Evaluation of Dynamic Profiling Methodologies for Optimization of Sensor Networks

Ashish Shenoy, Jeff Hiner, Susan Lysecky, *Member, IEEE*, Roman Lysecky, *Member, IEEE*, and Ann Gordon-Ross, *Member, IEEE*

***Abstract*—To reduce the complexity associated with application-specific tuning of sensor-based systems, dynamic profiling enables an accurate view of the application behavior, such that the network can be reoptimized at runtime in response to changing application behavior or environmental conditions. However, dynamic profiling must be able to accurately capture application behavior without incurring significant runtime overheads. We present several profiling methods for dynamically monitoring sensor-based platforms and analyze the associated network traffic, energy, and code impacts.**

***Index Terms*—Application-specific optimization, design automation, dynamic profiling, sensor networks.**

I. Introduction

NUMEROUS sensor-based platforms have appeared enabling a wide range of application possibilities. With each application scenario, developers have a unique set of application requirements such as lifetime, responsiveness, reliability, or throughput that must be met. For example, a disaster response application requires high responsiveness and reliability to survey damage or detect survivors, but may only require a lifetime of days or weeks. Conversely, an automated vineyard irrigation system would have a longer lifetime requirement as it is intended to operate on the order of years.

To achieve various application goals, developers can tune configurable node-level parameters such as voltage levels, processor mode, or configurable baud rates [4]. Furthermore, developers can also consider numerous protocol-level design choices such as power cycling to sensing units [3], or data aggregation and filtering [5]. While the effects of various parameter configurations have on high-level design metrics have been well documented, balancing these numerous competing metrics remains challenging. To further complicate matters, predicting application behavior is extremely difficult at design time. Tuning the underlying platform to inaccurate application behavior estimates can yield either suboptimal results or negatively impact the resulting system. Currently, application developers are left to specify application-behavior via an input file [11], a mathematical model [10], or through synthetic data generation [12]. While a few real-time tools have appeared [9], they are not designed for dynamic profiling and incur significant overhead.

To alleviate some of the complexity associated with application-specific tuning of sensor-based systems, we have begun to develop a dynamic profiling and optimization (DPOP) framework. Dynamic profiling and optimization not only reduces designer effort, but an automated environment increases accessibility to nonexpert application developers by abstracting much of the underlying platform specific knowledge. Dynamic profiling enables an accurate view of the application behavior while immersed in its intended environment, eliminating the guesswork of trying to create a "good" benchmark. Furthermore, by profiling applications dynamically, we can monitor how the application responds to changes in environmental conditions or changes in the underlying platform (i.e., a node is disabled), opening opportunities to dynamically reoptimize and update the underlying platform accordingly.

However, such dynamic optimization relies upon accurate profiling results collected at runtime. Currently, an accurate and robust method to capture external application-specific stimuli remains elusive. While many dynamic profiling techniques exist, these techniques are highly specific to their intended system and thus, are quite low level. For example, working set analysis [1] monitors the current set of executing instructions to determine changes in system execution. Kaxiras *et al.* [6] determines changes in cache requirements using counters embedded within the cache structure, while other methods simply observe current idle periods [2]. Whereas idle period observation is a generalized, high-level mechanism to profile a system, this method, when applied to sensor networks, provides little information on overall system behavior. Furthermore, the distributed nature of sensor-based networks complicate adoption of previously developed profiling methods to sensor-based networks.

One of the major challenges of dynamically profiling sensor-based platforms is accurately capturing application behavior without incurring significant overhead or significantly altering system behavior. In this letter, we present several profiling methods for dynamically monitoring sensor-based platforms and analyze the traffic/energy/code impacts for two prototyped sensor-based systems.

II. DPOP Environment

Fig. 1 illustrates the proposed dynamic profiling and optimization platform. Three main components contribute to the

Manuscript received January 22, 2010; revised March 06, 2010: accepted March 06, 2010. Date of publication March 15, 2010. Date of current version April 26, 2010. This work was supported in part by the National Science Foundation, under Grants CNS-0834102 and CNS-0834080.

A. Shenoy, J. Hiner, S. Lysecky, and R. Lysecky are with the Department of Electrical and Computer Engineering, University of Arizona, Tucson, AZ 85721 USA (e-mail: ashenoy@email.arizona.edu; jhiner@email.arizona.edu; slysecky@ece.arizona.edu; rlysecky@ece.arizona.edu).

A. Gordon-Ross is with the Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL 32611 USA (e-mail: ann@ece.ufl.edu).

Digital Object Identifier 10.1109/LES.2010.2045634

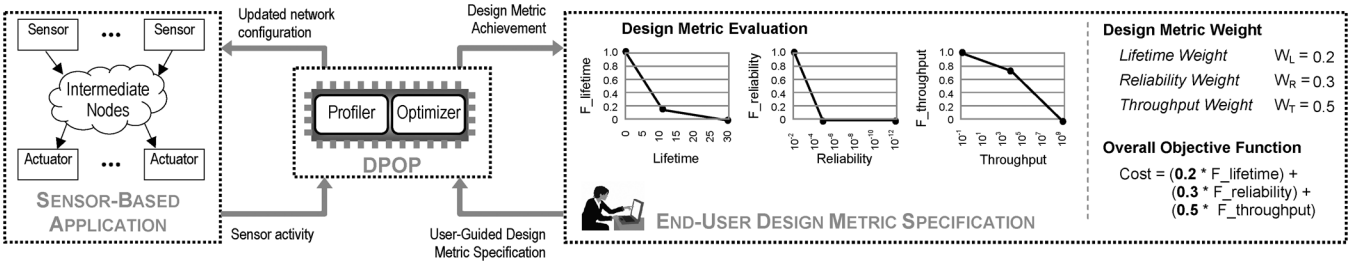

Fig. 1. DPOP environment highlighting various tasks including specifying design metric evaluation equations and assigning design metric weights to indicate the relative importance between each design metric.

proposed environment and include the sensor-based application, the end-user design metric specification, and the DPOP module.

The *sensor-based application* is the physical deployment of the application within the intended environment and consists of sensor nodes, intermediate processing and routing nodes, and actuator nodes, working together to achieve the desired application functionality.

Ultimately, the application developer is interested in high-level system metrics such as the expected lifetime of a node or sensor network utilizing two AA batteries, the time required to process a single packet, or the time required to process and respond to a sensor event. The *end-user design metric specification* allows an application developer to define which design metrics are of importance to a particular application, and of those design metrics, what are the acceptable or unacceptable values of each, thereby providing a method to interpret the resulting system achievement within the context of a given application.

First, for each design metric, an application developer correlates a design cost to the raw metric value (i.e., lifetime of two months), where a lower design cost corresponds to a superior design metric value. Fig. 1 illustrates three design metric objective functions corresponding to lifetime, reliability, and throughput, for which a graphical interface [7] is utilized to specify the design metric objective functions as piecewise linear functions.

Second, to determine the relative importance of each design metric, the application developer additionally specifies weights for each design metric, as shown in Fig. 1. The overall objective function—or overall design cost—combines the individual design metric weights as well as the resulting costs assigned by each design metric objective function. This overall design cost indicates how well an individual node configuration meets the specified application requirements.

The *DPOP* node is a separate component—implemented either within the base station node or as a separate sensor node—dedicated to the *profiling* and *optimization* of the underlying sensor-based platform, as the platform interacts within the intended environment. The *profiler module* dynamically monitors the application behavior while the sensor-based system is deployed, tracking statistics of interest to the application developer. The *optimizer module* evaluates possible node configurations within the design space to determine which configuration best meets the application requirements. Given the design metric evaluation specification and dynamic profile data, the optimizer first utilizes an equation based estimation methodology that estimates each design metric using both

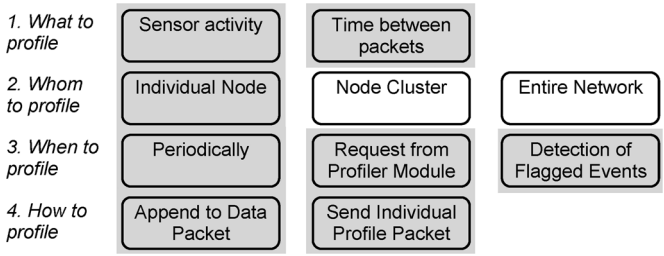

Fig. 2. Overview of profiling strategies considerations including *what*, *whom*, *when*, and *how* to profile (*shaded options are currently supported by our profiling implementation*).

the node configuration and profile data. The optimizer then explores the design space by evaluating each feasible node configuration to determine which node configuration is best suited for a given application, i.e., the configuration yielding the lowest overall design cost. Dynamic optimization of sensor nodes using the DPOP environment can yield up to an 83% improvement in overall design costs compared to a statically optimized node configuration.

III. Dynamic Profiling of Application Behavior

Within the DPOP environment, dynamically profiling a sensor-based application requires profiling methods to be incorporated within each node to monitor the execution behavior for individual sensor nodes. Additionally, in order to optimize a sensor-based system, a global view of the entire system is needed. As such, the resulting node-level profile data must eventually be transmitted and analyzed by the system-level *profiler module*. Numerous profiling strategies can be employed to collect the pertinent application level information. As highlighted in Fig. 2, each profiling strategy must consider: 1) what application level parameters needed to be profiled; 2) whom to profile within the network; 3) when to perform profiling; and 4) how to transmit profile information from the individual sensor nodes.

One of the foremost concerns for a profiling strategy is to determine what low-level execution statistics (e.g., sensor sampling rate, packet transmissions, battery charge) need to be profiled within the deployed network. At each sensor node, various low level execution details must be monitored in order to enable the optimization approach to accurately estimate the various high-level design metrics of interest. For example, consider a sensor node that periodically samples and reports sensor data. The power consumption of the software required to process each

sensor event and transmit the packet can be statically measured through physical measurements. The runtime power consumption of the software executing on a node can be estimated as a function of the measured power consumption and the dynamic sensor sampling rate. Overall, determining what low-level metrics to profile within a sensor-based platform is thus related to both the high-level design metrics of interest and the estimation method utilized to evaluate those design metrics. Within our current profiling implementation, both the *sensor sampling rate* and the *time between successive packets* can be profiled for individual sensor nodes.

Profiling individual sensor nodes is not always necessary. Hence, a profiling strategy must consider the granularity at which profiling is performed, including profiling an individual node, a cluster of nodes, or profiling the network as a whole. The granularity at which to profile is affected by both application and network topology. For example, the profiler may want to profile only those intermediate nodes whose job is to forward packets as these nodes may have higher energy consumption for which optimizing lifetime is of critical importance. Alternatively, the profiler may consider a single sensor node placed in a known area with high activity to determine the minimum sampling rate to needed by the application. Profiling the entire network is also possible in which nodes directly aggregate—or average—the collected profile information as it is forwarded profiler module. However, profiling at different levels of granularity provides the ability to tradeoff profiling detail with profiling overhead. Although various profiling granularities are possible, our current profiling implementation only supports profiling of *individual nodes*.

Given the desired profiling information to be collected, the frequency at which profiling is performed directly impacts both the accuracy of the profile data, as well as the intrusiveness of the profiling method. On the one hand, profiling can be performed periodically at each node or cluster of nodes. Although the performance and energy overhead of periodic profiling is often easily predicted, dynamic activity patterns of individual nodes or across the sensor network may be unpredictable and a periodically collected profile may not accurately describe the current execution behavior. On the other hand, nodes can directly detect all events related to the required profiling information and directly transmit those event occurrences to the profile module as they occur. Such an approach provides the advantage of highly accurate profile information, but at the expense of potential increases in both code size needed to detect such events and increased packet transmission overhead due to unpredictable event occurrences. An alternative approach to control when to profile is requiring the profiler module to explicitly send a profile request packet. While a packet transmission overhead is incurred to transmit the profile request packet, the profiler module can dynamically control how often these requests are sent based on the data collected thus far or patterns previously observed. Our current profiling implementation provides support for all three methods of controlling when profiling is performed for individual nodes, specifically *periodic*, *event-driven*, and *profiler module directed*.

Finally, the method of transmitting the collected profile data back to the profiler module directly impacts both the network traffic of the sensor network, as well as node's energy consumption as the radio subsystem must remain active for longer durations to transmit the profile data. Currently, our profiler implementation provides support for either transmitting profile data as *separate profile packets* or appending, i.e., *piggybacking*, the profile data to existing packets already transmitted by the application. Requesting nodes to send separate profiling packets may increase overall network traffic as each dedicated profile packet must also include the packet header. Instead, by piggybacking the profile information onto existing data packets, the profile data can be transmitted without requiring an additional packet header. However, piggybacking profile data onto existing data packets may require individual sensor nodes to store the profile data until the sensor nodes transmit a data packet.

To evaluate the feasibility of the proposed profiling methodologies within the DPOP framework, we implemented the above mentioned profiling methods on the Crossbow IRIS platform as a set of software functions that can be readily integrated within a sensor application. While incorporating these profiling methods into the target application currently requires designer effort, the required changes do not directly impact the main application functionality. Rather, these methods are inserted within the underlying software infrastructure for packet transmission/reception and sensor interfaces. As many application developers will not need to directly modify this code, various profiling methods can be selected by simply including the required set of software driver source files for the target application.

The resulting profiling methodologies can supported all combinations of what, whom, when, and how to profile mentioned above. However, we currently consider the following four specific profiling methods.

- PM1: sensor sampling rate and time between successive packets; individual nodes; profiler module directed; piggybacked.
- PM2: sensor sampling rate and time between successive packets; individual nodes; profiler module directed; separate profile packets.
- PM3: sensor sampling rate and time between successive packets; individual nodes; periodic; separate profile packets.
- PM4: sensor sampling rate; individual nodes; profiler module directed; separate profile packets.

IV. Experiments

We consider two sensor-based applications to evaluate the overheads of the four proposed profiling methods. The first application is a Forest Fire Detection and Propagation Tracking system. During the normal observation mode, individual sensors sample and transmit the surrounding temperate reading every five minutes to a base station. In the event that a node detects an elevated temperature, beyond a user-defined threshold, an alert to nearby nodes is issued to transition to a fire tracking mode and report the temperature every ten seconds. We have also developed a Building Monitor application. Within this application, sensor nodes synchronize hourly with the base station to verify the node is still functioning, as well as obtain which operation mode the node should be in. In the low-power mode, nodes do

TABLE I
NETWORK TRAFFIC (%/BYTES), ENERGY CONSUMPTION (mAH), AND CODE SIZE (%/kb) OVERHEADS OF VARIOUS PROFILING STRATEGIES FOR THE FOREST FIRE DETECTION AND BULIDING MONITOR APPLICATIONS

Application	Profiling Method PM1			Profiling Method PM2			Profiling Method PM3			Profiling Method PM4		
	Traffic	*Energy*	*Code*	*Traffic*	*Energy*	*Code*	*Traffic*	*Energy*	*Code*	*Traffic*	*Energy*	*Code*
Forest Fire	8.7%/35	0.01	1.9%/0.8	14.8%/98	0.06	2.2%/0.9	7.9%/54	0.02	2.4%/1.0	8.4%/50	< 0.01	1.7%/0.7
Building Monitor	11.6%/18	0.02	3.5%/1.4	32.2%/30	0.02	2.5%/1.0	17.8%/16	0.01	3.0%/1.2	16.5%/17	< 0.01	2.2%/0.9

not need to detect movement and return to a sleep state until the next synchronization. During the monitor mode, if the standard deviation of the last four samples is greater than a user defined threshold, the sensor node will transmit a message to the base station indicating movement with the corresponding time. In contrast to the forest fire monitoring application that is *periodic* in nature, the building monitor is a more *reactive* system in that it reports movement when detected.

We implemented all applications without profiling and with each of the four profiling methods to determine the network traffic, energy, and code size overheads, presented in Table I.

For the forest fire detection and propagation tracking application, PM3 yields the lowest network traffic overhead. Although the profile information is sent as individual packets within the PM3 strategy, by eliminating the need to transmit profile request packets from the base station, the overhead is reduced compared to both PM1 and PM2. However, for the building monitor application, the PM1 strategy incurs the lowest overhead. For this application, the profiling data is primarily piggybacked within the time synchronization packets. Due to the low overall number of packets transmitted within the network for this application, piggybacking significantly reduces the additional traffic that would be required for the other methodologies. This implies that profiler module directed profiling is well suited to reactive systems due to the unpredictable nature of these applications. In addition, piggybacking profile data to existing packets is only preferable when periodic profiling is employed, as transmitting both profile request packets and separate profile packets leads to significant 14.8% and 32.2% overheads for the two respective applications.

Across all profiling methods, energy and code size overhead remain reasonable with a maximum overhead of only 0.06 milliamp-hours and 1.4 kilobytes (or 3.5%).

We note that the profiling overheads are dependent on the both application behavior and profiling methods. As such, it is necessary to be able to either characterize an application at design time to select an appropriate profiling method or dynamically select between profiling methods at runtime, although we currently leave this as future work.

V. CONCLUSION

Dynamic profiling of sensor-based platforms enables an accurate view of an application's execution behavior, but at the expense of network traffic, energy, and code size overheads. We have developed various methods for controlling the profiling process and analyzed the corresponding overhead for four specific profiling methods. While the energy and code size increases are reasonable, network traffic overheads range from 7.9% to 32.2%. Furthermore, choosing an appropriate profiling mechanism is dependent on the application behavior itself and a single profiling method is unlikely to provide good results across different classes of applications.

REFERENCES

[1] A. Dhodapkar and J. Smith, "Managing multi-configuration hardware via dynamic working set analysis," in *Proc. Int. Symp. Comput. Arch.*, Anchorage, AK, 2002, pp. 233–244.
[2] F. Douglis, P. Krishnan, and B. Bershad, "Adaptive disk spindown policies for mobile computers," in *Proc. Symp. Mobile Location-Independent Comput.*, Ann Arbor, MI, 1995, pp. 121–137.
[3] P. Dutta and D. Culler, "System software techniques for low-power operation in wireless sensor networks," in *Proc. Int. Conf. Comput.-Aided Design*, San Jose, CA, 2005, pp. 925–932.
[4] J. Hill and D. Culler, "MICA: A wireless platform for deeply embedded networks," *IEEE Micro*, vol. 22, no. 6, pp. 12–24, Nov. 2002.
[5] I. Kadayif and M. Kandemir, "Tuning in-sensor data filtering to reduce energy consumption in wireless sensor networks," in *Proc Design, Autom., Test Eur. Conf. (DATE)*, Paris, France, 2004, pp. 1530–1591.
[6] S. Kaxiras, Z. Hu, and M. Martonosi, "Cache decay: Exploiting generational behavior to reduce cache leakage power," in *Proc. Int. Symp. Comput. Arch.*, Goteborg, Sweden, 2001, pp. 240–251.
[7] S. Lysecky and F. Vahid, "Automated application-specific tuning of parameterized sensor-based embedded system building blocks," in *Proc. UbiComp*, Orange County, CA, 2006.
[8] A. Munir and A. Gordon-Ross, "An MDP-based application oriented optimal policy for wireless sensor networks," in *Proc. Conf. Hardware/Software Codesign Syst. Synth.*, Grenoble, France, 2009, pp. 183–192.
[9] C. Park and P. Chou, "EmPro: An environment/energy emulation and profiling platform for wireless sensor networks," in *Proc. Conf. Sensor Ad Hoc Commun. Netw.*, 2006, pp. 158–167.
[10] F. Perrone and D. Nicol, "A scalable simulator for TinyOS applications," in *Proc. Winter Simulation Conf.*, San Diego, CA, 2002, pp. 679–687.
[11] J. Polley, D. Blazakis, J. McGee, D. Rusk, J. Baras, and M. Karir, "ATEMU: A fine-grained sensor network simulator," in *Proc. Conf. Sensor Ad Hoc Commun. Netw.*, 2004, pp. 145–152.
[12] Y. Yu, D. Ganseen, L. Girod, D. Estrin, and R. Govindan, "Synthetic data generation to support irregular sampling in sensor networks," *GeoSensor Netw.*, vol. 1, no. 4, pp. 211–234, 2004.