

# A First Step Towards Dynamic Profiling of Sensor-Based Systems

Srihari Sridharan, Susan Lysecky

Department of Electrical and Computer Engineering

University of Arizona

sriharis, slysecky@ece.arizona.edu

## Abstract

*Application specific tuning has been shown to be beneficial for a variety of platforms, sensor-based systems are no exception. However, accurately capturing external stimuli or modeling application specific stimuli remains a challenge. Tuning a sensor-based system to erroneous or incomplete application information can limit the optimization achieved or even negatively impact the resulting system performance. We propose to obtain accurate application stimuli by dynamically observing the sensor-based system in the intended deployment location, storing application characteristics for later analysis or optimization.*

## 1. Introduction

The advances in sensor-based networks have enabled a plethora of application possibilities, ranging from industrial equipment monitoring, military surveillance, health monitoring, and weather forecasting [7][9]. With the diversity of application possibilities it is not surprising to see diversity in high-level application requirements. For example, a disaster response application requires high responsiveness and reliability to survey damage or detect survivors, but may only be intended to operate for days or weeks. Conversely, an automated vineyard irrigation system is intended to be a long lasting system operating on the order of years. However, responsiveness requirements are less demanding as the application monitors weather. Requirements within the application itself can also change over time.

To achieve the low cost per node afforded by economy of scale, sensor network node manufacturers often produce nodes in large volumes providing software configurable node-level parameters [4] such as voltage levels, processor modes (e.g., active, idle, sleep), or configurable baud rates to ensure the usability of the same node across a wide variety of applications. Application developers must carefully balance these node-level parameters along with protocol-level and system-level parameters as numerous previous works have shown the effects these parameters can have on high-level metrics such as energy consumption or reliability [2][5][13][15].

While many simulators exist that robustly model the underlying hardware and communication channels, the application level information remains difficult to capture. The application developer is left to specify application behavior as an input file [6][12], mathematic models [11][14], or through synthetic data generation [16]. While sensor network emulators have also appeared that enable control of particular sensor nodes providing controllability and repeatability for testing, evaluating, and comparing sensor networks [10], the proposed framework is ideally suited to developing and benchmarking sensor networks in a reliable and repeatable manner. It's use in deployed systems would incur significant overhead. Similarly, application layer tools exist that provide server software services. Specifically of interest is the support of real-time monitoring of a deployed sensor network, providing visualization and data collection capabilities. While the proposed real-time monitoring techniques

are applicable to monitoring a system in the field, as the authors themselves point out, the overhead of these services can be significant [3].

Currently, a dynamic and efficient method to capture external application specific stimuli remains elusive. As a result, static analysis and configuration tools do not have a holistic view of the system when optimizing the underlying platform. We propose to tackle this problem by exploring methods to dynamically profile a sensor-based network in-situ, to provide an accurate view of the environment and records changes within the environment over time.

## 2. Dynamic Profiling Environment

The environment in which the application is deployed plays an important role in performance of the underlying platform. However, it is extremely difficult to precisely predict the actual environment at design time. Thus, we propose to remove the burden of application characterization from the developer and instead utilize a dynamic profiling environment to obtain an accurate view of the application behavior. By dynamically profiling the environment the guesswork of creating a "good" benchmark is eliminated. Additionally, dynamic profiling is able to monitor changes in the environment over time, changes within the platform itself, or capture unanticipated situations. Dynamic profiling should ensure non-intrusive profiling, low-overhead in terms of additional data transmitted through network or power consumption, and maintain an accurate record of environmental stimuli. Figure 1 provides a block diagram of the proposed dynamic profiling environment composed of three main modules, the sensor-based application, end user specification, and the profiler module.

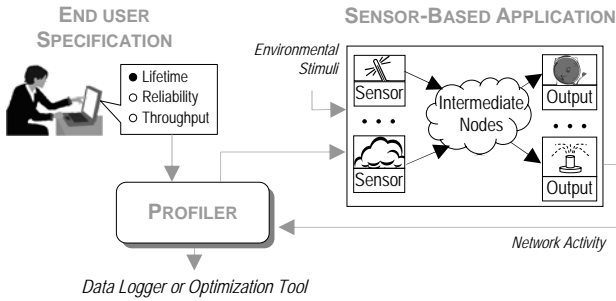
### 2.1 Sensor-Based Application

The sensor-based application corresponds to the physical implementation of an application deployed within its intended environment. While a variety of sensor-based platforms are available, we chose to begin with the eBlock platform [1] due to its light-weight communication protocol and point-to-point communication channels. As we form and evaluate the initial profiling methodologies, we keep mindful that these methodologies are intended to scale to more robust platforms.

The eBlock platform is targeted towards non-expert users and consists of fixed-function nodes, which fall into one of three categories - sensor, intermediate, or output nodes. Sensor nodes monitor the environment and detect events such as motion or light. Intermediate nodes perform logic transformations (AND, OR, INV), state based transformations (toggle, trip, prolong), or route data within the network. Output nodes provide stimuli, such as blink LEDs or control electric relays.

In this paper, the sensor-based application is an eBlock application constructed by an application developer. The system is placed into the desired environment to monitor external stimuli. The profiler module is connected to the sensor nodes indicating when to actively collect data for profiling. If the application is instructed to profile, information is gathered and

Figure 1: Dynamic profiling environment.



propagated through the network. The output nodes serve as sinks to collect the information generated by the application, and are also connected to the profiler module to relay the information to the profiler for storage.

## 2.2 End User Specification

Which network statistics to collect is application dependent. A developer however is not typically interested in low-level metrics such as the frequency a particular sensor is activated or the number of packets transmitted through the network. Rather, they are interested in the effect these low-level metrics have on lifetime, reliability, throughput, and so on. Thus an intermediate interface needs to be provided which lists a set of high-level metrics an application developer can choose to track. Then depending on the selection, the profiler module is responsible for determining which low level metrics are actually tracked in the sensor-based system. A similar approach can be utilized as described in [8], which provides an application specific tuning tool for non-expert end users by obscuring the low-level metrics in favor of high-level design metrics. We note that in this work, we only consider the lifetime metric. Thus, the information collected from the sensor-based application corresponds solely to sensor activity.

## 2.3 Profiler Module

The profiler module is responsible for orchestrating which network statistics to track, when to track these statistics, and storing the resulting data. To initiate profiling, the profiler module sends a control packet to the sensor nodes within the application. Again, the data generated by the application is dependent on the high-level metrics (e.g. lifetime or throughput) selected by an application developer. In this paper, because we are interested in the lifetime metric we track sensor activity, specifically the frequency at which sensors detect external stimuli, the relative time between such events, and the disbursement of activity across all sensor nodes.

Once profiling is activated, sensor nodes indicate when activity occurs by generating profiling packets that are propagated through the network. The existing network structure is utilized to transmit the profiling packets to simplify data collection. However, the amount and frequency of profile data transmitted through the network needs to be carefully determined. Introduction of excessive data transmissions or processing by nodes can severely impact the existing network's lifetime and performance. The following sections propose several methodologies to minimize the overhead incurred by the introduction of profiling packets.

As profiling packets are propagated to the output nodes, the profiling packets are sent back to the profiler module. The

information gleaned from profiling packets is stored in a memory maintained by the profiler module. Currently, the information is simply stored to be accessed later by a secondary analysis or optimization tool. It is feasible for the profiler module to conduct its own analysis and dynamically optimize the underlying platform to meet the needs of the changing environment; however this is currently left as future work.

### 2.3.1 Piggybacking Profiling Methodology

The piggybacking methodology modifies the existing communication protocol by utilizing data packets transmitted through the network. By using packets that are required regardless of profiling to maintain normal network functionality, overhead is minimized. A single bit within the data packet is designed as the profiling bit, indicating whether a packet should be forwarded to the profiler module. The advantage of the piggybacking methodology is its simplicity. No new packets are introduced into the network and all sensor activity is tracked while the profiling mode is activated. The drawback of this methodology is the loss of a bit within the data packet. For some applications this is not detrimental, these applications simply do not need the entire range of values provided (or accuracy if the bit is removed from the fractional representation). Additionally, in the piggybacking methodology the profiler module cannot discern profiling packets generated between multiple sensor nodes. Thus, to determine activity specific to a single sensor node within applications consisting of multiple sensor nodes, profiling of nodes must be activated one sensor node at a time. While the profiler module can now determine which sensor node is generating profiling packets, the tradeoff is the potential to miss sensor activity in nodes where profiling is not activated.

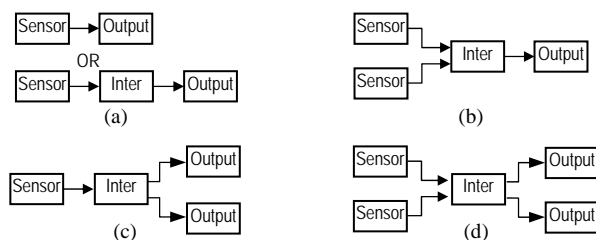
The piggybacking methodology operates as follows. The profiler sends a start command to the sensor nodes, sensor nodes update an internal register indicating that activity should be tracked. Only when the sensor node detects new activity is the augmented data packet transmitted. Control packets, as defined by the eBlock protocol, are not of interest to the profiler module and are not flagged. Packets marked as profiling packets traverse the network, the data contained within the packet having the same effect on intermediate and output nodes. The main difference being the detection of a profiling packet verses an unmarked packet by an intermediate node is that the outgoing packet also sets the profiling bit. Output nodes that detect profiling packets forward the packet to the profiler module.

Tracking the number of profiling packets detected by the profiler module will indicated the number of events detected by sensor nodes. However, to minimize the overhead of profiling, no timing data is contained within these packets to indicate the relative frequency of events. Instead, the profiler maintains a timer to track the relative frequency at which profiling packets arrive. The profiler module resets the internal timer when profiling is initiated. Detection of the first profiling packet starts the timer. The arrival time of subsequent profiling packets are denoted and stored in a memory. While the actual time of event detection is not maintained, the relative frequency of events contains sufficient information to determine lifetime.

### 2.3.2 ID-Based Profiling Methodology

To address the problem of tracking activity in networks with multiple sensors nodes, the ID-based profile methodology is proposed. Instead of piggybacking profiling packets unto existing data packets, this methodology injects new profiling packets. The packet is again generated only when a sensor node

Figure 2: eBlock topologies considered (a) single sensor node to single output node, (b) multiple sensor nodes to single output node, (c) single sensor nodes to multiple output nodes, and (d) multiple sensor nodes to multiple output nodes, with each scenario possibly containing multiple intermediate nodes.



detects new activity, containing the activated sensor's ID. The profiling packet does not contain data about the event detected. Rather, this profiling packet is generated in addition to the data packet normally transmitted by the sensor node to maintain network functionality. The advantage of the ID-based methodology is the ability to capture all sensor activity and differentiate between sensor nodes. The drawback of this methodology is that additional overhead is incurred by the introduction of a new packet.

Profiling via the ID-based methodology operates as follows. The profiler sends a start command to all sensor nodes. When a sensor node detects new activity a profiling packet and a data packet is transmitted. The profiling packet is propagated through the network, manipulation of profiling packet is not required as it does not contain data. Output nodes that detect profiling packets forward the packets to the profiler module.

Tracking sensor activity is similar to the method outlined in Section 2.3.1. The profiler maintains a timer to track the relative frequency at which profiling packets arrive. However, in this methodology a list of arrival times are correlated to sensor IDs.

### 3. Experiments

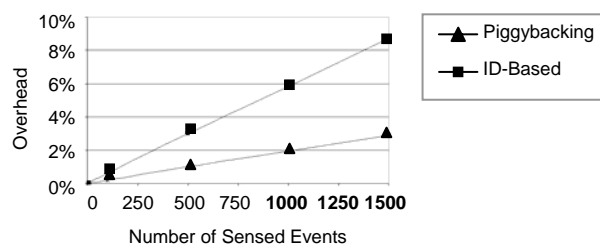
To determine the effectiveness of the piggybacking and ID-based approaches, each protocol was implemented on physical prototypes. Figure 2 illustrates the various system topologies considered. The specific type of sensor and output nodes varies, as well as the type and number of intermediate nodes.

The average overhead per node incurred by the piggybacking and ID-based methodologies is shown in Figure 3. Overhead is determined by comparing the total number of packets transmitted within the original network to the total number of packet transmitted within the augmented network tracking sensor activity. A one hour window is considered, in which the profiler module is activated during the entire time. The number of events detected during the 1 hour profiling period varies from 5 to 1500 events. The piggybacking methodology scales well with the increased number of sensed events, averaging overheads from 0.01% to 2.9%. The ID-based methodology is impacted more by the increased in sensed events, with overheads ranging from 0.03% to 8.7%.

### 4. Conclusions and Future Work

Environmental stimuli greatly impact the operation of sensor-based applications. To alleviate the difficulty of predicting application behavior at design time, we propose to gather sensor activity during the normal execution of a deployed application. We have proposed several dynamic profiling methodologies that

Figure 3: Profiling overhead incurred for piggybacking and ID-based profiling methodologies.



non-intrusively monitor the environmental stimuli while introducing minimal overhead.

Dynamic profiling reduces designer effort and enables numerous opportunities such as dynamic optimization and network reconfiguration based on current environmental activity. Much future work remains, including targeting profiling of additional metrics such as throughput or reliability, investigating the impact of the proposed methodology on various sensor platforms, and taking advantage of the dynamic profiling environment to consider dynamic optimizations. Additionally, the profiling itself can be a parameter within the design space, choosing among various techniques or tuning the frequency and duration of profiling to meet user specified platform restrictions.

### 5. References

- [1] Cotterell, S., et. al. First Results with eBlocks: Embedded Systems Building Blocks. CODES+ISSS, 2003.
- [2] Ganesan, D., et. al. Large-scale Network Discovery: Design Tradeoffs in Wireless Sensor Systems. SOSP, 2001.
- [3] Girod, L., et. al. EmStar: a Software Environment for Developing and Deploying Wireless Sensor Networks. USENIX Technical Conference, 2004.
- [4] Hill, J., D. Culler. MICA: A Wireless Platform For Deeply Embedded Networks. IEEE Micro, Vol. 22. No. 6, 2002.
- [5] Kadayif, I., M. Kandemir. Tuning In-Sensor Data Filtering to Reduce Energy Consumption in Wireless Sensor Networks. DATE, 2004.
- [6] Levis, P., N. Lee, M. Welsh, D. Culler. TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications. SenSys, 2003.
- [7] Lo, B., S. Thiernjarus, R. King, G. Yang. Body Sensor Network – A Wireless Sensor Platform for Pervasive Healthcare Monitoring. Pervasive, 2005.
- [8] Lysecky, S., F. Vahid. Automated Application-Specific Tuning of Parameterized Sensor-Based Embedded System Building Blocks. UbiComp, 2006.
- [9] Mainwaring, A., J. Polastre, R. Szewczyk, D. Culler, J. Anderson. Wireless sensor networks for habitat monitoring. WSN, 2002.
- [10] Park, C., P. Chou. EmPro: an Environment/Energy Emulation and Profiling Platform for Wireless Sensor Networks. SECON, 2006.
- [11] Perrone, F., L., D. Nicol. A Scalable Simulator for TinyOS Applications. Winter Simulation Conference, 2002.
- [12] Polley, J., et. al. ATEMU: A Fine-grained Sensor Network Simulator. SECON, 2004.
- [13] Shih, E., B. Calhoun, S. Cho, A. Chandrakasan. Energy-Efficient Link-Layer for Wireless Microsensor Networks. WVLSS, 2001.
- [14] Sundresh, S., W. Kim, Gul. Agha. SENS: A Sensor, Environment and Network Simulator. Simulation Symposium, 2004.
- [15] Tilak, S., N. Abu-Ghazaleh, W. Heinzelman. Infrastructure Tradeoffs for Sensor Networks. WSN, 2002.
- [16] Yu, Y., D. Ganesan, L. Girod, D. Estrin, R. Govindan. Synthetic data generation to support irregular sampling in Sensor Networks. International Conference on Geosensor Networks (GSN'06), 2003.