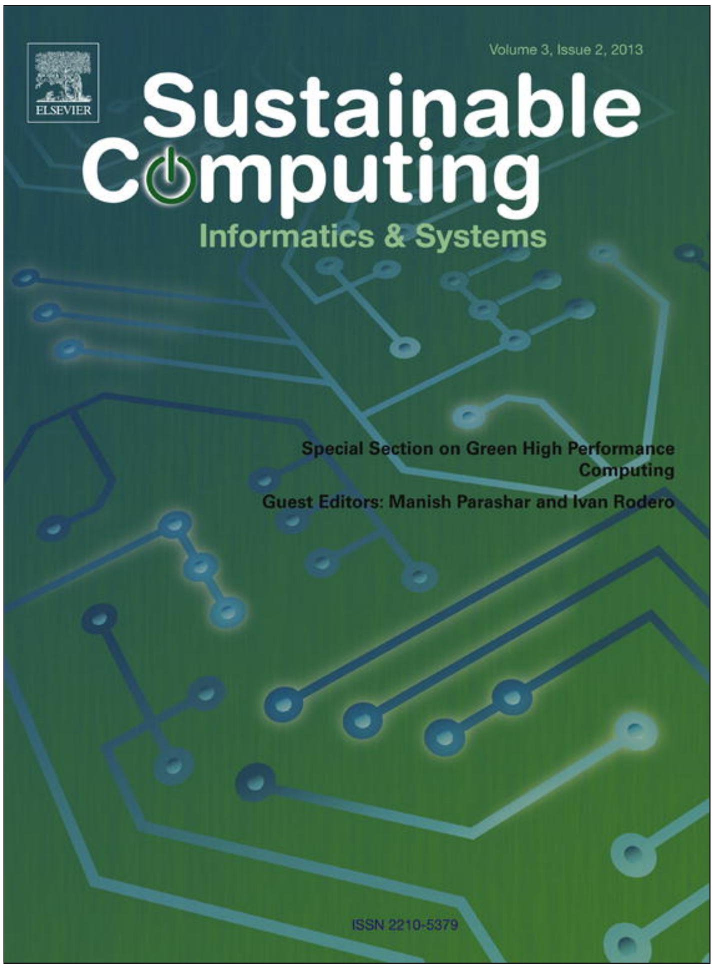
Provided for non-commercial research and education use.
Not for reproduction, distribution or commercial use.


Volume 3, Issue 2, 2013
ELSEVIER
Sustainable Computing
Informatics & Systems
Special Section on Green High Performance Computing
Guest Editors: Manish Parashar and Ivan Rodero
ISSN 2210-5379

This article appeared in a journal published by Elsevier. The attached copy is furnished to the author for internal non-commercial research and education use, including for instruction at the authors institution and sharing with colleagues.

Other uses, including reproduction and distribution, or selling or licensing copies, or posting to personal, institutional or third party websites are prohibited.

In most cases authors are permitted to post their version of the article (e.g. in Word or Tex form) to their personal website or institutional repository. Authors requiring further information regarding Elsevier's archiving and manuscript policies are encouraged to visit:

http://www.elsevier.com/authorsrights

Author's personal copy

Sustainable Computing: Informatics and Systems 3 (2013) 94–108

Contents lists available at SciVerse ScienceDirect

Sustainable Computing: Informatics and Systems

journal homepage: www.elsevier.com/locate/suscom

A lightweight dynamic optimization methodology and application metrics estimation model for wireless sensor networks[☆]

Arslan Munir[a,*], Ann Gordon-Ross[a,1], Susan Lysecky[b], Roman Lysecky[b]

[a] *University of Florida, Gainesville, USA*
[b] *University of Arizona, Tucson, USA*

ARTICLE INFO

Article history:
Received 18 May 2012
Accepted 28 January 2013

Keywords:
Dynamic optimization
Greedy algorithm
Application metrics estimation
Wireless sensor networks

ABSTRACT

Technological advancements in embedded systems due to Moore's law have led to the proliferation of wireless sensor networks (WSNs) in different application domains (e.g., defense, health care, surveillance systems) with different application requirements (e.g., lifetime, reliability). Many commercial-off-the-shelf (COTS) sensor nodes can be specialized to meet these requirements using tunable parameters (e.g., processor voltage and frequency) to specialize the operating state. Since a sensor node's performance depends greatly on environmental stimuli, dynamic optimizations enable sensor nodes to automatically determine their operating state in situ. However, dynamic optimization methodology development given a large design space and resource constraints (memory and computational) is an extremely challenging task. In this paper, we propose a lightweight dynamic optimization methodology that intelligently selects initial tunable parameter values to produce a high-quality initial operating state in *one-shot* for time-critical or highly constrained applications. Further operating state improvements are made using an efficient greedy exploration algorithm, achieving optimal or near-optimal operating states while exploring only 0.04% of the design space on average. We also propose an application metrics estimation model, which is leveraged by our dynamic optimization methodology, to estimate high-level application metrics (e.g., lifetime, throughput) from sensor node tunable parameters and hardware specific internals.

© 2013 Elsevier Inc. All rights reserved.

1. Introduction and motivation

Advancements in semiconductor technology, as predicted by Moore's law, have enabled high transistor density in a small chip area resulting in the miniaturization of embedded systems (e.g., sensor nodes). Wireless sensor networks (WSNs) are envisioned as ubiquitous computing systems, which are proliferating in many application domains (e.g., defense, health care, surveillance systems) each with varying application requirements that can be defined by high-level *application metrics* (e.g., lifetime, reliability). However, the diversity of WSN application domains makes it difficult for commercial-off-the-shelf (COTS) sensor nodes to meet these application requirements.

Since COTS sensor nodes are mass-produced to optimize cost, many COTS sensor nodes possess tunable parameters (e.g., processor voltage and frequency, sensing frequency), whose values can be *tuned* for application specialization [18]. The WSN application designers (those who design, manage, or deploy the WSN for an application) are typically biologists, teachers, farmers, and household consumers that are experts within their application domain, but have limited technical expertise. Given the large design space and operating constraints, determining appropriate parameter values (operating state) can be a daunting and/or time consuming task for non-expert application managers. Typically, sensor node vendors assign initial generic tunable parameter value settings, however, no one tunable parameter value setting is appropriate for all applications. To assist the WSN managers with parameter tuning to best fit the application requirements, an automated parameter tuning process is required.

Parameter optimization is the process of assigning appropriate (optimal or near-optimal) tunable parameter value settings to meet application requirements. Parameter optimizations can be static or dynamic. *Static optimizations* assign parameter values at deployment and these values remain fixed for the lifetime of the sensor node. One of the challenges associated with static optimizations is

[☆] Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee. © 2012 ACM 0000-0000/2012/0000-0001 $5.00.

[*] Corresponding author.
E-mail address: arslan@rice.edu (A. Munir).

[1] Also with the NSF Center for High-Performance Reconfigurable Computing (CHREC) at the University of Florida, Gainesville, FL 32611, USA.

2210-5379/$ – see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.suscom.2013.01.003

Author's personal copy

accurately determining the tunable parameter value settings using environmental stimuli prediction/simulation. Furthermore, static optimizations are not appropriate for applications with varying environmental stimuli. Alternatively, *dynamic optimizations* assign (and re-assign/change) parameter values during runtime enabling the sensor node to adapt to changing environmental stimuli, and thus more accurately meet application requirements.

WSN dynamic optimizations present additional challenges as compared to traditional processor or memory (cache) dynamic optimizations because sensor nodes have more tunable parameters and a larger design space. The dynamic profiling and optimization (DPOP) project aims to address these challenges and complexities associated with sensor-based system design through the use of automated optimization methods [6]. The DPOP project has gathered dynamic profiling statistics from a sensor-based system, however, the parameter optimization process has not been addressed.

In this paper, we investigate parameter optimization using dynamic profiling data already collected from the platform. We analyze several dynamic optimization methods and evaluate algorithms that provide a good operating state without significantly depleting the battery energy. We explore a large design space with many tunable parameters and values, which provide a fine-grained design space, enabling sensor nodes to more closely meet application requirements as compared to smaller, more course-grained design spaces. Gordon-Ross et al. [8] showed that finer-grained design spaces contain interesting design alternatives and result in increased benefits in the cache subsystem (though similar trends follow for other subsystems). However, the large design space exacerbates optimization challenges, taking into consideration a sensor node's constrained memory and computational resources. Considering the sensor node's limited battery life, energy-efficient computing is always of paramount significance. Therefore, optimization algorithms that conserve energy by minimizing design space exploration to find a good operating state are critical, especially for large design spaces and highly constrained systems. Additionally, rapidly changing application requirements and environmental stimuli coupled with limited battery reserves necessitates a highly responsive and low overhead methodology.

Our main contributions in this paper are:

- We propose a lightweight dynamic optimization methodology that intelligently selects appropriate initial tunable parameter value settings by evaluating application requirements, the relative importance of these requirements with respect to each other, and the magnitude in which each parameter effects each requirement. This *one-shot* operating state obtained from appropriate initial parameter value settings provides a high-quality operating state with minimal design space exploration for highly constrained applications. Results reveal that the one-shot operating state is within 8% of the optimal operating state averaged over several different application domains and design spaces.
- We present a dynamic optimization methodology to iteratively improve the one-shot operating state to provide an optimal or near-optimal operating state for less constrained applications. Our dynamic optimization methodology combines the initial tunable parameter value settings with an intelligent exploration ordering of tunable parameter values and an exploration arrangement of tunable parameters (since some parameters are more critical for an application than others and thus should be explored first [31] (e.g., the transmission power parameter may be more critical for a lifetime-sensitive application than processor voltage)).
- We architect a lightweight online greedy algorithm that leverages intelligent parameter arrangement to iteratively explore the design space, resulting in an operating state within 2% of the optimal operating state while exploring only 0.04% of the design space.
- We for the first time, to the best of our knowledge, propose an *application metrics estimation model* that estimates high-level application metrics from low-level sensor node tunable parameters and the sensor node's hardware internals (e.g., transceiver voltage, transceiver receive current). Our dynamic optimization methodology leverages this estimation model while comparing different operating states for optimization purposes.

Our research has a broad impact on WSN design and deployment. Our work enables non-expert application managers to leverage our dynamic optimization methodology to automatically tailor the sensor node tunable parameters to best meet the application requirements with little design time effort. Our proposed methodology is suitable for all WSN applications ranging from highly constrained to highly flexible applications. The one-shot operating state provides a good operating state for highly constrained applications, whereas greedy exploration of the parameters provides improvement over the one-shot operating state to determine a high-quality operating state for less constrained applications. Our initial parameter value settings, parameter arrangement, and exploration ordering techniques are also applicable to other systems or application domains (e.g., cache tuning) with different application requirements and different tunable parameters. Our application metrics estimation model provides a first step toward high-level metrics estimation from sensor node tunable parameters and hardware internals. The estimation model establishes a relationship between sensor node operating state and high-level metrics. Since application managers typically focus on high-level metrics and are generally unaware of low-level sensor node internals, this model provides an interface between the application manager and the sensor node internals. Additionally, our model can potentially spark further research in application metrics estimation for WSNs.

The remainder of this paper is organized as follows. Section 2 overviews the related work. Section 3 presents our dynamic optimization methodology along with the state space and objective function formulation. We describe our dynamic optimization methodology's steps and algorithms in Section 4. Section 5 describes our application metrics estimation model that is leveraged by our dynamic optimization methodology. Experimental results are presented in Section 6. Finally, Section 7 concludes the paper and discusses future research work directions.

2. Related work

There exists much research in the area of dynamic optimizations [9–11,31], but most previous work targets the processor or memory (cache) in computer systems. There exists little previous work on WSN dynamic optimization, which presents more challenges given a unique design space, design constraints, platform particulars, and external influences from the WSN's operating environment.

In the area of dynamic profiling and optimization, Sridharan et al. [26] dynamically profiled a WSN's operating environment to gather profiling statistics, however, they did not describe a methodology to leverage these profiling statistics for dynamic optimization. Shenoy et al. [25] investigated profiling methods for dynamically monitoring sensor-based platforms with respect to network traffic and energy consumption, but did not explore dynamic optimizations. In prior work, Munir et al. [18,19] proposed a Markov Decision Process (MDP)-based methodology for optimal

Author's personal copy

sensor node parameter tuning to meet application requirements as a first step toward WSN dynamic optimization. The MDP-based methodology required high computational and memory resources for large design spaces and needed a high-performance base station node (sink node) to compute the optimal operating state for large design spaces. The operating states determined at the base station were then communicated to the other sensor nodes. The high resource requirements made the MDP-based methodology infeasible for autonomous dynamic optimization for large design spaces given the constrained resources of individual sensor nodes. Kogekar et al. [14] proposed dynamic software reconfiguration to adapt software to new operating conditions, however, their work did not consider sensor node tunable parameters and application requirements. Verma [27] investigated simulated annealing (SA) and particle swarm optimization (PSO)-based parameter tuning for WSNs and observed that SA performed better than PSO because PSO often quickly converged to local minima. Although there exists work on optimization of WSNs, our work uses multi-objective optimization and fine-grained design space to find optimal (or near-optimal) sensor node operating states that meet application requirements.

One of the prominent dynamic optimization techniques for reducing energy consumption is dynamic voltage and frequency scaling (DVFS). Several previous works explored DVFS in WSNs. Min et al. [17] utilized a voltage scheduler, running in tandem with the operating system's task scheduler, to perform DVFS based on an *a priori* prediction of the sensor node's workload, and resulted in a 60% reduction in energy consumption. Similarly, Yuan et al. [30] used additional transmitted data packet information to select appropriate processor voltage and frequency values. Although DVFS is a method for dynamic optimization, DVFS considers only two sensor node tunable parameters (processor voltage and frequency). In this paper, we expand the sensor node parameter tuning space, which provides a finer-grained design space, enabling sensor nodes to more closely meet application requirements.

Some dynamic optimization work utilized dynamic power management for energy conservation in WSNs. Wang et al. [28] proposed a strategy for optimizing mobile sensor node placement to meet coverage and energy requirements. Their strategy utilized dynamic power management to optimize the sensor nodes' sleep state transitions for energy conservation. Ning et al. [22] presented a link layer dynamic optimization approach for energy conservation in WSNs by minimizing the idle listening time. Their approach utilized traffic statistics to optimally control the receiver sleep interval. In our work, we incorporate energy conservation by switching the sensors, processors, and transceivers to low power, idle modes when these components are not actively sensing, processing, and communicating, respectively.

Even though there exists work on WSN optimizations, dynamic optimization requires further research. Specifically, there is a need for lightweight dynamic optimization methodologies for sensor node parameter tuning considering a sensor node's limited energy and storage. Furthermore, sensor node tunable parameter arrangement and exploration order requires further investigation. Our work provides contribution to the dynamic optimization of WSNs by proposing a lightweight dynamic optimization methodology for WSNs in addition to a sensor node's tunable parameters arrangement and exploration order techniques.

3. Dynamic optimization methodology

In this section, we give an overview of our dynamic optimization methodology along with the state space and objective function formulation for the methodology.

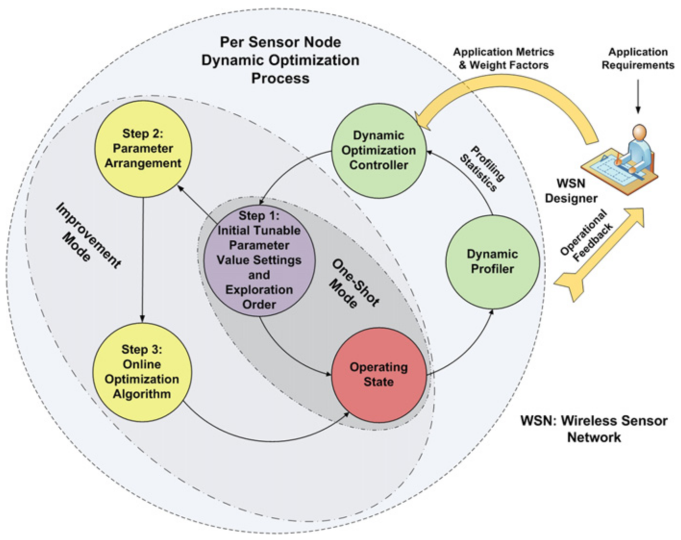

Per Sensor Node Dynamic Optimization Process
Application Metrics & Weight Factors
Application Requirements
Step 2: Parameter Arrangement
Dynamic Optimization Controller
Profiling Statistics
WSN Designer
Improvement Mode
Step 1: Initial Tunable Parameter Value Settings and Exploration Order
One-Shot Mode
Dynamic Profiler
Operational Feedback
Step 3: Online Optimization Algorithm
Operating State
WSN: Wireless Sensor Network

Fig. 1. Our dynamic optimization methodology per sensor node for WSNs.

3.1. Overview

Fig. 1 depicts our dynamic optimization methodology for WSNs. WSN designers evaluate application requirements and capture these requirements as high-level *application metrics* (e.g., lifetime, throughput, reliability) and associated *weight factors*. The weight factors signify the weightage/importance of application metrics with respect to each other. The sensor nodes use application metrics and weight factors to determine an appropriate operating state (tunable parameter value settings) by leveraging an application metrics estimation model. The application metrics estimation model estimates high-level application metrics from low-level sensor node parameters and sensor node hardware-specific internals (Section 5 discusses our application metrics estimation model in detail).

Fig. 1 shows the per sensor node dynamic optimization process (encompassed by the dashed circle), which is orchestrated by the *dynamic optimization controller*. The process consists of two operating modes: the *one-shot mode* wherein the sensor node operating state is directly determined by initial parameter value settings and the *improvement mode* wherein the operating state is iteratively improved using an online optimization algorithm. The dynamic optimization process consists of three steps. In the first step corresponding to the one-shot mode, the dynamic optimization controller intelligently determines the initial parameter value settings (operating state) and exploration order (ascending or descending), which is critical in reducing the number of states explored in the third step. In the one-shot mode, the dynamic optimization process is complete and the sensor node transitions directly to the operating state specified by the initial parameter value settings. The second step corresponds to the improvement mode, which determines the parameter arrangement based on application metric weight factors (e.g., explore processor voltage then frequency then sensing frequency). This parameter arrangement reduces the design space exploration time using an optimization algorithm in the third step to determine a good quality operating state. The third step corresponds to the improvement mode and invokes an *online optimization algorithm* for parameter exploration to iteratively improve the operating state to more closely meet application requirements as compared to the one-shot's operating state. The online optimization algorithm leverages the intelligent initial parameter value settings, exploration order, and parameter arrangement.

Author's personal copy

A *dynamic profiler* records profiling statistics (e.g., processor voltage, wireless channel condition, radio transmission power) given the current operating state and environmental stimuli and passes these profiling statistics to the dynamic optimization controller. The dynamic optimization controller processes the profiling statistics to determine whether the current operating state meets the application requirements. If the application requirements are not met, the dynamic optimization controller reinvokes the dynamic optimization process to determine a new operating state. This feedback process continues to ensure that the application requirements are best met under changing environmental stimuli. We point out that our current work describes the dynamic optimization methodology, however, incorporation of profiling statistics to provide feedback is part of our future work.

3.2. State space

The state space S for our dynamic optimization methodology given N tunable parameters is defined as:

$$S = P_1 \times P_2 \times \cdots \times P_N \tag{1}$$

where P_i denotes the state space for tunable parameter i, $\forall\, i \in \{1, 2, \ldots, N\}$ and $\times$ denotes the Cartesian product. Each tunable parameter's state space P_i consists of n values:

$$P_i = \{p_{i_1}, p_{i_2}, p_{i_3}, \ldots, p_{i_n}\} \quad |P_i| = n \tag{2}$$

where $|P_i|$ denotes the tunable parameter i's state space cardinality (the number of tunable values in P_i). S is a set of n-tuples formed by taking one tunable parameter value from each tunable parameter. A single n-tuple $s \in S$ is given as:

$$\begin{aligned} s = (p_{1_y}, p_{2_y}, \ldots, p_{N_y}) : \; p_{i_y} \in P_i, \\ \forall\, i \in \{1, 2, \ldots, N\},\; y \in \{1, 2, \ldots, n\} \end{aligned} \tag{3}$$

Each n-tuple represents a sensor note operating state. We point out that some n-tuples in S may not be feasible (such as invalid combinations of processor voltage and frequency) and can be regarded as *do not care* tuples.

3.3. Optimization objective function

The sensor node dynamic optimization problem can be formulated as:

$$\begin{aligned} \max \quad & f(s) \\ s.t. \quad & s \in S \end{aligned} \tag{4}$$

where $f(s)$ represents the objective function and captures application metrics and weight factors, and is given as:

$$\begin{aligned} f(s) \;=\; & \sum_{k=1}^{m} \omega_k f_k(s) \\ s.t. \quad & s \in S \\ & \omega_k \geq 0, \quad k = 1, 2, \ldots, m. \\ & \omega_k \leq 1, \quad k = 1, 2, \ldots, m. \\ & \sum_{k=1}^{m} \omega_k = 1 \end{aligned} \tag{5}$$

where $f_k(s)$ and ω_k denote the objective function and weight factor for the kth application metric, respectively, given that there are m application metrics.

For our dynamic optimization methodology, we consider three application metrics ($m = 3$): lifetime, throughput, and reliability, whose objective functions are represented by $f_l(s)$, $f_t(s)$, and $f_r(s)$, respectively. We define $f_l(s)$ (Fig. 2) using the piecewise linear function:

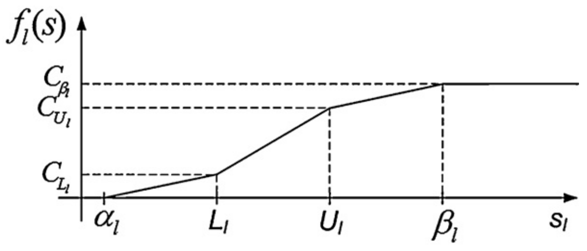


Fig. 2. Lifetime objective function $f_l(s)$.

$$f_l(s) = \begin{cases} 1, & s_l \geq \beta_l \\ C_{U_l} + \dfrac{(C_{\beta_l} - C_{U_l})(s_l - U_l)}{(\beta_l - U_l)}, & U_l \leq s_l < \beta_l \\ C_{L_l} + \dfrac{(C_{U_l} - C_{L_l})(s_l - L_l)}{(U_l - L_l)}, & L_l \leq s_l < U_l \\ C_{L_l} \cdot \dfrac{(s_l - \alpha_l)}{(L_l - \alpha_l)}, & \alpha_l \leq s_l < L_l \\ 0, & s_l < \alpha_l \end{cases} \tag{6}$$

where s_l denotes the lifetime offered by state s, the constant parameters L_l and U_l denote the *desired* minimum and maximum lifetime, and the constant parameters α_l and β_l denote the *acceptable* minimum and maximum lifetime. The piecewise linear objective function provides WSN designers with a flexible application requirement specification, as it allows both desirable and acceptable ranges [16]. The objective function reward gradient (slope) would be greater in the desired range than the acceptable range, however, there would be no reward/gain for operating outside the acceptable range. The constant parameters C_{L_l}, C_{U_l}, and C_{β_l} in (6) denote the $f_l(s)$ value at L_l, U_l, and β_l, respectively (constant parameters are assigned by the WSN designer based on the minimum and maximum acceptable/desired values of application metrics). The $f_t(s)$ and $f_r(s)$ can be defined similar to (6).

The objective function characterization enables the reward/gain calculation from operating in a given state based on the high-level metric values offered by the state. Although different characterization of objective functions results in different reward values from different states, our dynamic optimization methodology selects a high-quality operating state from the design space to maximize the given objective function value. We consider piecewise linear objective functions as a typical example from the possible objective functions (e.g., linear, piecewise linear, non-linear) to illustrate our dynamic optimization methodology, though other objective functions characterizations work equally well for our methodology.

4. Algorithms for dynamic optimization methodology

In this section, we describe our dynamic optimization methodology's three steps (Fig. 1) and associated algorithms.

4.1. Initial tunable parameter value settings and exploration order

The first step of our dynamic optimization methodology determines initial tunable parameter value settings and exploration order (ascending or descending). These initial tunable parameter value settings results in a high-quality operating state in *one-shot*,

Author's personal copy

hence the name *one-shot mode* (Fig. 1). The algorithm calculates the application metric objective function values for the first and last values in the set of tunable values for each tunable parameter while other tunable parameters are set to an arbitrary initial setting (either first or last value). We point out that the tunable values for a tunable parameter can be arranged in an ascending order (e.g., for processor voltage $V_p = \{2.7, 3.3, 4\}$ V). This objective function values calculation determines the effectiveness of setting a particular tunable parameter value in meeting the desired objective (e.g., lifetime). The tunable parameter setting that gives a higher objective function value is selected as the initial parameter value for that tunable parameter. The exploration order for that tunable parameter is set to descending if the last value in the set of tunable values (e.g., $V_p = 4$ in our previous example) gives a higher objective function value or ascending otherwise. This exploration order selection helps in reducing design space exploration for a greedy-based optimization algorithm (step 3), which stops exploring a tunable parameter as soon as a tunable parameter setting gives a lower objective function value than the initial setting. This initial parameter value setting and exploration order determination procedure is then repeated for all other tunable parameters and application metrics.

Algorithm 1 *(Initial tunable parameter value settings and exploration order algorithm).*

```
    Input: $f(s), N, n, m, P$
    Output: Initial tunable parameter value settings and exploration order
 1  for $k \leftarrow 1$ to $m$ do
 2      for $P_i \leftarrow P_1$ to $P_N$ do
 3          $f^k_{p_{i_1}} \leftarrow k^{th}$ metric objective function value when parameter setting is
            $\{P_i = p_{i_1}, P_j = P_{j_0}, i \neq j\}$;
 4          $f^k_{p_{i_n}} \leftarrow k^{th}$ metric objective function value when parameter setting is
            $\{P_i = p_{i_n}, P_j = P_{j_0}, i \neq j\}$;
 5          $\delta f^k_{P_i} \leftarrow f^k_{p_{i_n}} - f^k_{p_{i_1}}$;
 6          if $\delta f^k_{P_i} \geq 0$ then
 7              explore $P_i$ in descending order ;
 8              $P^k_d[i] \leftarrow$ descending ;
 9              $P^k_0[i] \leftarrow p^k_{i_n}$ ;
10          else
11              explore $P_i$ in ascending order ;
12              $P^k_d[i] \leftarrow$ ascending ;
13              $P^k_0[i] \leftarrow p^k_{i_1}$ ;
14          end
15      end
16  end
    return $P^k_d, P^k_0, k \in \{1,...,m\}$
```

Algorithm 1 describes our technique to determine initial tunable parameter value settings and exploration order (first step of our dynamic optimization methodology). The algorithm takes as input the objective function $f(s)$, the number of tunable parameters N, the number of values for each tunable parameter n, the number of application metrics m, and $\boldsymbol{P}$ where $\boldsymbol{P}$ represents a vector containing the tunable parameters, $\boldsymbol{P} = \{P_1, P_2, \ldots, P_N\}$. For each application metric k, the algorithm calculates vectors $\boldsymbol{P^k_0}$ and $\boldsymbol{P^k_d}$ (where d denotes the exploration direction (ascending or descending)), which store the initial value settings and exploration order, respectively, for the tunable parameters. The algorithm determines the kth application metric objective function values $f^k_{p_{i_1}}$ and $f^k_{p_{i_n}}$ where the parameter being explored P_i is assigned its first p_{i_1} and last p_{i_n} tunable values, respectively, and the rest of the tunable parameters P_j, $\forall j \neq i$ are assigned initial values (lines 3–4). $\delta f^k_{P_i}$ stores the difference between $f^k_{p_{i_n}}$ and $f^k_{p_{i_1}}$. If $\delta f^k_{P_i} \geq 0$, p_{i_n} results in a greater (or equal when $\delta f^k_{P_i} = 0$) objective function value as compared to p_{i_1} for parameter P_i (i.e., the objective function value decreases as the parameter value decreases). Therefore, to reduce the number of states explored while considering that the greedy algorithm (Section 4.3) stops exploring a tunable parameter if a tunable parameter's value yields a comparatively lower objective function value, P_i's exploration order must be descending (lines 6–8). The algorithm assigns p_{i_n} as the initial value of P_i for the kth application metric (line 9). If $\delta f^k_{P_i} < 0$, the algorithm assigns the exploration order as ascending for P_i and p_{i_1} as the initial value setting of P_i (lines 11–13). This $\delta f^k_{P_i}$ calculation procedure is repeated for all m application metrics and all N tunable parameters (lines 1–16).

4.2. Parameter arrangement

Depending on the application metric weight factors, some parameters are more critical to meeting application requirements than other parameters. For example, sensing frequency is a critical parameter for applications with a high responsiveness weight factor and therefore, sensing frequency should be explored first. In this subsection, we describe a technique for parameter arrangement such that parameters are explored in an order characterized by the parameters' impact on application metrics based on relative weight factors. This parameter arrangement technique (step 2) is part of the *improvement mode*, which is suitable for relatively less constrained applications that would benefit from a higher quality operating state than the one-shot mode's operating state (Fig. 1).

The parameter arrangement step determines an arrangement for the tunable parameters corresponding to each application metric, which dictates the order in which the parameters will be explored. This arrangement is based on the difference between the application metric's objective function values corresponding to the first and last values of the tunable parameters, which is calculated in step 1 (i.e., the tunable parameter that gives the highest difference in an application metric's objective function values is the first parameter in the arrangement vector for that application metric). For an arrangement that considers all application metrics, the tunable parameters' order is set in accordance with application metrics' weight factors such that the tunable parameters having a greater effect on application metrics with higher weight factors are situated before parameters having a lesser affect on application metrics with lower weight factors in the arrangement. We point out that the effect of the tunable parameters on an application metric is determined from the objective function value calculations as described in step 1. The arrangement that considers all application metrics selects the first few tunable parameters corresponding to each application metric, starting from the application metric with the highest weight factor such that no parameters are repeated in the final intelligent parameter arrangement. For example, if processor voltage is amongst the first few tunable parameters corresponding to two application metrics, then the processor voltage setting corresponding to the application metric with the greater weight factor is selected whereas the processor voltage setting corresponding to the application metric with the lower weight factor is ignored in the final intelligent parameter arrangement. Step 3 (online optimization algorithm) uses this intelligent parameter arrangement for further design space exploration. The mathematical details of the parameter arrangement step are as follows.

Our parameter arrangement technique is based on calculations performed in Algorithm 1. We define:

$$\boldsymbol{\nabla f_P} = \{\nabla f^1_P, \nabla f^2_P, \ldots, \nabla f^m_P\} \tag{7}$$

Author's personal copy

where ∇f_P is a vector containing $\nabla f_P^k, \forall k \in \{1, 2, \ldots, m\}$ arranged in descending order by their respective values and is given as:

$$\nabla \boldsymbol{f_P^k} = \{\delta f_{P_1}^k, \delta f_{P_2}^k, \ldots, \delta f_{P_N}^k\} : |\delta f_{P_i}^k| \geq |\delta f_{P_{i+1}}^k|, \quad \forall i \in \{1, 2, \ldots, N-1\} \tag{8}$$

The tunable parameter arrangement vector $\mathbf{P^k}$ corresponding to $\nabla \boldsymbol{f_P^k}$ (one-to-one correspondence) is given by:

$$\boldsymbol{P^k} = \{P_1^k, P_2^k, \ldots, P_N^k\}, \quad \forall k \in \{1, 2, \ldots, m\} \tag{9}$$

An intelligent parameter arrangement $\widehat{\boldsymbol{P}}$ must consider all application metrics' weight factors with higher importance given to the higher weight factors, i.e.,:

$$\widehat{\boldsymbol{P}} = \{P_1^1, \ldots, P_{l_1}^1, P_1^2, \ldots, P_{l_2}^2, P_1^3, \ldots, P_{l_3}^3, \ldots, P_1^m, \ldots, P_{l_m}^m\} \tag{10}$$

where l_k denotes the number of tunable parameters taken from P^k, $\forall k \in \{1, 2, \ldots, m\}$ such that $\sum_{k=1}^{m} l_k = N$. Our technique allows taking more tunable parameters from parameter arrangement vectors corresponding to higher weight factor application metrics: $l_k \geq l_{k+1}$, $\forall k \in \{1, 2, \ldots, m-1\}$. In (10), l_1 tunable parameters are taken from vector P^1, then l_2 from vector P^2, and so on to l_m from vector P^m such that $\{P_1^k, \ldots, P_{l_k}^k\} \cap \{P_1^{k-1}, \ldots, P_{l_{k-1}}^{k-1}\} = \emptyset, \forall \ k \in \{2, 3, \ldots, m\}$. In other words, we select those tunable parameters from parameter arrangement vectors corresponding to the lower weight factors that are not already selected from parameter arrangement vectors corresponding to the higher weight factors (i.e., $\widehat{\boldsymbol{P}}$ comprises of disjoint or non-overlapping tunable parameters corresponding to each application metric).

In the situation where weight factor ω_1 is much greater than all other weight factors, an intelligent parameter arrangement $\widetilde{\boldsymbol{P}}$ would correspond to the parameter arrangement for the application metric with weight factor ω_1, i.e.,:

$$\widetilde{\boldsymbol{P}} = \boldsymbol{P^1} = \{P_1^1, P_2^1, \ldots, P_N^1\} \Leftrightarrow \omega_1 \gg \omega_q, \quad \forall q \in \{2, 3, \ldots, m\} \tag{11}$$

The initial parameter value vector $\widehat{\boldsymbol{P}}_{\boldsymbol{0}}$ and the exploration order (ascending or descending) vector $\widehat{\boldsymbol{P}}_{\boldsymbol{d}}$ corresponding to $\widehat{\boldsymbol{P}}$ (10) can be determined from $\widehat{\boldsymbol{P}}$ (10), $\boldsymbol{P_d^k}$, and $\boldsymbol{P_0^k}$, $\forall k \in \{1, \ldots, m\}$ (Algorithm 1) by examining the tunable parameter from $\widehat{\boldsymbol{P}}$ and determining the tunable parameter's initial value setting from $\boldsymbol{P_0^k}$ and exploration order from $\boldsymbol{P_d^k}$.

4.3. Online optimization algorithm

Step three of our dynamic optimization methodology, which also belongs to the *improvement mode*, iteratively improves the one-shot's operating state. This step leverages information from steps one and two, and uses a greedy optimization algorithm for tunable parameters exploration in an effort to determine a better operating state than the one obtained from step one (Section 4.1). The greedy algorithm explores the tunable parameters in the order determined in step 2. The greedy algorithm stops exploring a tunable parameter as soon as a tunable parameter setting yields a lower objective function value as compared to the previous tunable parameter setting for that tunable parameter, and hence named as *greedy*. This greedy approach helps in reducing design space exploration to determine an operating state. Even though we propose a greedy algorithm for design space exploration, any other algorithm can be used in step three.

Algorithm 2 *(Online greedy optimization algorithm for tunable parameters exploration).*

```
Input: $f(s)$, $N$, $n$, $\widehat{P}$, $\widehat{P}_0$, $\widehat{P}_d$
Output: Sensor node state that maximizes $f(s)$ and the corresponding $f(s)$ value
1  $\kappa \leftarrow$ initial tunable parameter value settings from $\widehat{P}_0$ ;
2  $f_{best} \leftarrow$ solution from initial parameter settings $\kappa$ ;
3  for $\widehat{P}_i \leftarrow \widehat{P}_1$ to $\widehat{P}_N$ do
4      explore $\widehat{P}_i$ in ascending or descending order as suggested by $\widehat{P}_d$ ;
5      foreach $\widehat{P}_i = \{\widehat{p}_{i_1}, \widehat{p}_{i_2}, \ldots, \widehat{p}_{in}\}$ do
6          $f_{temp} \leftarrow$ current state $\zeta$ solution ;
7          if $f_{temp} > f_{best}$ then
8              $f_{best} \leftarrow f_{temp}$ ;
9              $\xi \leftarrow \zeta$ ;
10         else
11             break ;
12         end
13     end
14 end
   return $\xi, f_{best}$
```

Algorithm 2 depicts our online greedy optimization algorithm, which leverages the initial parameter value settings (Section 4.1), parameter value exploration order (Section 4.1), and parameter arrangement (Section 4.2). The algorithm takes as input the objective function $f(s)$, the number of tunable parameters N, the number of values for each tunable parameter n, the intelligent tunable parameter arrangement vector $\widehat{\boldsymbol{P}}$, the tunable parameters' initial value vector $\widehat{\boldsymbol{P}}_{\boldsymbol{0}}$, and the tunable parameter's exploration order (ascending or descending) vector $\widehat{\boldsymbol{P}}_{\boldsymbol{d}}$. The algorithm initializes state κ from $\widehat{\boldsymbol{P}}_{\boldsymbol{0}}$ (line 1) and f_{best} with κ's objective function value (line 2). The algorithm explores each parameter in $\widehat{P}_i$ where $\widehat{P}_i \in \widehat{\boldsymbol{P}}$ (10) in ascending or descending order as given by $\widehat{\boldsymbol{P}}_{\boldsymbol{d}}$ (lines 3–4). For each tunable parameter $\widehat{P}_i$ (line 5), the algorithm assigns f_{temp} the objective function value from the current state ζ (line 6). The current state $\zeta \in S$ denotes tunable parameter value settings and can be written as:

$$\zeta = \{P_i = p_{i_x}\} \cup \{P_j, \ \forall j \neq i\}, \quad i, j \in \{1, 2, \ldots, N\} \tag{12}$$

where $p_{i_x} : x \in \{1, 2, \ldots, n\}$ denotes the parameter value corresponding to the tunable parameter P_i being explored and set P_j, $\forall j \neq i$ denotes the parameter value settings other than the current tunable parameter P_i being explored and is given by:

$$P_j = \begin{cases} P_{j0}, & \text{if } P_j \text{ not explored before}, \quad \forall \ \ j \neq i \\ P_{jb}, & \text{if } P_j \text{ explored before}, \quad \forall \ \ j \neq i \end{cases} \tag{13}$$

where P_{j0} denotes the initial value of the parameter as given by $\widehat{\boldsymbol{P}}_{\boldsymbol{0}}$ and P_{jb} denotes the best found value of P_j after exploring P_j (lines 5–13 of Algorithm 2).

If $f_{temp} > f_{best}$ (the objection function value increases), f_{temp} is assigned to f_{best} and the state ζ is assigned to state ξ (lines 7–9). If $f_{temp} \leq f_{best}$, the algorithm stops exploring the current parameter $\widehat{P}_i$ and starts exploring the next tunable parameter (lines 10–12). The algorithm returns the best found objective function value f_{best} and the state ξ corresponding to f_{best}.

4.4. Computational complexity

The computational complexity for our dynamic optimization methodology is $\mathcal{O}(Nm \log N + Nn)$, which is comprised of the intelligent initial parameter value settings and exploration ordering (Algorithm 1) $\mathcal{O}(Nm)$, parameter arrangement $\mathcal{O}(Nm \log N)$ (sorting ∇f_P^k (8) contributes the $N \log N$ factor) (Section 4.2), and the online optimization algorithm for parameter exploration (Algorithm 2) $\mathcal{O}(Nn)$. Assuming that the number of tunable parameters N is larger than the number of parameter's tunable values n, the

Author's personal copy

computational complexity of our methodology can be given as $\mathcal{O}(Nm \log N)$. This complexity reveals that our proposed methodology is lightweight and is thus feasible for implementation on sensor nodes with tight resource constraints.

5. Application metrics estimation model

This section presents our application metrics estimation model, which is leveraged by our dynamic optimization methodology. This estimation model estimates high-level application metrics (lifetime, throughput, reliability) from low-level tunable parameters and sensor node hardware internals. The use of hardware internals is appropriate for application metrics modeling as similar approaches have been used in literature especially for lifetime estimation [24,12,13]. Based on tunable parameter value settings corresponding to an operating state and hardware specific values, the application metrics estimation model determines corresponding values for high-level application metrics. These high-level application metric values are then used in their respective objective functions to determine the objective function values corresponding to an operating state (e.g., lifetime estimation model determines s_l (lifetime offered by state s), which is then used in (6) to determine the lifetime objective function value). This section presents a complete description of our application metrics estimation model, including a review of our previous application metrics estimation model [20] and additional details.

5.1. Lifetime estimation

A sensor node's *lifetime* is defined as the time duration between sensor node deployment and sensor node failure due to a wide variety of reasons (e.g., battery depletion, hardware/software fault, environmental damage, external destruction, etc.). Lifetime estimation models typically consider battery depletion as the cause of sensor node failure [21]. Since sensor nodes can be deployed in remote and hostile environments, manual battery replacement after deployment is often impractical. A sensor node reaches the failed or *dead* state once the entire battery energy is depleted. The critical factors that determine a sensor node's lifetime are battery energy and energy consumption during operation.

The sensor node lifetime in days $\mathcal{L}_s$ can be estimated as:

$$\mathcal{L}_s = \frac{E_b}{E_c \times 24} \tag{14}$$

where E_b denotes the sensor node's battery energy in Joules and E_c denotes the sensor node's energy consumption per hour. The battery energy in mWh E'_b can be given by:

$$E'_b = V_b \cdot C_b \,(\text{mWh}) \tag{15}$$

where V_b denotes battery voltage in V and C_b denotes battery capacity, typically specified in mAh. Since 1 J = 1 W s, E_b can be calculated as:

$$E_b = E'_b \times \frac{3600}{1000} \,(\text{J}) \tag{16}$$

The sensors in the sensor node gather information about the physical environment and generate continuous sequences of analog signals/values. Sample-and-hold-circuits and analog-to-digital (A/D) converters digitize these analog signals. This digital information is processed by a processor, and the results are communicated to other sensor nodes or a base station node (sink node) via a transmitter. The *sensing energy* is the energy consumed by the sensor node due to sensing events. The *processing energy* is the energy consumed by the processor to process the sensed data (e.g., calculating the average of the sensor values over a time interval or the difference between the most recent sensor values and the previously sensed values). The *communication energy* is the energy consumed due to communication with other sensor nodes or the sink node. For example, sensor nodes send packets containing the sensed/processed data information to other sensor nodes and the sink node, which consumes communication energy.

We model E_c as the sum of the processing energy, communication energy, and sensing energy, i.e.:

$$E_c = E_{sen} + E_{proc} + E_{com}\,(\text{J}) \tag{17}$$

where E_{sen}, E_{proc}, and E_{com} denote the sensing energy per hour, processing energy per hour, and communication energy per hour, respectively.

The sensing (sampling) frequency and the number of sensors attached to the sensor board (e.g., the MTS400 sensor board [5] has Sensirion SHT1x temperature and humidity sensors [23]) are the main contributors to the total sensing energy. Our model considers energy conservation by allowing sensors to switch to a low power, idle mode while not sensing. E_{sen} is given by:

$$E_{sen} = E^m_{sen} + E^i_{sen} \tag{18}$$

where E^m_{sen} denotes the sensing measurement energy per hour and E^i_{sen} denotes the sensing idle energy per hour. E^m_{sen} can be calculated as:

$$E^m_{sen} = N_s \cdot V_s \cdot I^m_s \cdot t^m_s \times 3600 \tag{19}$$

where N_s denotes the number of sensing measurements per second, V_s denotes the sensing board voltage, I^m_s denotes the sensing measurement current, and t^m_s denotes the sensing measurement time. N_s can be calculated as:

$$N_s = N_r \cdot F_s \tag{20}$$

where N_r denotes the number of sensors on the sensing board and F_s denotes the sensing frequency. E^i_{sen} is given by:

$$E^i_{sen} = V_s \cdot I_s \cdot t^i_s \times 3600 \tag{21}$$

where I_s denotes the sensing sleep current and t^i_s denotes the sensing idle time. t^i_s is given by:

$$t^i_s = 1 - t^m_s \tag{22}$$

We assume that the sensor node's processor operates in two modes: active mode and idle mode [3]. The processor operates in active mode while processing the sensed data and switches to the idle mode for energy conservation when not processing. The processing energy is the sum of the processor's energy consumption while operating in the active and the idle modes. We point out that although we only consider active and idle modes, a processor operating in additional sleep modes (e.g., power-down, power-save, standby, etc.) can also be incorporated in our model. E_{proc} is given by:

$$E_{proc} = E^a_{proc} + E^i_{proc} \tag{23}$$

where E^a_{proc} and E^i_{proc} denote the processor's energy consumption per hour in the active and idle modes, respectively. E^a_{proc} is given by:

$$E^a_{proc} = V_p \cdot I^a_p \cdot t^a \tag{24}$$

where V_p denotes the processor voltage, I^a_p denotes the processor active mode current, and t^a denotes the time spent by the processor in the active mode. t^a can be estimated as:

$$t^a = \frac{N_I}{F_p} \tag{25}$$

Author's personal copy

where N_I denotes the average number of processor instructions to process one sensing measurement and F_p denotes the processor frequency. N_I can be estimated as:

$$N_I = N^b \cdot R^b_{sen} \tag{26}$$

where N^b denotes the average number of processor instructions to process one bit and R^b_{sen} denotes the sensing resolution bits (number of bits required for storing one sensing measurement).

E^i_{proc} is given by:

$$E^i_{proc} = V_p \cdot I^i_p \cdot t^i \tag{27}$$

where I^i_p denotes the processor idle mode current and t^i denotes the time spent by the processor in the idle mode. Since the processor switches to the idle mode when not processing sensing measurements, t_i can be given as:

$$t^i = 1 - t^a \tag{28}$$

The transceiver (radio) is the main contributor to the total communication energy consumption. The transceiver transmits/receives data packets and switches to the idle mode for energy conservation when there are no more packets to transmit/receive. The number of packets transmitted (received) and the packets' transmission (receive) interval dictates the communication energy. The communication energy is the sum of the transmission, receive, and idle energies for the sensor node's transceiver, i.e.,:

$$E_{com} = E^{tx}_{trans} + E^{rx}_{trans} + E^i_{trans} \tag{29}$$

where E^{tx}_{trans}, E^{rx}_{trans}, and E^i_{trans} denote the transceiver's transmission energy per hour, receive energy per hour, and idle energy per hour, respectively. E^{tx}_{trans} is given by:

$$E^{tx}_{trans} = N^{tx}_{pkt} \cdot E^{pkt}_{tx} \tag{30}$$

where N^{tx}_{pkt} denotes the number of packets transmitted per hour and E^{pkt}_{tx} denotes the transmission energy per packet. N^{tx}_{pkt} can be calculated as:

$$N^{tx}_{pkt} = \frac{3600}{P_{ti}} \tag{31}$$

where P_{ti} denotes the packet transmission interval in seconds (1 h = 3600 s). E^{pkt}_{tx} is given as:

$$E^{pkt}_{tx} = V_t \cdot I_t \cdot t^{pkt}_{tx} \tag{32}$$

where V_t denotes the transceiver voltage, I_t denotes the transceiver current, and t^{pkt}_{tx} denotes the time to transmit one packet. t^{pkt}_{tx} is given by:

$$t^{pkt}_{tx} = P_s \times \frac{8}{R_{tx}} \tag{33}$$

where P_s denotes the packet size in bytes and R_{tx} denotes the transceiver data rate (in bits/s).

The transceiver's receive energy per hour E^{rx}_{trans} can be calculated using a similar procedure as E^{tx}_{trans}. E^{rx}_{trans} is given by:

$$E^{rx}_{trans} = N^{rx}_{pkt} \cdot E^{pkt}_{rx} \tag{34}$$

where N^{rx}_{pkt} denotes the number of packets received per hour and E^{pkt}_{rx} denotes the receive energy per packet. N^{rx}_{pkt} can be calculated as:

$$N^{rx}_{pkt} = \frac{3600}{P_{ri}} \tag{35}$$

where P_{ri} denotes the packet receive interval in s. P_{ri} can be calculated as:

$$P_{ri} = \frac{P_{ti}}{n_s} \tag{36}$$

where n_s denotes the number of neighboring sensor nodes. E^{pkt}_{rx} is given as:

$$E^{pkt}_{rx} = V_t \cdot I^{rx}_t \cdot t^{pkt}_{rx} \tag{37}$$

where I^{rx}_t denotes the transceiver receive current and t^{pkt}_{rx} denotes the time to receive one packet. Since the packet size is the same, the time to receive a packet is equal to the time to transmit the packet, i.e., $t^{pkt}_{rx} = t^{pkt}_{tx}$.

E^i_{trans} can be calculated as:

$$E^i_{trans} = V_t \cdot I^s_t \cdot t^i_{tx} \tag{38}$$

where I^s_t denotes the transceiver sleep current and t^i_{tx} denotes the transceiver idle time per hour. t^i_{tx} can be calculated as:

$$t^i_{tx} = 3600 - (N^{tx}_{pkt} \cdot t^{pkt}_{tx}) - (N^{rx}_{pkt} \cdot t^{pkt}_{rx}) \tag{39}$$

5.2. Throughput estimation

Throughput is defined as the amount of work processed by a system in a given unit of time. Defining throughput semantics for sensor nodes is challenging because three main components contribute to the throughput, sensing, processing, and communication (transmission), and these throughput components can have different significance for different applications. Since these throughput components are related, one possible interpretation is to take the throughput of the lowest throughput component as the *effective throughput*. However, the effective throughput may not be a suitable metric for a designer who is interested in throughputs associated with all three components.

In our model, we define the *aggregate throughput* as the combination of the sensor node's sensing, processing, and transmission rates to observe/monitor a phenomenon (measured in bits/second). The aggregate throughput can be considered as the weighted sum of the constituent throughputs. Our aggregate throughput model can be used for the effective throughput estimation by assigning a weight factor of one to the slowest of the three components and assigning a weight factor of zero to the others. Since aggregate throughput modeling allows flexibility and can be adapted to varying needs of a WSN designer, we focus on modeling of the aggregate throughput. We model aggregate throughput as:

$$R = \omega_s R_{sen} + \omega_p R_{proc} + \omega_c R_{com} : \omega_s + \omega_p + \omega_c = 1 \tag{40}$$

where R_{sen}, R_{proc}, and R_{com} denote the sensing, processing, and communication throughputs, respectively, and ω_s, ω_p, and ω_c denote the associated weight factors.

The sensing throughput, which is the throughput due to sensing activity, depends upon the sensing frequency and sensing resolution bits per sensing measurement. R_{sen} is given by:

$$R_{sen} = F_s \cdot R^b_{sen} \tag{41}$$

where F_s denotes the sensing frequency.

The processing throughput, which is the processor's throughput while processing sensed measurements, depends upon the processor frequency and the average number of instructions required to process the sensing measurement. R_{proc} is given by:

$$R_{proc} = \frac{F_p}{N^b} \tag{42}$$

The communication throughput, which measures the number of packets transferred successfully over the wireless channel, depends upon the packet size and the time to transfer one packet. R_{com} is given by:

$$R_{com} = P^{eff}_s \times \frac{8}{t^{pkt}_{tx}} \tag{43}$$

Author's personal copy

where P_s^{eff} denotes the effective packet size excluding the packet header overhead (i.e., $P_s^{eff} = P_s - P_h$ where P_h denotes the packet header size).

5.3. Reliability estimation

The reliability metric measures the number of packets transferred reliably (i.e., error-free packet transmission) over the wireless channel. Accurate reliability estimation is challenging due to dynamic changes in the network topology, number of neighboring sensor nodes, wireless channel fading, sensor network traffic, packet size, etc. The two main factors that affect reliability are transceiver transmission power P_{tx} and receiver sensitivity. For example, the AT86RF230 transceiver [2] has a receiver sensitivity of -101 dBm with a corresponding packet error rate (PER) ≤1% for an additive white gaussian noise (AWGN) channel with a physical service data unit (PSDU) equal to 20 bytes. Reliability can be estimated using Friis free space transmission equation [7] for different P_{tx} values, distance between transmitting and receiving sensor nodes, and assumptions on fading model parameters (e.g., shadowing fading model). Different reliability values can be assigned corresponding to different P_{tx} values such that the higher P_{tx} values give higher reliability, however, more accurate reliability estimation requires using profiling statistics for the number of packets transmitted and the number of packets received. These profiling statistics increase the estimation accuracy of the PER and, therefore, reliability.

5.4. Models validation

Our models provide good accuracy in estimating application metrics since our models accommodate many sensor node hardware internals such as the battery voltage, battery capacity, sensing board voltage, sensing sleep current, sensing idle time, sensing resolution bits, etc. Our models are also highly flexible since our models permit calculations for particular network settings such as the number of neighboring sensor nodes and different types of sensors with different hardware characteristics (e.g., sensing resolution bits, sensing measurement time, sensing measurement current, etc.).

Since our models provide a first step toward modeling application metrics, our models' accuracy cannot be completely verified against other models because there are no similar/related application metrics estimation models. The existing models for lifetime estimation take different parameters and have different assumptions, thus an exact comparison is not feasible, however, we observe that our lifetime model yields results in a similar range as other models [24,12,13]. We also compare the lifetime estimation from our model with an experimental study on WSN lifetimes [21]. This comparison verifies conformity of our lifetime model with real measurements. For example, with a sensor node battery capacity of 2500 mAh, experiments indicate a sensor node lifetime ranging from 72 to 95 h for a 100% duty cycle for different battery brands (e.g., Ansmann, Panasonic Industrial, Varta High Energy, Panasonic Extreme Power) [21]. Using our model with a duty cycle of 36% on average for the sensing, processing, and communication, we calculated that a lifetime of $95/0.36 = 264\,\text{h} \approx 11$ days can be attained. Similarly for a duty cycle of 0.25% on average for the sensing, communication, and processing, the lifetime can be calculated as $95/0.0025 = 38{,}000\,\text{h} \approx 1583$ days (example lifetime calculations using our model is given in Section 6.2.1).

The relative comparison of our models with existing models and real measurements provide insights into the accuracy of our models, however, more accurate models can be constructed following our modeling approach by considering additional parameters and more detailed hardware models for sensor nodes.

6. Experimental results

In this section, we describe the experimental setup and results for three application domains: security/defense, health care, and ambient conditions monitoring. The results include the percentage improvements attained by our initial tunable parameter settings (one-shot operating state) over other alternative initial value settings, and a comparison of our greedy algorithm (which leverages intelligent initial parameter settings, exploration order, and parameter arrangement) for design space exploration with other variants of a greedy algorithm and SA. This section also presents an execution time and data memory analysis to verify the complexity of our dynamic optimization methodology.

6.1. Experimental setup

Our experimental setup is based on the Crossbow IRIS mote platform [4] with a battery capacity of 2000 mAh using two AA alkaline batteries. The IRIS mote platform integrates an Atmel ATmega1281 microcontroller [3], an MTS400 sensor board [5] with Sensirion SHT1x temperature and humidity sensors [23], and an Atmel AT-86RF230 low-power 2.4 GHz transceiver [2]. Table 1 shows the sensor node hardware specific values, corresponding to the IRIS mote platform, which are used by the application metrics estimation model [4,3,23,2].

We analyze six tunable parameters: processor voltage V_p, processor frequency F_p, sensing frequency F_s, packet size P_s, packet transmission interval P_{ti}, and transceiver transmission power P_{tx}. In order to explore the fidelity of our methodology across small and large design spaces, we consider two design space cardinalities (number of states in the design space): $|S| = 729$ and $|S| = 31,104$. The tunable parameters for $|S| = 729$ are: $V_p = \{2.7, 3.3, 4\}$ (v), $F_p = \{4, 6, 8\}$ (MHz) [3], $F_s = \{1, 2, 3\}$ (samples per second) [23], $P_s = \{41, 56, 64\}$ (bytes), $P_{ti} = \{60, 300, 600\}$ (s), and $P_{tx} = \{-17, -3, 1\}$ (dBm) [2]. The tunable parameters for $|S| = 31,104$ are: $V_p = \{1.8, 2.7, 3.3, 4, 4.5, 5\}$ (V), $F_p = \{2, 4, 6, 8, 12, 16\}$ (MHz) [3], $F_s = \{0.2, 0.5, 1, 2, 3, 4\}$ (samples per second) [23], $P_s = \{32, 41, 56, 64, 100, 127\}$ (bytes), $P_{ti} = \{10, 30, 60, 300, 600, 1200\}$ (s), and $P_{tx} = \{-17, -3, 1, 3\}$ (dBm) [2]. All state space tuples are feasible for $|S| = 729$, whereas $|S| = 31,104$ contains 7779 infeasible state space tuples because all V_p and F_p pairs are not feasible.

In order to evaluate the robustness of our methodology across different applications with varying application metric weight factors, we model three sample application domains: a security/defense system, a health care application, and an ambient conditions monitoring application. We assign application specific values for the desirable minimum L, desirable maximum U, accept-

Table 1
Crossbow IRIS mote platform hardware specifications.

Notation	Description	Value
V_b	Battery voltage	3.6 V
C_b	Battery capacity	2000 mAh
N_b	Processing instructions per bit	5
R_{sen}^b	Sensing resolution bits	24
V_t	Transceiver voltage	3 V
R_{tx}	Transceiver data rate	250 kbps
I_t^{rx}	Transceiver receive current	15.5 mA
I_t^s	Transceiver sleep current	20 nA
V_s	Sensing board voltage	3 V
I_s^m	Sensing measurement current	550 μA
t_s^m	Sensing measurement time	55 ms
I_s	Sensing sleep current	0.3 μA

Author's personal copy

Table 2
Desirable minimum L, desirable maximum U, acceptable minimum α, and acceptable maximum β objective function parameter values for the studied applications. One lifetime unit = 5 days, one throughput unit = 20 kbps, one reliability unit = 0.05.

Notation	Security/defense	Health care	Ambient monitoring
L_l	8 units	12 units	6 units
U_l	30 units	32 units	40 units
α_l	1 units	2 units	3 units
β_l	36 units	40 units	60 units
L_t	20 units	19 units	15 unit
U_t	34 units	36 units	29 units
α_t	0.5 units	0.4 units	0.05 units
β_t	45 units	47 units	35 units
L_r	14 units	12 units	11 units
U_r	19.8 units	17 units	16 units
α_r	10 units	8 units	6 units
β_r	20 units	20 units	20 units

able minimum α, and acceptable maximum β objective function parameter values for application metrics (Section 3.3) as shown in Table 2. We specify the objective function parameters as a multiple of base units for lifetime, throughput, and reliability. We assume one lifetime unit is equal to 5 days, one throughput unit is equal to 20 kbps, and one reliability unit is equal to 0.05 (percentage of error-free packet transmissions).

Although we analyzed our methodology for the IRIS motes platform, three application domains, and two design spaces, our algorithms and application metrics estimation model are equally applicable to any platform, application domain, and design space. Our application metrics estimation model accommodates several sensor node hardware internals, which are hardware platform-specific and can be obtained from the platform's datasheets. Since the appropriate values can be substituted for any given platform, our model can be used with any hardware platform. Since the constant assignments for the minimum and maximum desirable values and weight factors are application-dependent and designer-specified, appropriate assignments can be made for any application given the application's specific requirements. Finally, since the number of tunable parameters and the parameters' possible/allowed tunable values dictates the size of the design space, we evaluate both large and small design spaces but any sized design space could be evaluated by varying the number of tunable parameters and associated values.

6.2. Results

In this subsection, we present example application metrics calculations using our application metrics estimation model as well as present results for percentage improvements attained by our dynamic optimization methodology over other optimization methodologies.

6.2.1. Application metrics estimation

Since the objective function values corresponding to different states depends upon the estimation of high-level application metrics, we present example calculations to exemplify this estimation process using our application metrics estimation model (Section 5) and the IRIS mote platform hardware specifications (Table 1). We consider the example state $s_y = (V_{p_y}, F_{p_y}, F_{s_y}, P_{s_y}, P_{ti_y}, P_{tx_y}) = (2.7, 4, 1, 41, 60, -17)$.

First, we calculate the lifetime corresponding to s_y. Using (15), the battery energy is $E'_b = 3.6 \times 2000 = 7200$ mWh, which is $E_b = 7200 \times 3600/1000 = 25{,}920$ J from (16). The lifetime metric calculation requires calculation of processing, communication, and sensing energy.

For the processing energy per hour, (26) and (25) give $N_I = 5 \times 24 = 120$ and $t^a = 120/(4 \times 10^6) = 30\,\mu$s, respectively. The processor's active mode energy consumption per hour from (24) is $E^a_{proc} = 2.7 \times 2.5 \times 10^{-3} \times 30 \times 10^{-6} = 0.2025\,\mu$J where $I^a_p = 2.5$ mA corresponding to $(V_{p_y}, F_{p_y}) = (2.7, 4)$ [3]. Using (28) gives $t^i = 1 - 30 \times 10^{-6}$ s = 999.97 ms. The processor's idle mode energy consumption per hour from (27) is $E^i_{proc} = 2.7 \times 0.65 \times 10^{-3} \times 999.97 \times 10^{-3} = 1.755$ mJ where $I^i_p = 0.65$ mA corresponding to $(V_{p_y}, F_{p_y}) = (2.7, 4)$ [3]. The processor energy consumption per hour from (23) is $E_{proc} = 0.2025 \times 10^{-6} + 1.755 \times 10^{-3} = 1.7552$ mJ.

For the communication energy per hour, (31) and (33) give $N^{tx}_{pkt} = 3600/60 = 60$ and $t^{pkt}_{tx} = 41 \times 8/(250 \times 10^3) = 1.312$ ms, respectively. (32) gives $E^{pkt}_{tx} = 3 \times 9.5 \times 10^{-3} \times 1.312 \times 10^{-3} = 37.392\,\mu$J. The transceiver's transmission energy per hour from (30) is $E^{tx}_{trans} = 60 \times 37.392 \times 10^{-6} = 2.244$ mJ. (36) gives $P_{ri} = 60/2 = 30$ where we assume $n_s = 2$, however, our model is valid for any number of neighboring sensor nodes. (35) and (37) give $N^{rx}_{pkt} = 3600/30 = 120$ and $E^{pkt}_{rx} = 3 \times 15.5 \times 10^{-3} \times 1.312 \times 10^{-3} = 61.01\,\mu$J, respectively. The transceiver's receive energy per hour from (34) is $E^{rx}_{trans} = 120 \times 61.01 \times 10^{-6} = 7.3212$ mJ. (39) gives $t^i_{tx} = 3600 - (60 \times 1.312 \times 10^{-3}) - (120 \times 1.312 \times 10^{-3}) = 3599.764$ s. The transceiver's idle energy per hour from (38) is $E^i_{trans} = 3 \times 20 \times 10^{-9} \times 3599.764 = 0.216$ mJ. (29) gives communication energy per hour $E_{com} = 2.244 + 7.3212 + 0.216 = 9.7812$ mJ.

We calculate sensing energy per hour using (18). (20) gives $N_s = 2 \times 1 = 2$ (since MTS400 sensor board [5] has Sensirion SHT1x temperature and humidity sensors [23]). (19) gives $E^m_{sen} = 2 \times 3 \times 550 \times 10^{-6} \times 55 \times 10^{-3} \times 3600 = 0.6534$ J. Using (22) and (21) gives $t^i_s = 1 - 55 \times 10^{-3} = 0.945$ s and $E^i_{sen} = 3 \times 0.3 \times 10^{-6} \times 0.945 \times 3600 = 3.062$ mJ, respectively. (18) gives $E_{sen} = 0.6534 + 3.062 \times 10^{-3} = 0.6565$ J.

After calculating processing, communication, and sensing energy, we calculate the energy consumption per hour from (17) as $E_c = 1.7552 \times 10^{-3} + 9.7812 \times 10^{-3} + 0.6565 = 0.668$ J. (14) gives $\mathcal{L}_s = 25{,}920/(0.668 \times 24) = 1616.77$ days.

For the throughput application metric, (41), (42), and (43) give $R_{sen} = 1 \times 24 = 24$ bps, $R_{proc} = 4 \times 10^6/5 = 800$ kbps, and $R_{com} = 21 \times 8/(1.312 \times 10^{-3}) = 128.049$ kbps, respectively ($P^{eff}_s = 41 - 21 = 20$ where we assume $P_h = 21$ bytes). (40) gives $R = (0.4)(24) + (0.4)(800 \times 10^3) + (0.2)(128.049 \times 10^3) = 345.62$ kbps where we assume ω_s, ω_p, and ω_c equal to 0.4, 0.4, and 0.2, respectively.

We estimate the reliability corresponding to $P_{tx} = -17$ dBm to be 0.7 (Section 5.3), however, an accurate reliability value can only be obtained using profiling statistics for the number of packets transmitted and number of packets lost.

Similarly, the lifetime, throughput, and reliability for state $s_y = (V_{p_y}, F_{p_y}, F_{s_y}, P_{s_y}, P_{ti_y}, P_{tx_y}) = (5, 16, 4, 127, 10, 3)$ can be calculated as 10.6 days, 1321.77 kbps, and 0.9999, respectively. These calculation reveal that the tunable parameter value settings for a sensor node can have a profound impact on the application metrics. For example, the lifetime of a sensor node in our two examples varied from 10.6 days to 1616.8 days for different tunable parameter value settings. Hence, our proposed tunable parameter value settings technique and application metrics estimation model can help WSN designers to find appropriate tunable parameter value settings to conserve the sensor node's energy and to enhance the sensor node's lifetime after satisfying other application requirements such as throughput and reliability.

6.2.2. Percentage improvements of one-shot over other initial parameter settings

In order to evaluate our one-shot dynamic optimization solution quality, we compare the solution from the one-shot initial parameter settings $\widehat{\boldsymbol{P}_0}$ with the solutions obtained from the following four potential initial parameter value settings (although

Author's personal copy

Table 3
Percentage improvements attained by one-shot ($\widehat{P_0}$) over other initial parameter settings for $|S| = 729$ and $|S| = 31{,}104$.

	$\lvert S\rvert = 729$				$\lvert S\rvert = 31{,}104$			
App. domain	$\mathcal{I}_1$	$\mathcal{I}_2$	$\mathcal{I}_3$	$\mathcal{I}_4$	$\mathcal{I}_1$	$\mathcal{I}_2$	$\mathcal{I}_3$	$\mathcal{I}_4$
Security/defense	155%	10%	57%	29%	148%	0.3%	10%	92%
Health care	78%	7%	31%	11%	73%	0.3%	10%	45%
Ambient monitoring	52%	6%	20%	7%	15%	−7%	−12%	18%

any feasible n-tuple $s \in S$ can be taken as the initial parameter settings):

- $\mathcal{I}_1$ assigns the first parameter value for each tunable parameter, i.e., $\mathcal{I}_1 = p_{i_1}, \quad \forall \quad i \in \{1, 2, \ldots, N\}$.
- $\mathcal{I}_2$ assigns the last parameter value for each tunable parameter, i.e., $\mathcal{I}_2 = p_{i_n}, \quad \forall \quad i \in \{1, 2, \ldots, N\}$.
- $\mathcal{I}_3$ assigns the middle parameter value for each tunable parameter, i.e., $\mathcal{I}_3 = \lfloor p_{i_n}/2 \rfloor, \quad \forall \quad i \in \{1, 2, \ldots, N\}$.
- $\mathcal{I}_4$ assigns a random value for each tunable parameter, i.e., $\mathcal{I}_4 = p_{i_q} : q = rand()\,\%\,n, \quad \forall \quad i \in \{1, 2, \ldots, N\}$ where rand() denotes a function to generate a random/pseduo-random integer and % denotes the modulus operator.

Table 3 depicts the percentage improvements attained by the one-shot parameter settings $\widehat{P_0}$ over other parameter settings for different application domains and weight factors. We assume weight factors for the security/defense and health care applications as: $\omega_l = 0.25$, $\omega_t = 0.35$, and $\omega_r = 0.4$; and for the ambient conditions monitoring application as: $\omega_l = 0.4$, $\omega_t = 0.5$, and $\omega_r = 0.1$. We point out that different weight factors could result in different percentage improvements, however, we observed similar trends for other weight factors. Table 3 shows that one-shot initial parameter settings can result in as high as 155% improvement as compared to other initial value settings. We observe that some arbitrary settings may give a comparable or even a better solution for a particular application domain, application metric weight factors, and design space cardinality, but that arbitrary setting would not scale to other application domains, application metric weight factors, and design space cardinalities. For example, $\mathcal{I}_3$ obtains a 12% better quality solution than $\widehat{P_0}$ for the ambient conditions monitoring application for $|S| = 31, 104$, but yields a 10% lower quality solution for the security/defense and health care applications for $|S| = 31, 104$, and a 57%, 31%, and 20% lower quality solution than $\widehat{P_0}$ for the security/defense, health care, and ambient conditions monitoring applications, respectively, for $|S| = 729$.

The percentage improvement attained by $\widehat{P_0}$ over all application domains and design spaces is 33% on average. Our one-shot methodology is the first approach (to the best of our knowledge) to intelligent initial tunable parameter value settings for sensor nodes to provide a good quality operating state, as arbitrary initial parameter value settings typically result in a poor operating state. Results reveal that on average $\widehat{P_0}$ gives a solution within 8% of the optimal solution obtained from an exhaustive search [20].

6.2.3. *Comparison with greedy variants- and SA-based dynamic optimization methodologies*

For comparison purposes, we implemented an SA-based algorithm, our greedy online optimization algorithm (GD) (which leverages intelligent initial parameter value selection, exploration ordering, and parameter arrangement), and several other greedy online algorithm variations (Table 4) in C++. We compare our results with SA to provide relative comparisons of our dynamic optimization methodology with another methodology that leverages an SA-based online optimization algorithm and arbitrary initial value settings. We point out that step 3 of our dynamic optimization methodology can use any lightweight algorithm (e.g., greedy, SA-based) in the improvement mode (Fig. 1). Although, we present SA for comparison with the greedy algorithm, both of these algorithms are equally applicable to our dynamic optimization methodology. We compare GD results with different greedy algorithm variations (Table 4) to provide an insight into how initial parameter value settings, exploration ordering, and parameter arrangement affect the final operating state quality. We normalize the objective function value (corresponding to the operating state) attained by the algorithms with respect to the optimal solution (objective function value corresponding to the optimal operating state) obtained from an exhaustive search.

Table 4
Greedy algorithms with different parameter arrangements and exploration orders.

Notation	Description
GD	Greedy algorithm with parameter exploration order $\widehat{P_d}$ and arrangement $\widehat{P}$
GD^{ascA}	Explores parameter values in ascending order with arrangement $\mathcal{A} = \{V_p, F_p, F_s, P_s, P_{ti}, P_{tx}\}$
GD^{ascB}	Explores parameter values in ascending order with arrangement $\mathcal{B} = \{P_{tx}, P_{ti}, P_s, F_s, F_p, V_p\}$
GD^{ascC}	Explores parameter values in ascending order with arrangement $\mathcal{C} = \{F_s, P_{ti}, P_{tx}, V_p, F_p, P_s\}$
GD^{desD}	Explores parameter values in descending order with arrangement $\mathcal{D} = \{V_p, F_p, F_s, P_s, P_{ti}, P_{tx}\}$
GD^{desE}	Explores parameter values in descending order with arrangement $\mathcal{E} = \{P_{tx}, P_{ti}, P_s, F_s, F_p, V_p\}$
GD^{desF}	Explores parameter values in descending order with arrangement $\mathcal{F} = \{P_s, F_p, V_p, P_{tx}, P_{ti}, F_s\}$

Fig. 3 shows the objective function values normalized to the optimal solution for SA and greedy algorithms versus the number of states explored for a security/defense system for $|S| = 729$. Results indicate that GD^{ascA}, GD^{ascB}, GD^{ascC}, GD^{desD}, GD^{desE}, GD^{desF}, and GD converge to a steady state solution (objective function value corresponding to the operating state) after exploring 11, 10, 11, 10, 10, 9, and 8 states, respectively. We point out that we do not plot the results for each iteration and greedy algorithm variations for brevity, however, we obtained the results for all iterations and greedy algorithm variations. These convergence results show that GD converges to a final operating state slightly faster than other greedy algorithms, exploring only 1.1% of the design space. GD^{ascA} and GD^{ascB} converge to almost equal quality solutions as GD^{desD} and GD^{desE} showing that ascending or descending parameter values exploration and parameter arrangements do not significantly impact the solution quality for this application for $|S| = 729$.

Results also indicate that the SA algorithm outperforms all greedy algorithms and converges to the optimal solution after exploring 400 states or 55% of the design space. Fig. 3 also verifies the ability of our methodology to determine a good quality,

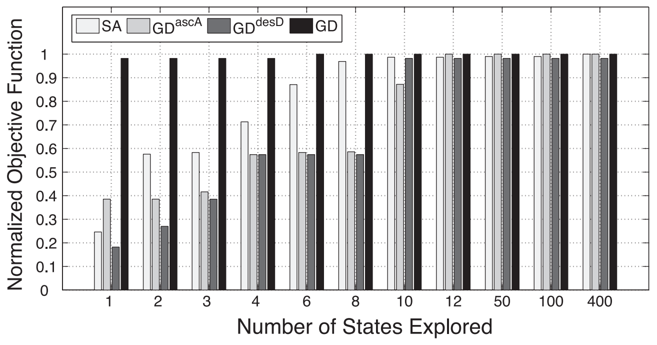


Fig. 3. Objective function values normalized to the optimal solution for a varying number of states explored for SA and the greedy algorithms for a security/defense system where $\omega_l = 0.25$, $\omega_t = 0.35$, $\omega_r = 0.4$, and $|S| = 729$.

Author's personal copy

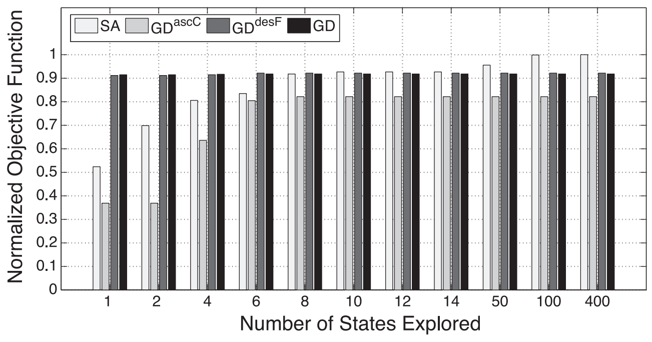


Fig. 4. Objective function values normalized to the optimal solution for a varying number of states explored for SA and greedy algorithms for a security/defense system where $\omega_l = 0.25$, $\omega_t = 0.35$, $\omega_r = 0.4$, and $|S| = 31, 104$.

near-optimal solution in one-shot that is within 1.4% of the optimal solution. GD achieves only a 1.8% improvement over the initial state after exploring 8 states.

Fig. 4 shows the objective function values normalized to the optimal solution for SA and greedy algorithms versus the number of states explored for a security/defense system for $|S| = 31, 104$. Results reveal that GD converges to the final solution by exploring only 0.04% of the design space. GD^{desD}, GD^{desE}, and GD^{desF} converge to better solutions than GD^{ascA}, GD^{ascB}, and GD^{ascC} showing that descending parameter values exploration and parameter arrangements $\mathcal{D}$, $\mathcal{E}$, and $\mathcal{F}$ are better for this application as compared to the ascending parameter values exploration and parameter arrangements $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$. This difference is because a descending exploration order tends to select higher tunable parameter values, which increases the throughput considerably as compared to lower tunable parameter values. Since throughput has been assigned a higher weight factor for this application than the lifetime, better overall objective function values are attained.

Comparing Fig. 4 and Fig. 3 reveals that the design space size also affects the solution quality in addition to the parameter value exploration order and parameter arrangement. For example, for $|S| = 729$, the ascending and descending parameter value exploration order and parameter arrangement results in comparable quality solutions, whereas for $|S| = 31, 104$, the descending parameter value exploration order results in higher quality solutions. Again, the SA algorithm outperforms all greedy algorithms and converges to the optimal solution for $|S| = 31, 104$ after exploring 100 states or 0.3% of the design space. Fig. 4 also verifies the ability of our methodology to determine a good quality, near-optimal solution in one-shot that is within 9% of the optimal solution. GD achieves only a 0.3% improvement over the initial state (one-shot solution) after exploring 11 states.

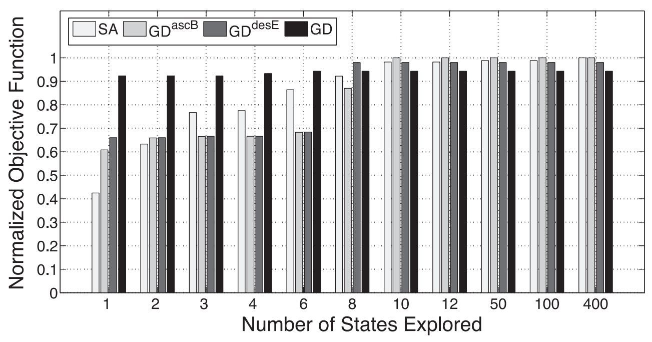


Fig. 5. Objective function values normalized to the optimal solution for a varying number of states explored for SA and greedy algorithms for an ambient conditions monitoring application where $\omega_l = 0.4$, $\omega_t = 0.5$, $\omega_r = 0.1$, and $|S| = 729$.

Results for a health care application for $|S| = 729$ reveal that GD converges to the final solution slightly faster than other greedy algorithms, exploring only 1% of the design space. The SA algorithm outperforms the greedy algorithm variants after exploring 400 states or 55% of the design space for $|S| = 729$, but the SA improvement over the greedy algorithm variants is insignificant as the greedy algorithm variants attain near-optimal solutions. Results indicate that the one-shot solution is within 2% of the optimal solution. GD achieves only a 2% improvement over the one-shot solution after exploring 8 states.

Results for a health care application for $|S| = 31, 104$ reveal that GD converges to the final solution by exploring only 0.0257% of the design space. The SA algorithm outperforms all greedy algorithms and converges to the optimal solution after exploring 100 states (0.3% of the design space). The one-shot solution is within 1.5% of the optimal solution. GD achieves only a 0.2% improvement over the one-shot solution after exploring 8 states.

Fig. 5 shows the objective function values normalized to the optimal solution versus the number of states explored for an ambient conditions monitoring application for $|S| = 729$. Results reveal that GD converges to the final solution slightly faster than other greedy algorithms, exploring only 1.1% of the design space. GD^{ascA}, GD^{ascB}, and GD^{ascC} converge to a higher quality solution than GD^{desD}, GD^{desE}, and GD^{desF} because the ascending exploration order tends to select lower tunable parameter values, which results in comparatively larger lifetime values as compared to higher tunable parameter values. This higher lifetime results in higher lifetime objective function values and thus higher overall objective function values. We observe that the greedy algorithm variants result in higher quality solutions after exploring more states than the one attained by GD, since GD^{ascA}, GD^{ascB}, GD^{ascC}, and GD^{desF} attain the optimal solution for $|S| = 729$. This observation reveals that other arbitrary parameter arrangements and exploration orders may obtain better solutions than GD but those arbitrary arrangements and exploration orders would not scale for different application domains with different weight factors and for different design space cardinalities. The SA algorithm outperforms GD^{desD}, GD^{desE}, and GD after exploring 400 states (55% of the design space). GD^{ascA}, GD^{ascB}, GD^{ascC}, and GD^{desF} attain optimal solutions. Our one-shot solution is within 8% of the optimal solution. GD achieves only a 2% improvement over the one-shot solution after exploring 8 states.

Results for an ambient conditions monitoring application for $|S| = 31, 104$ indicate that GD converges to the optimal solution after exploring 13 states (0.04% of design space), with a 17% improvement over the one-shot solution. The one-shot solution is within 14% of the optimal solution. GD^{ascA}, GD^{ascB}, and GD^{ascC} converge to a better solution than GD^{desD}, GD^{desE}, and GD^{desF} for similar reasons as $|S| = 729$. The one-shot solution is within 14% of the optimal solution. SA converges to a near-optimal solution after exploring 400 states (1.3% of the design space).

The results for different application domains and design spaces verify that the one-shot mode provides a high-quality solution that is within 8% of the optimal solution averaged over all application domains and design space cardinalities. These results also verify that improvements can be achieved over the one-shot solution during the improvement mode. The results indicate that GD may explore more states than other greedy algorithms if state exploration provides a noticeable improvement over the one-shot solution. The results also provide an insight into the convergence rates and reveal that even though the design space cardinality increases by 43x, both heuristic algorithms (greedy and SA) still explore only a small percentage of the design space and result in high-quality solutions. Furthermore, although SA outperforms the greedy algorithms after exploring a comparatively larger portion of the design space, GD still provides an optimal or near-optimal solution with significantly less design space exploration. These

Author's personal copy

results advocate the use of GD as a design space exploration algorithm for constrained applications, whereas SA can be used for relatively less constrained applications. We point out that both GD and SA are online algorithms for dynamic optimization and are suitable for larger design spaces as compared to other stochastic algorithms, such as MDP-based algorithms which are only suitable for restricted (comparatively smaller) design spaces [18].

6.2.4. Computational complexity

We analyze the relative complexity of the algorithms by measuring their execution time and data memory requirements. We perform data memory analysis for each step of our dynamic optimization methodology. Our data memory analysis assumes an 8-bit processor for sensor nodes with integer data types requiring 2 bytes of storage and float data types requiring 4 bytes of storage. Analysis reveals that the one-shot solution (step 1) requires only 150, 188, 248, and 416 bytes whereas step two requires 94, 140, 200, and 494 bytes for (number of tunable parameters N, number of application metrics m) equal to $(3, 2), (3, 3), (6, 3)$, and $(6, 6)$, respectively. GD in step 3 requires 458, 528, 574, 870, and 886 bytes, whereas SA in step 3 requires 514, 582, 624, 920, and 936 bytes of storage for design space cardinalities of 8, 81, 729, 31104, and 46656, respectively.

The data memory analysis shows that SA has comparatively larger memory requirements than the greedy algorithm. Our analysis reveals that the data memory requirements for all three steps of our dynamic optimization methodology increases linearly as the number of tunable parameters, tunable values, and application metrics, and thus the design space, increases. The analysis verifies that although all three steps of our dynamic optimization methodology have low data memory requirements, the one-shot solution in step one requires 361% less memory on average.

We measured the execution time for all three steps of our dynamic optimization methodology averaged over 10,000 runs (to smooth any discrepancies in execution time due to operating system overheads) on an Intel Xeon CPU running at 2.66 GHz [29] using the Linux/Unix `time` command [15]. We scaled these execution times to the Atmel ATmega1281 microcontroller [3] running at 8 MHz. Even though scaling does not provide 100% accuracy for the microcontroller runtime because of different instruction set architectures and memory subsystems, scaling provides reasonable runtime estimates and enables relative comparisons. Results showed that step one and step two required 1.66 ms and 0.332 ms, respectively, both for $|S| = 729$ and $|S| = 31, 104$. For step three, we compared GD with SA. GD explored 10 states and required 0.887 ms and 1.33 ms on average to converge to the solution for $|S| = 729$ and $|S| = 31, 104$, respectively. SA required 2.76 ms and 2.88 ms to explore the first 10 states (to provide a fair comparison with GD) for $|S| = 729$ and $|S| = 31, 104$, respectively. The other greedy algorithms required comparatively more time than GD because they required more design state exploration to converge than GD, however, all the greedy algorithms required less execution time than SA.

To verify that our dynamic optimization methodology is lightweight, we compared the execution time results for all three steps of our dynamic optimization methodology with the exhaustive search. The exhaustive search required 29.526 ms and 2.765 s for $|S| = 729$ and $|S| = 31, 104$, respectively, which gives speedups of 10× and 832×, respectively, for our dynamic optimization methodology. The execution time analysis reveals that all three steps of our dynamic optimization methodology requires execution time on the order of milliseconds, and the one-shot solution requires 138% less execution time on average as compared to all three steps of the dynamic optimization methodology. Execution time savings attained by the one-shot solution as compared to the three steps of our dynamic optimization methodology are 73% and 186% for GD and SA, respectively, when $|S| = 729$, and are 100% and 138% for GD and SA, respectively, when $|S| = 31, 104$. These results indicate that the design space cardinality affects the execution time linearly and our dynamic optimization methodology's advantage increases as the design space cardinality increases. We verified our execution time analysis using the `clock()` function [1], which confirmed similar trends.

Table 5
Energy consumption for the one-shot and the improvement mode for our dynamic optimization methodology. $\boldsymbol{IM}^{\boldsymbol{GD}}_{\boldsymbol{|S|=X}}$ and $\boldsymbol{IM}^{\boldsymbol{SA}}_{\boldsymbol{|S|=X}}$ denote the improvement mode using GD and SA as the online algorithms, respectively, for $|S| = X$ where $X = \{729, 31{,}104\}$. $\mathbf{ES}_{|\mathbf{S}|=\mathbf{X}}$ denotes the exhaustive search for $|S| = X$ where $X = \{729, 31{,}104\}$. $\mathbf{B_f}$ denotes the fraction of battery energy consumed in an operating mode and $\boldsymbol{R_l}$ denotes the maximum number of times (runs) our dynamic optimization methodology can be executed in a given mode depending upon the sensor node's battery energy.

Mode	$T_{exe}(ms)$	E_{dyn} (μJ)	$\mathcal{B}_f$	$\mathcal{R}_l$
One-shot	1.66	23.75	9.16×10^{-10}	1.1×10^{10}
$\mathcal{IM}^{GD}_{\lvert S\rvert=729}$	2.879	41.2	1.6×10^{-9}	629.13×10^{6}
$\mathcal{IM}^{SA}_{\lvert S\rvert=729}$	4.752	68	2.62×10^{-9}	381.2×10^{6}
$\mathcal{ES}_{\lvert S\rvert=729}$	29.526	422.52	1.63×10^{-10}	61.35×10^{6}
$\mathcal{IM}^{GD}_{\lvert S\rvert=31,104}$	3.322	47.54	1.83×10^{-9}	545.22×10^{6}
$\mathcal{IM}^{SA}_{\lvert S\rvert=31,104}$	4.872	69.72	2.7×10^{-9}	371.77×10^{6}
$\mathcal{ES}_{\lvert S\rvert=31,104}$	2765	39,570	1.53×10^{-6}	655×10^{3}

To further verify that our dynamic optimization methodology is lightweight, we calculate the energy consumption for the two modes of our methodology — the one-shot and the improvement modes with either a GD- or SA-based online algorithm. We calculate the energy consumption E^{dyn} for an Atmel ATmega1281 microcontroller [3] operating at $V_p = 2.7$ V and $F_p = 8$ MHz as $E^{dyn} = V_p \cdot I^a_p \cdot T_{exe}$ where I^a_p and T_{exe} denote the processor's active current and the execution time for the methodology's operating mode at (V_p, F_p), respectively (we observed similar trends for other processor voltage and frequency settings). We point out that we consider the execution time for exploring the first 10 states both for the GD- and SA-based online algorithms in our energy calculations as both the GD and SA algorithms attained near-optimal results after exploring 10 states both for $|S| = 729$ and $|S| = 31{,}104$. Table 5 summarizes the energy calculations for different modes of our dynamic optimization methodology as well as for the exhaustive search for $|S| = 729$ and $|S| = 31{,}104$. We assume that the sensor node's battery energy in our calculations is $E_b = 25, 920$ J, which is computed using (16). Results indicate that one-shot consumes 1679% and 166,510% less energy as compared to the exhaustive search for $|S| = 729$ and $|S| = 31{,}104$, respectively. Improvement mode using GD as the online algorithm consumes 926% and 83,135% less energy as compared to the exhaustive search for $|S| = 729$ and $|S| = 31{,}104$, respectively. Improvement mode using SA as the online algorithm consumes 521% and 56,656% less energy as compared to the exhaustive search for $|S| = 729$ and $|S| = 31{,}104$, respectively. Furthermore, our dynamic optimization methodology using GD as the online algorithm can be executed $545.22 \times 10^6 - 655 \times 10^3 = 544.6 \times 10^6$ more times than the exhaustive search and 173.45×10^6 more times than when using SA as the online algorithm for $|S| = 31{,}104$. These results verify that our dynamic optimization methodology is lightweight and can be theoretically executed on the order of million times even on energy-constrained sensor nodes.

7. Conclusions and future work

In this paper, we proposed a lightweight dynamic optimization methodology for WSNs, which provided a high-quality solution in one-shot using an intelligent initial tunable parameter value

Author's personal copy

settings for highly constrained applications. We also proposed an online greedy optimization algorithm that leveraged intelligent design space exploration techniques to iteratively improve on the one-shot solution for less constrained applications. Results showed that our one-shot solution is near-optimal and within 8% of the optimal solution on average. Compared with simulating annealing (SA) and different greedy algorithm variations, results showed that the one-shot solution yielded improvements as high as 155% over other arbitrary initial parameter settings. Results indicated that our greedy algorithm converged to the optimal or near-optimal solution after exploring only 1% and 0.04% of the design space whereas SA explored 55% and 1.3% of the design space for design space cardinalities of 729 and 31,104, respectively. Data memory and execution time analysis revealed that our one-shot solution (step one) required 361% and 85% less data memory and execution time, respectively, when compared to using all the three steps of our dynamic optimization methodology. Furthermore, one-shot consumed 1,679% and 166,510% less energy as compared to the exhaustive search for $|S| = 729$ and $|S| = 31{,}104$, respectively. Improvement mode using GD as the online algorithm consumed 926% and 83,135% less energy as compared to the exhaustive search for $|S| = 729$ and $|S| = 31{,}104$, respectively. Computational complexity along with the execution time, data memory analysis, and energy consumption confirmed that our methodology is lightweight and thus feasible for sensor nodes with limited resources.

Future work includes the incorporation of profiling statistics into our dynamic optimization methodology to provide feedback with respect to changing environmental stimuli. In addition, we plan to further verify our dynamic optimization methodology by implementing our methodology on a physical hardware sensor node platform. Future work also includes the extension of our dynamic optimization methodology to *global optimizations*, which will ensure that individual sensor node tunable parameter settings are both optimal for the sensor node and for the entire WSN.

Acknowledgements

This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) and the National Science Foundation (NSF) (CNS-0834080). 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 NSERC and the NSF.

References

[1] C++Reference Library in cplusplus.com, 2010, available from: http://cplusplus.com/reference/clibrary/ctime/clock/
[2] ATMEL AT86RF230 Low Power 2.4 GHz Transceiver for ZigBee, IEEE 802.15.4, 6LoWPAN, RF4CE and ISM Applications, ATMEL Corporation, San Jose, CA, 2010, available from: http://www.atmel.com/dyn/resources/prod_documents/doc5131.pdf
[3] ATMEL ATmega1281 Microcontroller with 256K Bytes in-System Programmable Flash, ATMEL Corporation, San Jose, CA, 2010, available from: http://www.atmel.com/dyn/resources/prod_documents/2549S.pdf
[4] Crossbow IRIS Datasheet, Crossbow Technology, Inc., San Jose, CA, 2010, available from: http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/IRIS_Datasheet.pdf
[5] MTS/MDA Sensor Board Users Manual, Crossbow Technology, Inc., San Jose, CA, 2010, available from: http://www.xbow.com/support/Support_pdf_files/MTS-MDA_Series_Users_Manual.pdf
[6] Dynamic Profiling and Optimization (DPOP) for Sensor Networks, 2010.
[7] H. Friis, A note on a simple transmission formula, Proceedings of the IRE 34 (1946) 254.
[8] A. Gordon-Ross, F. Vahid, N. Dutt, Fast configurable-cache tuning with a unified second-level cache, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 17 (January (1)) (2009) 80–91.
[9] H. Hamed, A. El-Atawy, A.-S. Ehab, On dynamic optimization of packet matching in high-speed firewalls, IEEE Journal on Selected Areas in Communications 24 (October (10)) (2006) 1817–1830.
[10] K. Hazelwood, M. Smith, Managing bounded code caches in dynamic binary optimization systems, ACM Transactions on Architecture and Code Optimization 3 (September (3)) (2006) 263–294.
[11] S. Hu, M. Valluri, L. John, Effective management of multiple configurable units using dynamic optimization, ACM Transactions on Architecture and Code Optimization 3 (December (4)) (2006) 477–501.
[12] D. Jung, T. Teixeira, A. Barton-Sweeney, A. Savvides, Model-based design exploration of wireless sensor node lifetimes, in: Proceedings of the ACM 4th European Conference on Wireless Sensor Networks (EWSN'07), Delft, The Netherlands, 2007.
[13] D. Jung, T. Teixeira, A. Savvides, Sensor node lifetime analysis: models and tools, ACM Transactions on Sensor Networks (TOSN) 5 (February (1)) (2009).
[14] S. Kogekar, S. Neema, B. Eames, X. Koutsoukos, A. Ledeczi, M. Maroti, Constraint-guided dynamic reconfiguration in sensor networks, in: Proceedings of the 3rd International Symposium on Information Processing in Sensor Networks (IPSN), ACM, Berkeley, CA, 2004, pp. 379–387.
[15] Linux Man Pages, 2010, available from: http://linux.die.net/man/
[16] S. Lysecky, F. Vahid, Automated application-specific tuning of parameterized sensor-based embedded system building blocks, in: Proceedings of the International Conference on Ubiquitous Computing (UbiComp), Orange County, CA, 2006, pp. 507–524.
[17] R. Min, T. Furrer, A. Chandrakasan, Dynamic voltage scaling techniques for distributed microsensor networks, in: Proceedings of the Workshop on VLSI (WVLSI), IEEE, Orlando, FL, 2000, pp. 43–46.
[18] A. Munir, A. Gordon-Ross, An MDP-based application oriented optimal policy for wireless sensor networks, in: Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis (CODES + ISSS), ACM, Grenoble, France, 2009, pp. 183–192.
[19] A. Munir, A. Gordon-Ross, An MDP-based dynamic optimization methodology for wireless sensor networks, IEEE Transactions on Parallel and Distributed Systems (TPDS) 23 (4) (2012) 616–625.
[20] A. Munir, A. Gordon-Ross, S. Lysecky, R. Lysecky, A one-shot dynamic optimization methodology for wireless sensor networks, in: Proceedings of IARIA IEEE International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM), Florence, Italy, 2010.
[21] H. Nguyen, A. Forster, D. Puccinelli, S. Giordano, Sensor node lifetime: an experimental study, in: Proceedings of IEEE International Conference on Pervasive Computing and Communications (PerCom'11), Seattle, WA, 2011.
[22] X. Ning, C. Cassandras, Optimal dynamic sleep time control in wireless sensor networks, in: Proceedings of the Conference on Decision and Control (CDC), IEEE, Cancun, Mexico, 2008, pp. 2332–2337.
[23] Datasheet SHT1x (SHT10, SHT11, SHT15) Humidity and Temperature Sensor, SENSIRION - The Sensor Company, Staefa, Switzerland, 2010, available from: http://www.sensirion.com/en/pdf/product_information/Datasheet-humidity-sensor-SHT1x.pdf
[24] K. Sha, W. Shi, Modeling the lifetime of wireless sensor networks, Sensor Letters 3 (2005) 126–135.
[25] A. Shenoy, J. Hiner, S. Lysecky, R. Lysecky, A. Gordon-Ross, Evaluation of dynamic profiling methodologies for optimization of sensor networks, IEEE Embedded Systems Letters 2 (March (1)) (2010) 10–13.
[26] S. Sridharan, S. Lysecky, A first step towards dynamic profiling of sensor-based systems, in: Proceedings of the Conference on Sensor, Mesh and ad hoc Communications and Networks (SECON), IEEE, San Francisco, CA, 2008, pp. 600–602.
[27] R. Verma, Automated application specific sensor network node tuning for non-expert application developers, M.S. thesis, Department of Electrical and Computer Engineering, University of Arizona, 2008.
[28] X. Wang, J. Ma, S. Wang, Collaborative deployment optimization and dynamic power management in wireless sensor networks, in: Proceedings of the International Conference on Grid and Cooperative Computing (GCC), IEEE, Changsha, Hunan, China, 2006, pp. 121–128.
[29] Intel Xeon Processor E5430, 2010, available from: http://processorfinder.intel.com/details.aspx?sSpec=SLANU
[30] L. Yuan, G. Qu, Design Space exploration for energy-efficient secure sensor network, in: Proceedings of the International Conference on Application-Specific Systems, Architectures, and Processors (ASAP), IEEE, San Jose, CA, 2002, pp. 88–97.
[31] C. Zhang, F. Vahid, R. Lysecky, A self-tuning cache architecture for embedded systems, ACM Transactions on Embedded Computing Systems 3 (May (2)) (2004) 407–425.

Arslan Munir received his B.S. in electrical engineering from the University of Engineering and Technology (UET), Lahore, Pakistan, in 2004, and his M.A.Sc. degree in electrical and computer engineering (ECE) from the University of British Columbia (UBC), Vancouver, Canada, in 2007. He received his Ph.D. degree in ECE from the University of Florida (UF), Gainesville, Florida, USA, in 2012. He is currently a postdoctoral research associate in the ECE department at Rice University, Houston, TX, USA. From 2007 to 2008, he worked as a software development engineer at Mentor Graphics in the Embedded Systems Division. He was the recipient of many academic awards including the gold medals for the best performance in

Author's personal copy

Electrical Engineering, academic Roll of Honor, and doctoral fellowship from Natural Sciences and Engineering Research Council of Canada (NSERC). He received a Best Paper award at the IARIA International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM) in 2010. His current research interests include embedded systems, cyber-physical/transporation systems, low-power design, computer architecture, multi-core platforms, parallel computing, dynamic optimizations, fault-tolerance, and computer networks.

Ann Gordon-Ross received her B.S. and Ph.D. degrees in computer science and engineering from the University of California, Riverside (USA) in 2000 and 2007, respectively. She is currently an associate professor of electrical and computer engineering at the University of Florida (USA) and is a member of the NSF Center for High Performance Reconfigurable Computing (CHREC) at the University of Florida. She is also the faculty advisor for the Women in Electrical and Computer Engineering (WECE) and the Phi Sigma Rho National Society for Women in Engineering and Engineering Technology. She received her CAREER award from the National Science Foundation in 2010 and Best Paper awards at the Great Lakes Symposium on VLSI (GLSVLSI) in 2010 and the IARIA International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM) in 2010. Her research interests include embedded systems, computer architecture, low-power design, reconfigurable computing, dynamic optimizations, hardware design, real-time systems, and multi-core platforms.

Susan Lysecky received both her M.S. and Ph.D. degrees in computer science from the University of California, Riverside in 2003 and 2006, respectively. She is currently an assistant professor in the Department of Electrical and Computer Engineering at the University of Arizona. She coordinates research efforts for the Ubiquitous and Embedded Computing lab, and her current research interests include embedded system design, with emphasis on self-configuring architectures, human-computer interaction, and facilitating the design and use of complex sensor-based system by non-engineers. She is a member of IEEE and ACM.

Roman Lysecky is an associate professor of electrical and computer engineering at the University of Arizona. He received his B.S., M.S., and Ph.D. in computer science from the University of California, Riverside in 1999, 2000, and 2005, respectively. His primary research interests focus on embedded systems design, with emphasis on dynamic adaptability, hardware/software partitioning, field-programmable gates arrays (FPGAs), and low-power methodologies. He has coauthored two textbooks on hardware description languages and holds one US patent. He received a CAREER award from the National Science Foundation in 2009, Best Paper Awards from the International Conference on Hardware-Software Codesign and System Synthesis (CODES+ISSS) and the Design Automation and Test in Europe Conference (DATE), and an Outstanding Ph.D. Dissertation Award from the European Design and Automation Association (EDAA) in 2006.