Accuracy-Guided Runtime Adaptive Profiling Optimization of Wireless Sensor Networks

Lu Ding, Adrian Lizarraga, Susan Lysecky, Roman Lysecky, Ann Gordon-Ross
Department of Electrical and Computer Engineering
University of Arizona, Tucson, AZ
luding@email.arizona.edu, adrianlm@email.arizona.edu, slysecky@ece.arizona.edu, rlysecky@ece.arizona.edu,
ann@ece.ufl.edu

***Abstract*—Optimization of sensor networks relies on accurate profiling information collected about the state of individual nodes and the network as a whole. A single fixed profiling methodology may incur significant overheads on the sensor network or produce inaccurate profiling results due to dynamic changes in application behavior at runtime. Alternatively, reconfiguring the profiling methodology at runtime in response to such changes can help maintain the accuracy of profiling results while minimizing the associated overheads. In this paper, we present a runtime adaptive profiling methodology that can adapt to runtime behavior of the network and preserve the accuracy of profiling data. This runtime adaptive profiling strategy further allows application experts to control the profiling accuracy, thereby providing a mechanism to tradeoff accuracy and overhead. Experimental results demonstrate that network, computational time, and power consumption overheads can be reduced by more than 50% compared to using a fixed profiling methodology while only missing 2% of profiled events.**

Keywords—runtime adaptive profiling, sensor networks, distributed embedded systems, dynamic optimization

I. INTRODUCTION

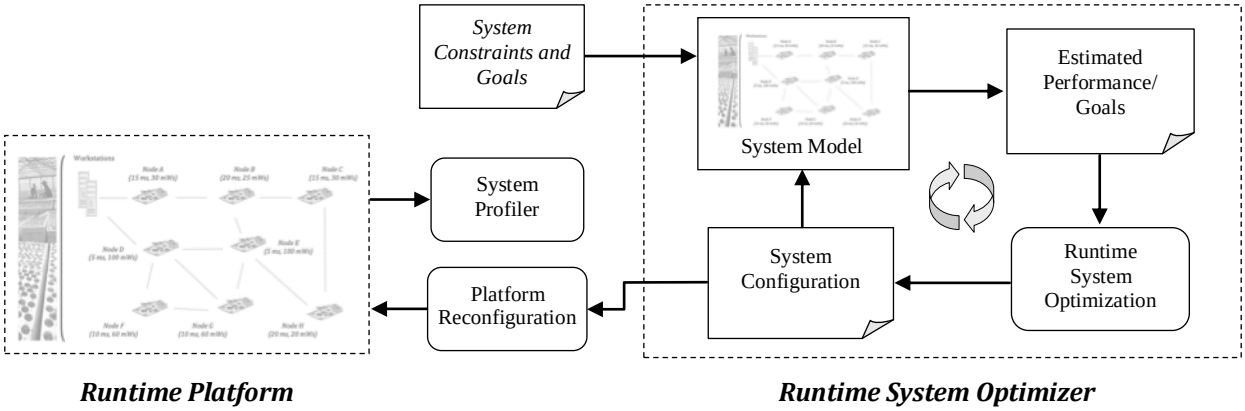
Wireless sensor networks (WSN) are used in a wide variety of applications. Given the uniqueness of applications, developers often spend considerable time configuring node and system-level parameters to meet specific application requirements such as lifetime, throughput, security, and reliability. Since application behavior can be highly affected by noise and unpredictable physical environments at runtime, it is difficult to find an optimal configuration at design time. Furthermore, a solution that may be perceived as optimal based on estimated application behavior at design time can often be found to be non-optimal at runtime due to dynamic changes in application behavior and states of sensor network. Thus, monitoring sensor network performance at runtime – the primary topic of this paper – is necessary for efficient and effective optimization of sensor networks.

While traditional hardware and software debugging methods can be used to analyze the execution behavior of WSNs, such tools are targeted at *platform developers* who have extensive understanding of hardware and software design methods. In contrast, *application experts* [1] utilizing these platforms are often not trained engineers, but rather scientists, biologists, or teachers who may lack the technical knowledge necessary to utilize these. Furthermore, these tools utilize static methods to monitor application behavior, and thus may incur significant overheads when used on WSN applications with dynamic behavior.

To alleviate this problem, we previously developed a dynamic profiling and optimization (DPOP) framework [17][16][12] to automatically monitor the runtime state of sensor networks and optimize node parameters in order to meet application expert specified requirements. Runtime monitoring of application behavior is currently achieved by employing a configurable profiling methodology that enables application experts to specify the manner in which profile information is collected and distributed. Application experts are able to configure profiling methodologies and evaluate their expected performance at design time. However, in the current implementation, profiling methodologies are not automatically adapted to changes in application behavior or changes in the external environment unless manually reconfigured by the application expert. Consequently, significant changes in application or environmental behaviors may render the initial profiling methodology unsuitable and lead to significant overheads. Thus, there is a need for a method to dynamically adapt the profiling methodology based on fluctuations in application or environmental behavior.

In this paper, we present an adaptive profiling optimization algorithm that optimizes the profiling methodology at runtime by analyzing the collected profile data. Additionally, the adaptive profiling approach allows application experts to specify simplified accuracy requirements that will be used to guide the optimization of the profiling methodology in order to reduce profiling overheads while maintaining the required level of accuracy. The rest of the paper is organized as follows. Section II summarizes related work in profiling of sensor networks. Section III provides an overview of the DPOP framework. Section IV highlights our existing static profiling methodology. In Section V, we present our adaptive profiling approach, highlighting both the interface that enables application experts to specify profiling requirements and our adaptive profiling optimization algorithm. Section VI presents experimental results that demonstrate the effectiveness of the proposed approach to reduce overhead

Figure 1. Overview of DPOP framework.



while maintaining accuracy. Finally, Section VII concludes and summarizes future research directions.

II. Related Work

Accurately capturing the runtime state of sensor networks is critical for dynamic optimization. Therefore, an accurate and efficient method to capture such information is needed. While many sensor network tools have been designed to monitor or manage sensor networks, most of them are used for debugging sensor networks at runtime and thus are quite low level. They are able to report the state of routing protocols and node failures but may not be able to profile application behavior. Clairvoyant [20] is a source-level debugger tool used to find bugs in application code. However, the user is not able to dynamically adjust its configuration at runtime in order to lower overheads. Tools like AVEKSHA [18] and FlockLab [2] are on-chip debuggers that inherently require additional hardware on sensor nodes. This approach is expensive and increases node power consumption. PAD [6] uses a lightweight packet marking scheme, an inference model, and an inference engine to generate a fault report of the entire sensor network. However, it only provides an overview of the overall network architecture and omits data regarding node status and application behavior. EnviroLog [9] is a tool used to achieve repeatability of asynchronous events in wireless sensor networks. It logs all issued function calls and their parameters in order to record module events. Similarly, Marionette [19] and L-SNMS are tools that allow a PC to access the functions and variables of a statically compiled program executing on a sensor node at run-time. Such low level information about function calls and variables cannot be easily used to analyze the status of nodes or the overall sensor network. SNMS [4] provides query and logging systems to collect attributes selected by the user and to log unexpected events. In order to use this method, the user must manually retrieve this information, as each node maintains its own logs. Thus, it is not suitable for dynamic optimization.

III. DPOP Environment and Profiler Module

The Dynamic Profiling and Optimization (DPOP) framework provides automated runtime profiling and system optimization for sensor networks, as shown in Figure 1. It seeks to simplify design tasks and improve accessibility to application experts by abstracting much of the underlying platform specific knowledge. At runtime, the system is profiled within its deployed environment. The collected profile information is used to create a representative model of the system. An optimizer continuously evaluates the system against expert-defined constraints and goals, and explores a variety of system configurations. Once a suitable configuration is determined, the configuration information is sent to the physical system for platform reconfiguration [16].

The runtime system optimization employs Application Expert Design Metric Specifications to relate optimization of low level parameters such as processor frequency, RF output power, etc., to high level system constrains and goals such as the expected sensor node or network lifetime, the time required to process a single packet, or the time required to process and respond to a sensor event. The application expert design metric specifications allow application experts to define the importance and range of acceptable values for each design metric. A fuzzy-logic inspired classification function is used to map raw design metric values to an Unacceptable, Fair, Good, and Superior fuzzy classification term. Meanwhile, the application expert uses English sentences to specify a set of fuzzy design fitness rules to determine the relative importance of each design metric and how they relate to the overall design quality. Details of the implementation of the optimization module, and evaluation of the underlying optimization algorithms can be found in [7][12][13][14].

Table I. Summary of profiling methodology overheads for the High Sample Transmission (HSTR), Multi-Sensor (MSEN), Dual-Mode Power Saving (DMPS), Communication Intensive (COMM) and Computation Intensive (COMP) applications.

(a) Network Traffic Overhead

	HSTR	MSEN	DMPS	COMM	COMP	***Avg***
Min.	6.00%	5.21%	6.00%	0.33%	6.00%	***4.71%***
Max.	19.65%	24.83%	66.17%	3.82%	14.29%	***25.75%***
Avg.	15.20%	19.41%	51.96%	0.34%	11.11%	***19.60%***

(b) Energy Consumption Overhead

	HSTR	MSEN	DMPS	COMM	COMP	***Avg***
Min.	0.91%	0.52%	2.35%	0.50%	0.45%	***0.95%***
Max.	1.19%	0.66%	2.59%	0.83%	1.15%	***1.28%***
Avg.	1.05%	0.62%	2.45%	0.66%	0.79%	***1.11%***

(c) Computational Time Overhead

	HSTR	MSEN	DMPS	COMM	COMP	***Avg***
Min.	14.79%	19.84%	5.21%	16.42%	5.91%	***12.43%***
Max.	112.27%	136.59%	32.99%	87.15%	20.32%	***77.86%***
Avg.	52.28%	68.95%	16.08%	45.49%	11.15%	***38.79%***

(d) Code Size Overhead

	HSTR	MSEN	DMPS	COMM	COMP	***Avg***
Min.	66.55%	71.25%	16.47%	8.20%	10.69%	***34.63%***
Max.	71.32%	75.90%	18.49%	11.52%	12.61%	***37.97%***
Avg.	68.29%	72.93%	17.40%	9.46%	11.64%	***35.94%***

A key component within the DPOP framework is the System Profiler that observes the execution behavior of the underlying sensor network platform. Our existing System Profiler [10] is composed of three components: 1) a code generator, 2) an estimation module, and 3) a profile data management module that aids application experts in the customization and evaluation of dynamic profiling methodologies. At design time, the code generator is used to simplify the task of incorporating the desired dynamic profiling methodology within the existing application code. With the newly augmented application, the estimation module determines the resulting overheads to help the application expert analyze the expense of profiling the sensor network. If the incurred overheads are acceptable, the application is deployed. At runtime, the profile data management module receives profile data packets generated from the sensor nodes, cluster heads, and base station, and parses each profile packet, aggregating the profile data into an intermediate format required by the optimization module.

With this previous effort, once the application expert has selected a specific profiling methodology, the deployed systems will utilize that methodology throughout its deployment. This may lead to unacceptable profiling overheads as the application behavior changes at runtime. To overcome this, we propose incorporating a profiler optimization module that will monitor the profile information and dynamically adapt the profiling methodology to reduce profiling overhead while still achieving a profiling accuracy goal defined by the application expert.

IV. Overview of Static Profiling Methodology

To profile a sensor-based application, profiling code must be incorporated with each node to monitor application level information. A configurable profiling strategy must consider: 1) what application parameters need to be profiled, 2) when to profile 3) how to profile, 4) whom to profile, and 5) at what profile granularity [16][3].

What to profile allows application experts to configure the profiler to observe a subset of parameters from the following application profile parameters: *sensor sampling rate, time between successive packets, current battery voltage, number of packets transmitted* by an individual node, and the *number of packets dropped* by an individual node. Whom to profile determines which nodes are profiled within the network. Options include profiling an *individual node*, a *cluster of nodes*, or profiling *the network* as a whole. When to profile is the frequency of profiling. Profiling can be performed *periodically* at each node or cluster of nodes by utilizing an internal timer or receiving *profile request packets* sent by the System Profiler. Alternatively, profiling can be trigged by *detection of flagged events* such as battery voltage decreasing, packet dropping, etc. How to profile defines the method of transmitting of the profile data back to the System Profiler. Profile data can be *piggybacking* onto application's data packet or sent back to Profiler as a *separate profile packet*. Finally, the granularity denotes the level of aggregation of the profiling information within the network and can be configured as *no aggregation*, *aggregate at the cluster head*, or *aggregate for all nodes*.

To understand how the profiler configuration affects the resulting overhead, we studied a variety of profiling methodologies chosen from the configurable options. Seven profile methodologies were selected to ensure that the corner cases are considered given the possible configuration options, as well as ensure that each of the different configurable parameters appeared in at least one of the profile methodologies explored. We also developed five

general applications with selected profiling methodologies on the Crossbow IRIS platform [11] to evaluate network traffic, computational time, energy consumption, and code size overheads using five representative applications. The *High Sample-Transmission* (HSTR) application models applications that require high sampling and packet transmission rate. The *Multi-Sensor* (MSEN) application samples multiple sensor inputs. The Dual-Mode Power Saving (DMPS) application has two working modes: a low power sleep mode and a high power, high-speed monitor mode. The *Communication Intensive* (COMM) application has heavy network traffic. The *Computation Intensive* (COMP) application models applications with high computational requirements.

Table I presents the average profiling overheads incurred by the seven representative profiling configurations across five application scenarios. The profiling methodologies incur reasonable energy consumption overheads ranging from 0.45% to 2.59%. However, network traffic, code size, and computation time can be as high as 66.17%, 75.90%, and 136.59% respectively. Not all the profile methodologies have good performance across all five applications. One profiling methodology may incur the lowest overhead for one application but incur highest overhead for another. This is attributed to the unique execution behavior of each application—i.e. no single profile methodology is best for all applications.

Furthermore, the application expert may not be able to find the most efficient profiling methodology at design time since application behavior will be affected by environment. The profiling methodology chosen by the application expert may lead to large overhead or inaccurate profile data. Instead, a runtime adaptive profiling approach is needed to satisfy requirements of various applications and adapt to changes in environmental or application behavior.

V. Adaptive Runtime Profiling

In the current framework, an application expert is able to easily customize the profiling methodology at design time based on characteristics of the application being implemented. However, the overhead incurred by profiling may be higher than expected, thus the initial profiling methodology specified by the application expert may be inefficient at runtime due to unanticipated dynamic application behaviors or environmental changes over time. Thus, a dynamic profiling approach is considered in which the framework is able to tune the profiling methodology based on the profile data collected. Moreover, in order to further reduce overhead, we additionally consider trading off accuracy of the profile data collected.

A. Adaptive Profiling Optimization Algorithm

Figure 2 provides an overview of the adaptive profiling optimization algorithm. An expert specifies the *level number*, which represents an optimization level between 1 and 5. These optimization levels are utilized to provide application experts with the ability to balance the tradeoffs of the accuracy of the profile data collected and the profile overhead incurred. The Level 1 optimization strategy incurs the lowest overhead and yields the lowest profile data accuracy. Alternatively, the Level 5 optimization strategy incurs the highest overhead but collects the highest profile data accuracy. Specifically, the *level number* is used to determine the tolerance of profile data noise, when to start optimization process, and the range of suitable profiling periods.

In addition, the level number is used to determine how quickly to modify the existing profile methodology (line 5). If the profiler observes *n* consecutive profile packets with the same data collected, where *n = (3 * level number)*, the optimization methodology is called to tune the underlying profile methodology employed. For example, if the application expert is monitoring the sensor sampling rate and chooses a Level 3 optimization strategy, the profiler will not execute the profile optimization until 9 consecutive profile packets contain the same sensor sampling rate. Thus, the higher the accuracy level, the slower the profiler is in reconfiguring the underlying profiling methodology. Similar data is detected by measuring the difference – or noise – between two packets. The profile data noise, *p*, is defined as the difference between the current and previous profile data (line 2). For example, if the previous sensor sampling rate is 4s and the current sensor sampling rate is 4.3s, then the noise is 0.3s. If noise is less than *p* of the previous profile data, the profiler considers the profile data in two profile packets equivalent.

The adaptive profiling optimization algorithm also defines and upper and lower bounds (line 8 - 9) for which the profile period can be adjusted. The maximum profile period is defined as *((6 - level number) * original profile period * 10)*. Alternatively, the minimum profile period is *(original profile period / level number)*. The coefficients in the equations were experimentally determined such that higher levels of accuracy result in smaller profiling periods, which endows the Runtime System Optimizer with a more accurate and up-to-date view of the network. Alternatively, lower levels of accuracy should give the Runtime System Optimizer a more fragmented view of the state of the sensor network, and thus limit energy consumed by re-optimization.

For each profiling packet received by the Profiler, an internal *profiling packet counter* is incremented. Once the *profiling packet counter* reaches *(3 * level number)*, the Profiler re-evaluates the existing profiling methodology and attempts to optimize the methodology employed. First, the *How to profile* parameter is evaluated (line 16 – 20). If this parameter can be configured to the piggybacking option, the network traffic overhead and power consumption overhead can be reduced. However, piggybacking significantly delays the transmission of profile data as it can only be sent along with application data. Thus, to guarantee that the profile data is transmitted in a timely manner, the maximum transmission interval, *max_tr*, of packet is compared to *(current profile*

Figure 2. Pseudocode for the adaptive profiling optimization algorithm.

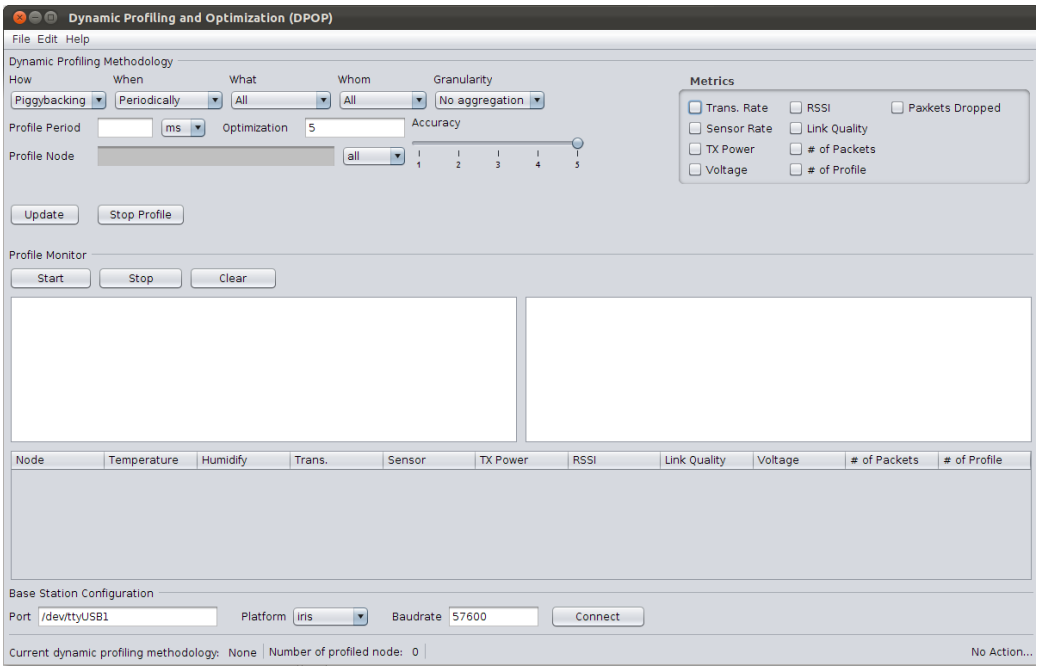
```
1   # tolerance of noise
2   p = 5 * (6 - level number)/100
3
4   # When to start profile
5   n = 3 * level number
6
7   # Range of profile period
8   MAX PROFILE PERIOD = (6 - level number) * ORIGINAL PROFILE PERIOD * 10
9   MIN PROFILE PERIOD = ORIGINAL PROFILE PERIOD / level number
10
11  Profiler increases counter by one.
12      if counter == n
13          counter is reset to 0.
14
15      # How to profile
16      (1) Find max transmission interval (MAX_TR) of application data packets in n profile packets
17      (2) if MAX_TR is < PROFILE PERIOD/10
18              How to profile = "PIGGYBACKING"
19          else
20              How to profile = "SEPARATE"
21
22      # When to profile
23      (1) Calculate power consumption (PT) of running the profile timer.
24      (2) Calculate power consumption (PR) of receiving a profiling request packet.
25      (3) if PT < PR
26              When to profile = "node periodically sends profile packets"
27          else
28              When to profile = "base station sends profiling request packet"
29
30      # What to profile
31          if one parameter does not change in n profile packets
32              Profiler stop profiling this parameter in next n profile packets.
33
34      # Profile period
35      For every parameter profiled by profiler
36      (1) Extracted profiled value in n consecutive profiling packets
37      (2) Find max, min, and average value
38      (3) if max - min < p * average and 2 * PREVIOUS PROFILE PERIOD < MAX PROFILE PERIOD
39              PROFILE PERIOD = 2 * PREVIOUS PROFILE PERIOD
40          else if PREVIOUS PROFILE PERIOD / 2 > MIN PROFILE PERIOD
41              PROFILE PERIOD = PREVIOUS PROFILE PERIOD / 2
42          else
43              PROFILE PERIOD remains same
```

period / 10). If *max_tr < (current profile period/10)* then a piggybacking scheme can be utilized. Otherwise, profile data is transmitted via separate packets.

The *When to profile* parameter similarly impacts the power consumption in both running a timer *(PT)* on each node to determine when to collect the profile data as well as in receiving profile requests *(PR)* from the Profiler. If the power consumption of running a timer locally is less than receiving the profile request packets from Profiler (line 25 – 32), the sensor node is configured to periodically send profile data to save power.

Lastly, changes to the profile period are also considered (line 35 – 43). Given *n* profile packets (where *n* is based on the optimization level defined by the application expert), the minimum, maximum, and average value of all profiled parameters are calculated. If the range of a profiled metric is less than the allowable tolerance (*p * average*), the Profiler increases the profile period to save power and computational time overheads, as these values do not demonstrate a high degree of variability. The Profiler will resume observation of these metrics after receiving *n* profile packets to again monitor whether change of these metrics is still within the allowable tolerance. If changes in the observed values again exceed the tolerance, the Profiler keeps profiling these metrics until suspend condition is satisfied.

Figure 3. Profiling optimization GUI.



B. *Dynamic Profiling Interface*

A user interface is provided to enable application experts to control and monitor the profile methodology employed, as well as monitor the profile data collected. The interface consists of two main views: a Dynamic Profiling Methodology view and a Profile Monitor view.

Figure 3 illustrates the Dynamic Profiling Methodology view from which an application expert is able to specify the initial profiling methodology including how to profile, when to profile, what to profile, whom to profile and the granularity. The overhead estimation module returns the estimated overhead corresponding to the selected profiling methodology. Based on this estimation, application experts can reconfigure their profiling methodology if the estimated overheads do not meet the application requirements. If application experts choose to use periodic profiling, the profile period must be set using the provided entry field.

By selecting the desired nodes in the *Whom to Profile* heading and providing node IDs, the application expert is able to profile specific nodes in the network. The application expert can also profile metrics in which they are interested by selecting the metrics of interest under the *What to Profile* heading. To enable or disable the profile methodology optimization, the application expert only need check or uncheck the *Optimization* box. If profile methodology optimization is enabled, the accuracy level slider can be used to control the overhead of the optimized profiling methodology and in turn the accuracy of profiled data where Level 5 yields the highest overhead/accuracy and Level 1 yields the lowest overhead/accuracy values.

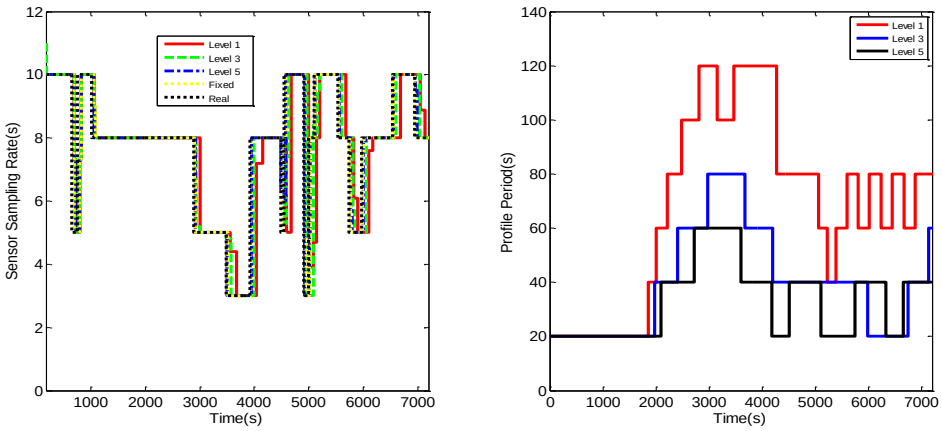
The Profile Monitor View parses the profile data collected from the sensor network at runtime and shows the current state of each node including with metrics such as the TX power, battery voltage, and so on. Any change in profiling methodology made by the adaptive profiling optimization algorithm is also displayed in the appropriate text area of this view.

VI. Experiment

To evaluate the accuracy/overhead tradeoffs of the adaptive runtime profiling methodology, we use the Arizona Transaction-level Simulator for Sensor Networks (ATLeS-SN), which is a simulation platform built using SystemC. Through the principles of transaction-level modeling [5], ATLeS-SN offers a modular simulator structure that enables developers to create custom wireless sensor network simulations by specifying custom sensor node and network architectures. Specifically, developers can specify different models for the network's environment and transmission medium, or a sensor node's transceiver, MAC layer, networking layer, application, sensor and other internal components. We refer the interested reader to [8] for an in-depth overview of the simulator and its features.

Within the ATLeS-SN simulation, the sensor node component was modified to include the aforementioned adaptive profiling functionality. The base station component is linked with our dynamic profiling user interface to enable profiling methodology optimization at runtime. With the

Figure 4. A snap shot of profiled (a) sensor sampling rate and (b) profile period



(a) (b)

help of the system monitoring components within the simulator, we are able to monitor power consumption of each node and gauge the accuracy achieved by the various optimization levels.

A. Simulation Results

Four applications with different behaviors are developed to evaluate the performance of our adaptive runtime profiling methodology. We first consider a *Constant Sensor Sampling Rate* (CONST) application. This application has a constant sampling rate at 10s and its simulation was run for 5 hours. Next a *Dual Working Mode* (DWM) application with two working modes is developed. The fast sensor sampling rate operates between 5s to 12s, whereas the slow sensor sampling rate operates between 5 minutes to 10 minutes. The DWM application automatically switches between the two working modes every 20 minutes. We ran a simulation for this application for 48 hours. Additionally, a *Purely Random Fast Sensor Sampling Rate* (RDFT) application with random sensor sampling rate between 2s and 15s is developed. We simulated this application for 5 hours. Lastly, a *Purely Random Slow Sensor Sampling Rate* (RDSL) application with random sensor sampling rates between 5 minutes and 10 minutes is considered. We ran the simulation for 48 hours.

The sensor sampling rate, packet transmission interval, transmission power, RSSI and battery voltage are monitored within the simulation framework. The application behavior changes are limited to sensor sampling rate for simplicity. We are able to compare the accuracy of the different profiling accuracy levels by determining whether profiling captures all sensor sampling rate changes. Random noise is also added to the sensor sampling rate in order to simulate environmental interference. Even if the sensor sampling rate does not change over the application's deployment, the profiled sensor sampling rates may be different as the profiling period is adapted by the optimization methodology. Within the simulation framework, we also collected the network traffic, computational time, and power consumption overhead incurred by the optimized profiling methodology on one sensor node.

Figure 4(a) shows the Level 1, Level 3 and Level 5 profiled sensor sampling rate and the actual sensor sampling rate for the DWM application. The plot in Figure 4(b) superimposes the varying profile periods corresponding to Level 1, Level 3, and Level 5 profiling accuracy levels. As the DWM application transitions to a fast sensor sampling working mode (73000s) the sensor sampling rate switches from 800s to 10s. In the Level 3 and Level 5 optimization strategies, the profile period is able to quickly update the underlying profiling methodology and catch up to the change in working modes thereby avoiding missing a large amount of changes in application behavior. On the other hand, the Level 1 optimization strategy uses a 5000s profile period when the DWM application changes working modes. The profiler does not profile the node until 75600s. All of the application behavior changes between 73000s and 75600s are missed.

As shown in the trace, the Adaptive Profiling Optimization algorithm is able to adapt as the application changes, increasing the profile period when the application is executing the fast sampling working mode and decreasing the profile period when application is executing the slow sampling working mode. For an application with a constant sensor sampling rate, the Level 5 accuracy level incurs more overhead than a fixed profiling methodology. Thus, in this application scenario a Level 1 profiling accuracy level has the best performance because it mitigates overheads without missing any application events.

Table II shows the maximum, minimum, and average overhead savings for Level 1, Level 3, and Level 5 profiling

Table II. Comparison of network, computational time, and power overheads and observed events for Purely Random Fast Sensor Sampling Rate (RDFT), Purely Random Slow Sensor Sampling Rate (RDSI), Dual Working Mode (DWM), and Constant Sensor Sample Rate (CONST) applications for a simulated period of 48 hours.

(a) Network Traffic (packets)

	RDFT	RDSL	DWM	CONST	*Avg %*
L1	52	54	143	62	23.59%
L3	287	175	591	71	19.89%
L5	866	287	2153	756	6.55%
Fixed	449	287	4319	449	N/A

(b) Power Consumption (mW)

	RDFT	RDSL	DWM	CONST	***Avg %***
L1	0.1	0.1	0.26	0.11	23.66%
L3	0.53	0.32	1.08	0.13	20.15%
L5	1.59	0.53	3.95	1.39	7.46%
Fixed	0.82	0.53	7.92	0.82	N/A

(c) Computational Time (ms)

	RDFT	RDSL	DWM	CONST	***Avg %***
L1	17.9	18.6	49.2	21.3	23.70%
L3	98.7	60.2	203.3	24.4	20.30%
L5	297.9	98.7	740.6	260.1	8.00%
Fixed	154.5	98.7	1485.7	154.5	N/A

(d) Observed Events (events)

	RDFT	RDSL	DWM	CONST	***Avg %***
L1	27	38	87	0	14.66%
L3	45	72	156	0	6.49%
L5	46	73	285	0	-2.36%
Fixed	46	72	251	0	N/A

accuracy levels. Additionally, the total number of observed events for each level is compared to the fixed profiling methodology across all applications. As previously stated, the fixed profiling methodology indicates that the profiling methodology is not optimized during simulation. As expected, Level 5 profiling accuracy incurs the highest network, computation time and power consumption overhead for all applications. For the RDFT application, all sensor sampling rate changes are captured by the Level 5 optimization profiling methodology. However, Level 5 incurs higher overheads than the fixed profiling methodology because the Level 5 optimization interprets the noise added to the sensor sampling rate as an actual change in sensor sampling rate, and consequently reduces profile period to acquire more accurate profile data. For the RDSI application, the Level 5 profiling accuracy level shows similar performance to the fixed profiling methodology. For the DWM application, the Level 5 profiling accuracy level sends 2166 less profile packets and reduces the number of missed events to 6 compared to fixed profiling methodology. This is attributed to the dynamic adaptation of the profile period in order to fit current application behavior.

In the Level 3 optimization setting, the RDFT application sends 167 less profile packets while only missing one application event. For the RDSL application, the Level 3 setting sent only 175 profile packets and missed 8 applications events. Compared to the Level 5 setting and the fixed profiling methodology, it both maintains the accuracy of profile data and reduces profiling overhead.

The Level 1 setting sacrifices the accuracy of profile data in order to lower profiling overheads for all applications. This optimization level missed 70% of application events in the DWM application. Because the profile period is extended during the slow sensor sampling rate working mode, the profiler is not able to quickly adapt to the changing application. When the application switches to the fast sensor sampling rate working mode, the profiler maintains the original extended profile period and therefore misses a significant number of application events. Thus, for applications with slow changing behavior, lower levels of profiling accuracy reduce the incurred overhead and yet are able to detect the same number of application events when compared to higher levels of profiling accuracy. For applications in which behavior changes rapidly, higher level settings can maintain the accuracy of profile data and reduce overheads at same time.

B. Physical Measurement

To further evaluate the adaptive runtime application, the adaptive profiling methodology is implemented on a physical platform. A Lighting Sensor application is implemented on the Crossbow IRIS platform [11]. Sensor nodes were placed in a black box. The only light source was a set of four LEDs. By changing how many LEDs are illuminated, we can easily control environmental changes and replay the experiment scenario. We ran the measurement for 2 hours for each profiling accuracy level.

Figure 4(a) shows the profiled sensor sampling rates based on Level 1, Level 3 and Level 5 accuracy settings. These traces are superimposed upon the real sensor sampling rate changes. This figure shows the sensor sampling rate remains at 8s from time 1100s to 2800s. Figure 4(b) shows the optimized profile period, demonstrating that the profiler is able to detect the same sensor sampling rate in consecutive

Table III. Comparison of network traffic, computational time, and power overheads and missed events for the Light Sensor application given various accuracy levels (L1 – L5) for physically measured periods of 2 hours.

	Network Traffic (packets)	Computational Time (ms)	Power Consumption (mW)	Missed Events
L1	159	54.70	0.29	0
L2	187	64.33	0.34	0
L3	220	75.68	0.40	0
L4	232	79.81	0.43	0
L5	240	82.56	0.44	0
Fixed	360	123.84	0.66	0

profiling packets and start optimization of the profiling methodology around 2000s. Moreover, in Figure 4(b) the Level 1 optimization strategy starts to increase the profile period because it only needs 3 consecutive profiling packets with the same sensor sampling period in order to re-evaluate the profiling methodology. For this reason, the adaptive profiler is able to quickly increase the profile period to the largest value. The Level 5 accuracy setting is only able to extend the profile period to 60s because it requires more equivalent profile packets to begin re-evaluating the profiling methodology. When the application behavior changes rapidly, the profiler starts to decrease the profile period to avoid miss application events.

Table III shows physical measurements of overheads associated with the Level 1 to Level 5 profile accuracy settings. The last row (Fixed) shows the overhead of using a fixed profiling methodology. Because the sensor sampling rate does not show rapid changes during the two hour experiment, all application events are captured by using all profiling accuracy levels. Compared to the fixed profiling methodology, the Leve1 3 setting shows savings of 55% for the network traffic overhead and savings of 58% for the computational time and power consumption overheads.

C. *Effects of Profiling Accuracy on Configuration Optimization*

The DPOP framework consists of two discrete components: the System Profiler, which is the focus of this paper, and the Runtime System Optimizer, which is a parallel research thrust [7]. The Runtime System Optimizer utilizes the profile data produced by the System Profiler in conjunction with application expert's specifications in order to generate an optimal sensor node configuration—i.e. voltages and frequencies.

Given that the primary aim of the DPOP framework is to generate and maintain optimal sensor node configurations, determining the effects of the proposed Adaptive Profiling Optimization algorithm on the quality of the produced configurations is a primary concern. Specifically,

Table IV. Comparison of the average *AvgFitnessScore* over 5 different applications using various profiling accuracy levels (L1 – L5).

	Average *AvgFitnessScore*
L1	9.51%
L2	6.09%
L3	5.70%
L4	5.19%
L5	4.24%

experimental data should support the hypothesis that increasing the accuracy level of the Adaptive Profiling Optimization algorithm results in sensor node configurations that are closer to the true optimal configuration at all times during application runtime; where the true optimal configuration corresponds to the configuration produced by the Runtime System Optimizer when the profile data is completely accurate.

In order to verify this hypothesis, the Adaptive Profiling Optimization algorithm was executed at every accuracy level for 48 hours in order to generate a set of data points *(t, pf)* consisting of a profile packet, *pf*, occurring at time, *t*. Each data point is then utilized by the Runtime System Optimizer in order to determine an optimal sensor node configuration and generate an output data point *(t, Fitness)* consisting of a configuration fitness score, *Fitness*, at time *t*. Each output data point is then compared to the true optimal configuration fitness score in order to calculate the average configuration fitness score deviation, *AvgFitnessDeviation*. The value of this metric represents the average deviation in configuration fitness at every time step, *t*, due to inaccuracies in the profile data. Table IV shows the average *AvgFitnessDeviation* for five applications. As expected, increasing the profiling accuracy level produces smaller deviations in configuration fitness score. The average *AvgFitnessDeviation* decreases from 9.51% at Level 1 to 4.24% at Level 5.

VII. Conclusion and Future Work

Static profiling methodologies may incur tremendous overheads due to application behavior changes at runtime. In this paper, we presented an adaptive runtime profiling methodology that uses the accuracy of profile data as a tradeoff to reduce profiling overhead. The Profiler dynamically adapts the profiling methodology's configuration in order to meet the application expert's accuracy requirements. Applying this method to five applications, we are able to reduce the network traffic, the computational time, and the power consumption overheads by 56%, 53%, and 53%, respectively, while only missing 2% of application events when compared to a non-adapting static profiling methodology. Currently, application experts can only set the expected accuracy level of the profile data. It is

hard to evaluate whether the produced profile data meets the specified accuracy without information about missed application events. Thus, returning the number of missed events and the accuracy of the profile data is a needed feature left for future work.

VIII. REFERENCES

[1] Bai, L., R. Dick, P. Dinda, "Archetype-based Design: Sensor Network Programming for Application Experts, not just Programming Experts," International Conference on Information Processing in Sensor Networks (IPSN), 2009, pp. 85-96.

[2] Beutel, J., R. Lim, A. Meier, L. Thiele, C. Walser, M. Woehrle, M. Yuecel, "The FlockLab testbed architecture," ACM Conference on Embedded Networked Sensor Systems (SenSys), 2009, pp. 415-416.

[3] Ding, L., A. Sheony, S. Lysecky, R.Lysecky, A. Grodon-Ross, "Application-Sepcfic Customization of Dynamic Profiling Mechanisms for Sensor Networks," ACM Transactions on Embedded Computing Systems (TECS), Submitted.

[4] Dutta, P., D. Culler, "System Software Techniques for Low-Power Operation in Wireless Sensor Networks", International Conference on Computer-Aided Design (ICCAD), 2005, pp. 925-932.

[5] Hiner, J., A. Shenoy, R. Lysecky, S. Lysecky, A. Gordon-Ross, "Transaction-Level Modeling for Sensor Networks Using SystemC", IEEE International Conference on Sensor Networks, Ubiquitous and Trustworthy Computing (SUTC), 2010, pp. 197-204.

[6] Liu, Y., K. Liu, M. Li, "Passive Diagnosis for Wireless Sensor Networks," IEEE/ACM Transactions on Networking (TON), 2010, Vol. 18, No. 4, pp. 1132-1144.

[7] Lizarraga, A., R. Lysecky, S. Lysecky, A. Grodon-Ross, "Dynamic Profiling and Fuzzy Logic Optimization of Sensor Networks Platforms," ACM Transactions on Embedded Computing Systems (TECS), To appear.

[8] Lizarraga, A., L. Ding, S. Lysecky, R. Lysecky, A. Gordon-Ross, "ATLeS-SN A Modular Simulator for Wireless Sensor Networks". Design Automation for Embedded Systems (DAEM), Submitted.

[9] Luo, L., T. He, G. Zhou, L. Gu, T. F. Abdelzaher, J.A. Stankovic, "Achieving Repeatability of Asynchronous Events in Wireless Sensor Networks with EnviroLog," IEEE International Conference on Computer Communication (INFOCOM 2006), 2006, pp. 1-14.

[10] Lysecky, S., F. Vahid, "Automated Application-Specific Tuning of Parameterized Sensor- Based Embedded System Building Blocks," International Conference on Ubiquitous Computing (UbiComp), 2006, pp. 507-524

[11] Memsic Corporation, IRIS Wireless Measurement System, http://www.memsic.com/products/wireless-sensornetworks/wireless-modules.html

[12] Munir, A., A. Gordon-Ross, "An MDP-based Application Oriented Optimal Policy for Wireless Networks," Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2009, pp. 183-192.

[13] Munir, A., A. Gordon-Ross, S. Lysecky, R. Lysecky, "A Lightweight Dynamic Optimization Methodology for Wireless Sensor Networks," IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), 2010, pp. 129-136.

[14] Munir, A., A. Gordon-Ross, S. Lysecky, R. Lysecky, "A One-Shot Dynamic Optimization Methodology for Wireless Sensor Networks. International Conference on Mobile Ubiquitous Computing, Systems, Services (UBICOMM), 2010, pp. 287-293.

[15] Ramanathan, N., K. Chang, R. Kapur, L. Girod, E. Kohler, D. Estrin, "Sympathy for the Sensor Network Debugger," International Conference on Embedded Networked Sensor Systems, 2005, pp. 255-267.

[16] Shenoy, A., J. Hiner, S. Lysecky, R. Lysecky, A. Gordon-Ross, "Evaluation of Dynamic Profiling Methodology for Optimization of Sensor Networks. IEEE Embedded Systems Letters, 2010, Vol. 2, No. 1, pp. 10-13.

[17] Sridharan, S., S. Lysecky, "A First Step towards Dynamic Profiling Sensor-Based Networks," Sensor and AdHoc Communications and Networks (SECON), 2008, pp. 600-602.

[18] Tancreti., M., M. Sajjad Hossain, S. Bagchi, V. Raghunathan, "Aveksha: a hardware-software approach for non-intrusive tracing and profiling of wireless embedded systems," ACM Conference on Embedded Networked Sensor Systems (SenSys), 2011, pp. 288-301.

[19] Whitehouse, K., G. Tolle, J. Taneja, C. Sharp, S. Kim, J. Jeong, J. Hul, P. Dutta, D. Culler, "Marionette: Using RPC for Interactive Development and Debugging of LWireless Embedded Networks," Information Processing in Sensor Networks (IPSN), 2006, pp. 416-423.

[20] Yang, J., M. Soffa, L. Selavo, K. Whitehouse. Clairvoyant: A Comprehensive Source-Level Debugger for Wireless Sensor Networks. ACM Conference on Embedded Networked Sensor Systems (SenSys), pp. 189-203, 2007.