Variation is expected to increase with scaling of silicon technologies, and to become more pronounced in active as well as sleep modes [2]. In face of this variability, finding a duty cycle schedule for a given battery capacity can be formulated as an optimization problem in four-dimensional space: instance characteristics, temperature, supply voltage (if unregulated), and aging. In subsequent discussions, however, we focus only on two aspects with the highest impact on current generation hardware: instance and temperature-dependent variation.

Our contributions are the following: (i) proposal and analysis of variability-aware duty cycle adaptation methods, and (ii) a duty cycle adaptation framework and abstraction for TinyOS, a popular operating system for embedded sensors. Our analysis of duty cycle adaptation methods shows that ignoring instance and temperature-dependent variability in low power embedded sensing systems leads to either untapped energy potential, or unmet lifetime requirements. We also show how the common practice of reactively adjusting work according to recent energy usage observations, or estimations of remaining energy capacity, may lead to sub-optimal quality of service across the lifetime of the system. Finally, our duty cycle abstraction for TinyOS allows applications to explicitly specify lifetime and minimum duty cycle requirements for individual tasks, and dynamically adjusts duty cycle rates according to a variability-aware scheduler so that overall quality of service is maximized.

II. Related Work

Prior work that addresses variability can be classified into (i) statistical design approaches [25] [9] [17], (ii) post silicon compensation and correction [12] [18] [32], and (iii) variation avoidance [8] [3] [11]. Our work differs in that it addresses hardware variability in the operating system layer. The closest resemblance is with [26], which proposes adapting software video codec configurations based on hardware signatures. In embedded sensing context, our work is closest to [23] [10], which propose sensor node deployment methodology based on variability in leakage power across different nodes.

Variations in power consumption can be interpreted as changes in resource (energy) usage (and hence availability). Adaptation of work to resource availability is a common theme in embedded and real-time systems. In imprecise computation [21], each task is designed to produce usable, approximate results whenever resource scarcity (e.g. due to transient failures or overloads) prevents the task from producing its desired precise result. Imprecise computation has been explored in the context of energy-aware systems, where tasks may be interrupted according to energy availability and lifetime requirements [7] [35].

Similarly, several systems have explored the concept of alternative task implementations with different resource usage patterns and quality of service characteristics. Whenever resources are available, tasks with higher quality of service are preferred to those with lower resource usage characteristics. Levels is an energy-aware programming abstraction for TinyOS based on alternative tasks [19]. With this abstraction, programmers define task levels, which provide identical functionality with different quality of service and energy usage characteristics. The run-time system dynamically choses the highest task levels that will meet the required lifetime.

The issue of distributing available energy resources to tasks has also been explored in the literature. ECOSystem [38] introduced the concept of "Currentcy" to allocate energy resources to tasks. The system periodically distributes Currentcy to tasks, which adjust their workload according to availability. Cinder [29] is an energy-aware system for mobile computing devices that features a Capacitor abstraction associated with tasks. Each capacitor represents a task's right to request energy from the system to perform its operations. While we do not deal with the issue of energy distribution to tasks explicitly in this work, we could support schemes like Currentcies or Capacitors by using our variability-aware power consumption model as the source of available energy to these systems.

III. Duty Cycle Scheduling

A duty cycle schedule indicates the activity rate of a system at any point in its lifetime. An optimal duty cycle schedule maximizes the active time of the system across its desired lifetime, given an energy constraint. If there is no variability in power consumption, the optimal duty cycle schedule can be uniform across the lifetime of the system. Given an energy budget of E Joules, a lifetime of L seconds, and invariable constants for active and sleep power consumption P_A and P_S Watts, the maximum allowable allowed duty cycle DC is given in (1). The values of P_A and P_S can be typically obtained from the processor datasheet.

$$P_A \cdot DC + P_S \cdot (1 - DC) = \frac{E}{L}$$
$$DC = \frac{E - L \cdot P_S}{L \cdot P_A - L \cdot P_S} \tag{1}$$

A. Variable Power Consumption

When instance and temperature-dependent variation is taken into consideration, the ***worst-case uniform duty cycle*** can be found by applying the worst-case active and sleep power consumption across all instances and operating temperature range as constants P_A and P_S in (1).

However, with prior characterization, active and sleep power can be expressed as functions of temperature $P_A(T)$ and $P_S(T)$. If the temperature profile is known (or can be learned) for the lifetime of the system, temperature can be expressed as a frequency distribution. For a known operating temperature profile and a given processor instance, the problem of finding an ***optimum duty cycle*** can be formulated as a linear program. Given the expected frequency distribution of (discretized) temperatures across the lifetime of the application, the optimum duty cycle at each temperature T, DC_T is given by (2):

$$\arg\max_{DC_T} \sum_{T=T_{min}}^{T_{max}} DC_T f_T \tag{2}$$

$$\text{s.t.} \sum_{T=T_{min}}^{T_{max}} f_T \cdot (P_A(T) \cdot DC_T + P_S(T) \cdot (1 - DC_T)) \leq \frac{E}{L}$$

$$DC_{min} \leq DC_T \leq DC_{max},\ T_{min} \leq T \leq T_{max}$$

where f_T is the relative frequency of temperature T across the lifetime L, assuming discretized temperature bins. DC_{min} and DC_{max} are the minimum and maximum duty cycles allowed for the application. The maximum duty cycle constraint can be used to limit duty cycles when increasing duty cycle beyond a given rate would bring no further increase to quality of service.

B. Variability-Aware Uniform Duty Cycle

Assuming a uniform duty cycle $DC_T = DC^*$ independent of temperature, we can determine DC^* that satisfies the constraints given in (2).

$$DC^* = \text{Min}\left[\frac{E - L \cdot \sum_{T=T_{min}}^{T_{max}} P_S(T) \cdot f_T}{L \cdot \sum_{T=T_{min}}^{T_{max}} (P_A(T) - P_S(T)) \cdot f_T}, DC_{max}\right] \tag{3}$$

Moreover, it can be shown that when $P_A(T) - P_S(T)$ is constant across all T, DC^* is the ***uniform duty cycle*** that optimizes the linear program in (2). We observed this to be *practically* true for the current generation microprocessors, like the Atmel SAM3U, because (i) their sleep power consumption $P_S(T)$ is much less than active power consumption $P_A(T)$, and (ii) the $P_A(T)$ is effectively constant as active mode leakage power is insignificant for their fabrication technology, and switching power variation across the temperatures is negligible.

C. Reactive Duty Cycle

Allowable duty cycle rates can also be found dynamically through measurements or estimations of past power consumption, given total energy capacity at the start of lifetime. Energy consumption can be directly measured with dedicated monitors [24], inferred from remaining battery capacity [19], or through variability-aware models that estimate energy expenditure by measuring conditions that affect power consumption, e.g. temperature and activity rates.

In a reactive model, duty cycle can be dynamically determined at time t as a ratio of duty cycle at time $t-1$, according to energy spent from time $t-1$ to time t, and remaining energy in the system. Remaining energy at time t is given by $E_t = E - \sum_{i=0}^{t-1} P_i$, where E is the total energy capacity and P_i is power estimated or measured at time i. An example of a ***reactive duty cycle*** adaptation model is given in (4).

$$DC_t = \frac{E_t \cdot DC_{t-1}}{(E_t - E_{t-1}) \cdot (L - t)} \tag{4}$$

The reactive model in (4) assumes that the power consumption rate for the previous time period is indicative of the power consumption for the remainder of lifetime of the system. While more complex models could incorporate longer histories, any reactive model will depend on accurate measurement of past energy consumption or estimation of remaining battery energy.

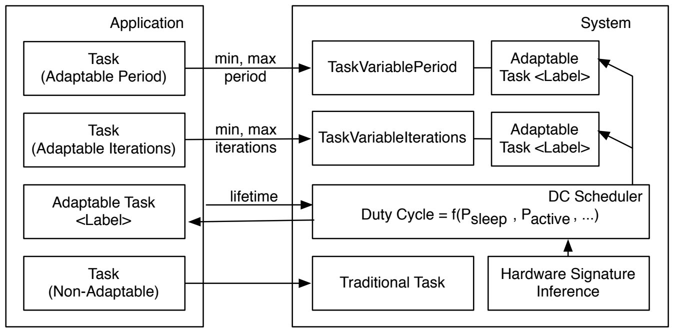

Fig. 2. System Architecture for Variability-Aware DC Scheduling in TinyOS

IV. Duty Cycle Scheduling in TinyOS

In this section we present our design and implementation of a duty cycle scheduling framework and abstraction for TinyOS [20]. TinyOS differs from traditional operating system in that it is *event-based*. Applications respond to events (e.g. interrupts from hardware, incoming radio messages) with event handlers. These handlers should typically complete within a few hundred processor cycles. To execute long running computations, applications post tasks, which work as deferred function calls. Each task runs to completion on a scheduler loop. Whenever the system has no tasks to schedule, it puts the processor in sleep mode, waiting for the next interrupt which will trigger new event handlers, and potentially new tasks. The event handler / background tasks model of TinyOS naturally lends itself to duty cycled systems: event handlers and tasks represent active periods, empty scheduler queues lead to inactive periods. Nevertheless, there's no explicit support for discovering and adapting duty cycle in TinyOS.

We introduce a new *Duty Cycle Scheduler* to TinyOS. Figure 2 shows our system architecture. A hardware signature inference module provides power vs. temperature curves for each processor instance. While in our work we assume that these curves are pre-characterized, extensions to this module could feature online learning through dedicated power meters, and take other variability vectors such as aging into account.

The scheduler determines allowable duty cycle based on: (i) sleep and active power vs. temperature curves provided by the hardware signature inference module, (ii) temperature profile for the application, which can be pre-characterized or learned dynamically, (iii) lifetime requirement specified by the application, (iv) battery capacity, and (v) one of the scheduling methods presented in section III.

To maintain compatibility with existing TinyOS tasks, we introduce *Adaptable Tasks*. These tasks respond to events from the Duty Cycle scheduler that inform them of their current and allowable duty cycle. However, the system does not *enforce* adaptation of these tasks. These are assumed to adapt according to the duty cycle change event from the scheduler.

Adaptable tasks are designed for flexibility. A module using adaptable tasks may, for example, use an alternative function mechanism, e.g. [19]. For standard applications, we provide two additional classes of tasks which implement two common adaptation scenarios: tasks with variable iterations and tasks with variable period. For the first class, the programmer provides a function that can be invoked repeatedly a bounded number of times within each fixed period. For the second class of tasks, the application programmer provides a function representing task functionality that is invoked once within each variable but bounded period of time. Internally, each of these tasks use an adaptable task and unique identifier. The system adjusts the number of iterations or period of the task based on the allowable DC.

V. Evaluation

A. Evaluation Scenario

To evaluate the duty cycle optimization methods, we make use of a common scenario in embedded sensing: a long running duty cycled application with a limited energy source (battery), which periodically becomes active to perform sensing/processing tasks, and subsequently returns to a low-power sleep mode until the next period.

We assume an application which, when active, uses only the main processor running at 4MHz, and when in sleep mode, disables all peripherals except for a low-power wake-up timer. Active and sleep power are obtained from the characterization model and measurements from [34] presented in Sec. I.

For temperature profile, we use hourly temperature data from the National Climactic Data Center for a full year in a fixed location [33], fitted as a triangular distribution. We assume a lifetime requirement of one year for all tests.

B. Comparison of Duty Cycle Scheduling Methods

In this section, we compare duty cycle schedules resulting from the methods discussed in Section III with an energy supply of approximately 850 mAh from AAA batteries. Figure 3 shows the DC schedules across the lifetime of an application for a single instance. It shows that the variability-unaware schedule resulting from the reactive and variability-aware schemes (optimum and uniform) show considerable improvement over the worst-case duty cycle due to untapped energy potential in the latter. The duty cycle resulting from from datasheet power specifications does not meet the required lifetime, as the specification is not guardbanded against temperature-dependent power variation. Having completely pessimistic sleep power specification is very difficult since leakage distribution has a long tail. Table II summarizes the results from the various regimes for all instances. On average, we found a 7.1x improvement in active time with the variability-aware DC over the worst-case DC, 61% of energy potential left untapped by the worst-case DC, and 14% reduction in lifetime with DC based on datasheet specifications.

From the models discussed in Section III, we can observe that the shape of an optimum duty cycle schedule across the lifetime of different processor instances depends on the slope of their respective switching power vs. temperature curves. Since our measurements show no temperature-dependent variation in switching power, the optimum duty cycle schedule is equivalent to the uniform variability-aware schedule in Figure 3. If there were variations in switching power, the optimum DC would no longer be uniform, but instead would have higher duty cycles when switching power is smaller.

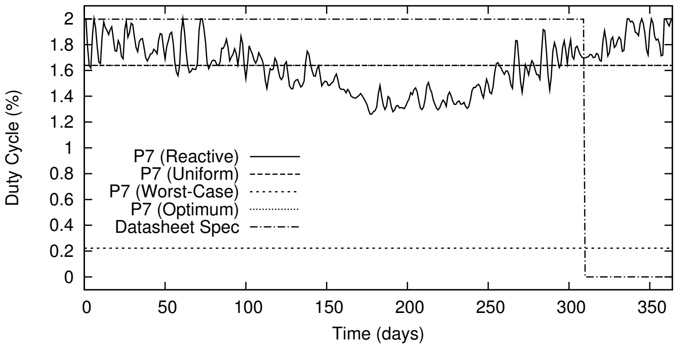

Fig. 3. Schedules resulting from different DC regimes for a single instance. Optimum and uniform DC are overlapping in this figure.

TABLE II
Results from the various DC regimes across all instances

P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	Avg.
Improvement of variability-aware DC over worst-case DC (x)										
8.9	9.1	2.1	6.8	7.4	8	7.3	8.8	3	9.4	7.1
Energy left untapped by worst-case DC (%)										
77	81	11	59	64	70	64	78	20	82	61
Lifetime reduction with DC based on datasheet specification (%)										
2	0	42	19	15	9	16	2	38	0	14

C. Impact of Duty Cycling on Application Performance

Higher duty cycle with a variability-aware optimal algorithm allows the sensors to stay "on" for a longer time and capture more data during deployment. This typically increases accuracy and shortens response times in high fidelity real-time sensing tasks such as object localization and tracking.Figure 4 quantifies the effect of different duty cycles on the efficiency of sound source localization with a network of 20 acoustic (e.g. microphone) sensor arrays using Maximum Likelihood Estimation (MLE). It shows that the algorithm generates estimates of target location with a lower mean error in presence of variability-aware optimal schemes as compared to worst-case schedules for the same battery capacity and target lifetime. For further details on this application, see [37].

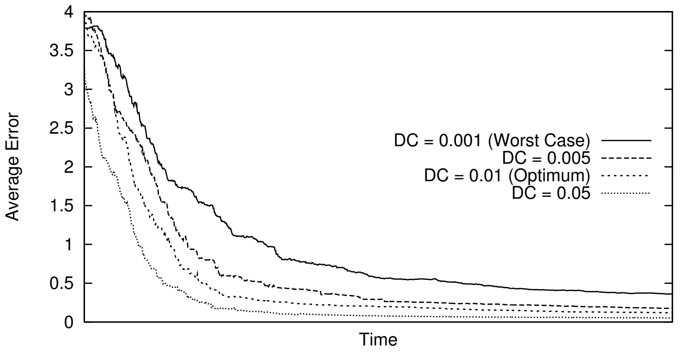

Fig. 4. Mean localization error using MLE. Simulations adapted from [37].

D. Errors in modeling and energy estimation

Adaptation of work based on estimation of remaining energy is a common strategy of the lifetime-aware systems, e.g. [19], [35], [38]. While a reactive duty cycle schedule seems at first glance to eliminate the need for pre-characterization of power vs. temperature curves, practical implementations of such a scheme suffer from limitations in the ability to accurately estimate remaining usable battery capacity. Prior efforts have mainly focused on estimating the state of charge of rechargeable batteries to help users operate their devices judiciously till the next recharge opportunity [6], [28]. Many "smart" batteries use state-of-art "Coulomb-meters" [36] to provide this estimate with an error of 2-5% [28], [36].

Figure 5 shows the effect of errors in the estimation of remaining battery capacity to average lifetime across processor instances, using the same application scenario as Figure 3. In this figure, each time the reactive scheme adapts duty cycle, there's a uniform random error in the estimation of remaining battery capacity, e.g. an actual remaining capacity of 90% with an error of 1% is estimated between 89.1 and 90.9. In target systems with long lifetimes (e.g. greater than a year), where the energy consumed in an hour may be less than 0.01% of total capacity, even small estimation errors make short term adaptation problematic. Figure 5 also shows the effect of errors in modeling of power vs. temperature curves to lifetime. In this graph, errors are modeled as uniform random offsets from the modeled to the actual power vs. temperature curves. This plot shows that errors in modeling of power vs. temperature result in more graceful degradation than errors in reactive adaptation based on remaining capacity, with lifetime reduction proportional to the error in the model.

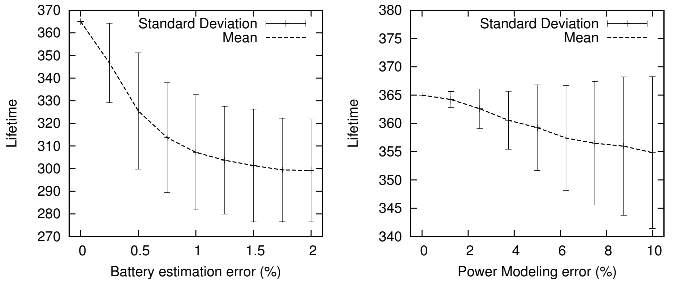

Fig. 5. Effect of errors in remaining battery capacity estimation and temperature vs. power modeling to lifetime

Furthermore, with power variability, previous power consumption may not be a good indicative of future power consumption. This contradicts the assumption stated in Section III-C that forms the basis of Equation 4. Hence, a reactive scheme may be overly optimistic or pessimistic in its assessment of duty cycling rates, resulting in unmet lifetime requirements or untapped energy resources. Finally, if switching power variation is also present, a reactive duty cycling scheme may fail to maximize total active time, as switching power may be lower when total power is higher. Therefore, we believe a priori characterization of power is valuable and may be the option of choice for low cost sensors.

E. Battery Capacity

Figure 6 shows the average percentile improvement of a variability-aware duty cycle schedule (3) over the worst-case duty cycle for different battery capacities. This plot shows that the variability-aware duty cycling regime is more advantageous for applications with smaller duty cycles. For "small" batteries (up to 1100 mAh, corresponding to worst-case duty cycles of up to 1%), improvement is more than 100%. The improvement for "large" batteries (greater than 6600 mAh, corresponding to worst-case duty cycles of more than 14%), improvement is less than 10%.

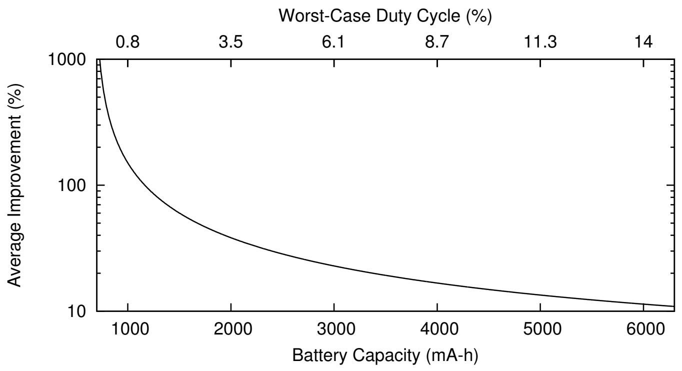

Fig. 6. Average percentile improvement of variability-aware duty cycle over worst-case duty cycle across different battery capacities

Fluctuations in battery capacity due to temperature variation, as well as deviations from the expected temperature profile, may result in reductions of lifetime. In future work, we intend to address this by providing lifetime guarantees with confidence intervals.

F. Active to Sleep Power Ratio and Temperature Range

Figure 7 shows the effect of temperature range and active to sleep power ratio on variability-aware DC (3), when compared to a worst-case duty cycle schedule. To study the effects of increased temperature ranges on duty cycle optimization, we scale the maximum temperature in our temperature distribution by a variable factor, without altering minimum and mode of temperature. Likewise, we scale sleep power by a variable factor to study the effects of different active to sleep power ratios. For each test point in Figure 7, we give each board an energy supply sufficient to achieve a worst-case DC of 10%, and show average improvement across 10 boards with variability-aware DC schedule using the same energy supply. As technology progresses and the ratio between active and sleep power decreases [27], variability-aware duty cycling regime shows considerable benefits even for higher duty cycles over wider temperature ranges.

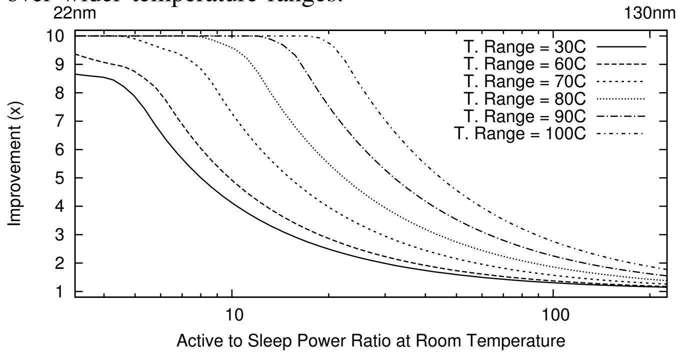

Fig. 7. Avg. improvement of variability-aware DC for all instances over worst-case DC across different temp. ranges and active to sleep power ratios

G. Runtime Overheads

To analyze the overhead of our duty cycle scheduler implementation, we compiled it for the Mica2 mote, which represents the lower end of platforms supported by TinyOS, and profiled it with Avrora, a cycle accurate simulator [31]. Compared to the base TinyOS scheduler, our implementation requires an additional 20 bytes of RAM memory. Each adaptable task instance uses 5 bytes of memory, and the TaskVariablePeriod and TaskVariableIterations abstractions require additional 4 bytes per task. This overhead is well within the capacity of low-end sensor nodes (typically > 4KB RAM).

Compared to basic TinyOS tasks, each adaptable task activation has an overhead of 28μs. As a point of comparison, the reported overhead for each activation in Levels [19] for the same platform, profiled with the same simulator, is 91μs, and the minimum time to acquire a single ADC sample with full precision in this platform is 125μs. Finally, the uniform variability-aware duty cycle control module, which runs periodically every 10 minutes, completes within $250(N+1)\mu$s, where N is the number of adaptable tasks in the system. This time is required to distribute available active time across all adaptable tasks, and to determine the rate of activity (i.e. period, number of iterations) of each task. When the system becomes stable, i.e., when all the tasks reach their allowable duty-cycle, the uniform variability-aware duty cycle control module completes within $250\mu s$. The runtime overhead of our simple duty cycling abstractions is considerably smaller than that of comparable solutions in the literature [19], [38]

VI. Conclusion

In this paper we showed the implication of power variation to duty cycling in low power sensors. We presented variability-aware methods to find optimum duty cycle schedules, and showed that instance and temperature-dependent variability aware duty cycle scheduling gives a 7.1x improvement in the sensing quality, whereas pessimistic estimations of power consumption leaves 61% of the energy untapped, and datasheet power specifications fail to meet required lifetimes by 14%. We introduced an adaptive duty cycling software stack for TinyOS which provides simple abstractions for application adaptation with very small runtime overhead.

As variation in active power becomes more significant, and sleep power becomes comparable in magnitude to active power, power variability will have implications beyond low power sensing applications. In future work we will explore online learning of variability parameters. A combination of pre-characterization and online learning of variability will allow adaptation to dynamic variation due to aging.

References

[1] J. P. Benson et al. Car-park management using wireless sensor networks. In *IEEE Conf. on Local Computer Networks*, 2006.

[2] K. Bernstein et al. High-performance CMOS variability in the 65-nm regime and beyond. *IBM J. Res. Dev.*, 50(4):433–449, 2006.

[3] S. Bhunia, S. Mukhopadhyay, and K. Roy. Process variations and process-tolerant design. In *Intl. Conf. on VLSI Design*, 2007.

[4] S. Borkar et al. Parameter variations and impact on circuits and microarchitecture. In *Design Automation Conf. (DAC)*, 2003.

[5] BSIM. http://www-device.eecs.berkeley.edu/∼bsim3/.

[6] R. Casas and O. Casas. Battery sensing for energy-aware system design. *IEEE Computer*, 38(11):48 – 54, Nov 2005.

[7] L.N. Chakrapani et al. Ultra-Efficient (Embedded) SOC Architectures based on Probabilistic CMOS (PCMOS) Technology. In *DATE*, 2006.

[8] S. H. Choi, B. C. Paul, and K. Roy. Novel sizing algorithm for yield improvement under process variation in nanometer technology. In *Design Autoion Conf. (DAC)*, 2004.

[9] A. Datta et al. Statistical modeling of pipeline delay and design of pipeline under process variation to enhance yield in sub-100nm technologies. In *DATE*, 2005.

[10] S. Garg and D. Marculescu. On the impact of manufacturing process variations on the lifetime of sensor networks. In *Intl. Conf. on Hardware/software codesign and system synthesis (CODES/ISSS)*, 2007.

[11] S. Ghosh, S. Bhunia, and K. Roy. A new paradigm for low-power, variation-tolerant circuit synthesis using critical path isolation. In *Intl. Conf. on Computer-aided design (ICCAD)*, 2006.

[12] Justin Gregg and Tom W. Chen. Post silicon power/performance optimization in the presence of process variations using individual well-adaptive body biasing. *IEEE Trans. VLSI Syst.*, 15(3), 2007.

[13] P. Gupta, A. B. Kahng, and S. Muddu. Quantifying error in dynamic power estimation of cmos circuits. *Analog Integrated Circuits Signal Process.*, 42(3), 2005.

[14] T. He et al. Achieving Real-time Target Tracking Using Wireless Sensor Networks. *IEEE RTAS*, 2006.

[15] H. Huang, G. Quan, and J. Fan. Leakage temperature dependency modeling in system level analysis. In *ISQED*, 2010.

[16] ITRS. The international technology roadmap for semiconductors. http://public.itrs.net/.

[17] K. Kang, B. C. Paul, and K. Roy. Statistical timing analysis using levelized covariance propagation considering systematic and random variations of process parameters. *ACM Trans. Des. Autom. Electron. Syst.*, 11(4):848–879, 2006.

[18] V. Khandelwal and A. Srivastava. Variability-driven formulation for simultaneous gate sizing and post-silicon tunability allocation. In *Intl. Sym. on Physical Design (ISPD)*, 2007.

[19] A. Lachenmann, P. Marrón, D. Minder, and K. Rothermel. Meeting lifetime goals with energy levels. In *ACM SenSys*, 2007.

[20] P. Levis et al. TinyOS: An operating system for sensor networks. *Ambient Intelligence*, pages 115–148, 2005.

[21] J. W. S. Liu, W.-K. Shih, K.-J. Lin, R. Bettati, and J.-Y. Chung. Imprecise computations. *Proc. of the IEEE*, 82(1):83–94, 1994.

[22] Mateusz Malinowski et al. CargoNet: a low-cost micropower sensor node exploiting quasi-passive wakeup for adaptive asychronous monitoring of exceptional events. In *ACM SenSys*, 2007.

[23] T. Matsuda et al. A power-variation model for sensor node and the impact against life time of wireless sensor networks. In *ICCE*, 2006.

[24] D. McIntire et al. The low power energy aware processing (LEAP) embedded networked sensor system. In *IPSN*, pages 449–457, 2006.

[25] O. Neiroukh and X. Song. Improving the process-variation tolerance of digital circuits using gate sizing and statistical techniques. In *Conf. on Design, Automation and Test in Europe (DATE)*, 2005.

[26] A. Pant, P. Gupta, and M. van der Schaar. Software adaptation in quality sensitive applications to deal with hardware variability. In *IEEE Great Lakes Sym. on VLSI (GLSVLSI)*, pages 85–90, 2010.

[27] R. Puri, L. Stok, and S. Bhattacharya. Keeping hot chips cool. In *Proc. Design Automation Conf. (DAC)*, pages 285–288, 2005.

[28] P. Rong and M. Pedram. An analytical model for predicting the remaining battery capacity of lithium-ion batteries. In *DATE*, 2003.

[29] S. M. Rumble, R. Stutsman, P. Levis, D. Mazières, and N. Zeldovich. Apprehending joule thieves with Cinder. In *MobiHeld*, 2009.

[30] T. Schmid, R. Shea, Z. Charbiwala, J. Friedman, Y. Cho, and M. Srivastava. On the interaction of clocks, power, and synchronization in duty-cycled embedded sensor nodes. *ACM Trans. on Sensor Net.*, 2010.

[31] B. L. Titzer, D. K. Lee, and J. Palsberg. Avrora: scalable sensor network simulation with precise timing. In *IPSN*, 2005.

[32] J. Tschanz et al. Adaptive body bias for reducing impacts of die-to-die and within-die parameter variations on microprocessor frequency and leakage. In *Intl. Solid-State Circuits Conference (ISSCC)*, 2002.

[33] U.S. Climate Reference Network (USCRN). Hourly temperature data for Stovepipe Wells, CA. www.ncdc.noaa.gov/crn/, 2009.

[34] L. Wanner, C. Apte, R. Balani, P. Gupta, and M. Srivastava. A case for opportunistic embedded sensing in presence of hardware power variability. In *Wksp. on Power Aware Comp. and Sys. (HotPower)*, 2010.

[35] G. Wiedenhoft, L. Wanner, G. Gracioli, and A. Fröhlich. Power management in the EPOS system. *Oper. Syst. Rev.*, 42(6), 2008.

[36] M. Yu, Y. Barsukov, and M. Vega. Theory and Implementation of Impedance Track^{TM} Battery Fuel-Gauging Algorithm in bq2750x Family. In *Texas Instruments Application Report - SLUA450*, Jan 2008.

[37] S. Zahedi, M.B. Srivastava, C. Bisdikian, and L.M. Kaplan. Quality Tradeoffs in Object Tracking with Duty-Cycled Sensor Networks. In *Real-Time Systems Symp. (RTSS)*, 2010.

[38] H. Zeng, C. S. Ellis, Alvin R. L., and A. Vahdat. ECOSystem: managing energy as a first class operating system resource. *SIGOPS Oper. Syst. Rev.*, 36(5), 2002.