A Framework for Programming Sensor Networks with Scheduling and Resource-Sharing Optimizations

(Invited Paper)

Vikram Gupta[†‡], Eduardo Tovar[†], Karthik Lakshmanan[‡], Ragunathan (Raj) Rajkumar[‡]
[†]*CISTER Research Center, ISEP, Polytechnic Institute of Porto, Portugal*
[‡]*Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, USA*
vikramg@ece.cmu.edu, emt@isep.ipp.pt, {klakshma, raj}@ece.cmu.edu

***Abstract*—Several projects in the recent past have aimed at promoting Wireless Sensor Networks as an infrastructure technology, where several independent users can submit applications that execute concurrently across the network. Concurrent multiple applications cause significant energy-usage overhead on sensor nodes, that cannot be eliminated by traditional schemes optimized for single-application scenarios. In this paper, we outline two main optimization techniques for reducing power consumption across applications. First, we describe a compiler based approach that identifies redundant sensing requests across applications and eliminates those. Second, we cluster the radio transmissions together by concatenating packets from independent applications based on Rate-Harmonized Scheduling.**

***Keywords*-Wireless Sensor Networks; Scheduling; Optimization; Programming**

I. Introduction

Wireless Sensor Networks are increasingly gaining popularity in many research communities, and various deployments have been put in place for developing varied technologies (such as [1] [2], [3]). Most of these deployments are, however, only limited to be used by computer-scientists or the network administrators. In order to allow general acceptance of sensor networking as an infrastructure technology, suitable programming support should be provided so that non-experts can write simple applications easily and collect data from a sensor network. Hence, we identify two main requirements for enabling sensor networks as an infrastructure:

- **R1:** The users should be able to write sensor networking applications through in-network programming, and
- **R2:** The network should provide upport for multiple concurrent network-level applications.

Several research projects in the past (e.g. [4] [5] [6] [7]) have focussed on developing support for simultaneous applications. These works, however, do not focus on optimizing the power consumption overhead resulting from independent applications on a sensor network.

Sensor Nodes are typically resource-constrained both in battery energy and computation power; therefore, it is important to optimize their consumption not only for individual applications but across applications as well. Application developers and operating system designers have long focused on saving energy on a sensor node, but multiple applications on a sensor node add extra overhead that may not be eliminated with traditional optimization approaches. Concurrent applications may force frequent turning On/Off of the processor and the radio leading to more power consumption. In addition to the radio power consumption, frequent sampling of the onboard sensors by independent applications can introduce redundant workload. Sensor networking applications are typically designed with very low duty-cycles, and On periods of radio and processor may not align across multiple applications. This adds non-proportional overhead because of frequent radio initialization and switching of the processor from sleep to active states.

As applications are assumed to be submitted independently by users to a sensor networking infrastructure, it may not be possible for an application-level optimization to reduce the global resource usage. Hence, a programming framework is required that incorporates inter-application resource optimizations. In our previous work [8], we proposed a framework to support multiple concurrent applications with radio-packet transmission scheduling based on Rate-Harmonized Scheduling (RHS) [9]. Such a framework has a global view of the network allowing optimization across all the applications. In such a framework, the users create applications using a higher-level or a *macro-programming* language, the programs are then compiled to executable byte-code that is delivered to the nodes in the network. We present a two-fold approach where we reduce redundancy across application using a compile-time approach and apply Rate-Harmonized Scheduling (RHS) for aligning packet transmissions from different *tasks*[1].

We describe the design and architecture of our framework and approaches for optimizing the resource usage on a sensor node. The paper is organized as follows: in the next section we provide the motivation and scope of energy optimizations in case of multiple applications. In Section III

[1]In this paper, we refer to the user-created network-level programs as *applications*, and the jobs that execute on individual sensor nodes as *tasks*. In other words, *tasks* are node-level executables based on the *applications*.

we provide details of the framework along with the design of the compiler for eliminating redundancies across applications. We then describe our approach of radio-transmission clustering using Rate-Harmonized Scheduling in Section IV. In the concluding section, we provide the Discussion and Future Work.

II. Motivation

Multiple applications cause frequent triggering on/off of processor and radio, and the overhead associated with the start-up of various components can add to non-proportional consumption on energy. As an example, Table I shows various timing parameters for CC2420 radio transceiver chip [10] associated with the initialization of its crystal oscillator, phase-locked loop, and voltage regulator. It can be seen from the table that each time the radio chip shifts from a power down mode to active mode, up to 33% extra time-overhead is added with respect to the time required for transmission of one packet. A packet of 128 Bytes takes 4 ms for transmission using a 250 kbps radio.

Table I
TIMING OVERHEAD PARAMETERS FOR CC2420 RADIO CHIP, COMMONLY USED IN SENSOR NODES. THE PERCENTAGE OF EACH TIMING PARAMETER WITH RESPECT TO RADIO USAGE FOR A PACKET TRANSMISSION IS ALSO PROVIDED.

Parameter	Time	Percentage
Crystal Oscillator Start-up Time	0.86 ms	21.5%
PLL Lock Time	192 μs	4%
Voltage Regulator Start-up Time	0.3 ms	7.5%

Similarly, sampling a sensor involves using the Analog-to-Digital Converter (ADC), and such an instruction can take about 2-3 orders of magnitude more time than a simple processor instruction. A sampling instruction is of the form:

```
temp_val = sample_sensor(TEMP);
```

The variable assignment instruction is similar to a copy instruction like the following:

```
temp_val_copy = temp_val;
```

A comparison of the time taken by a sampling instruction versus a variable assignment instruction is shown in Figure 1. This comparison is obtained by toggling a GPIO pin just before and after the execution of a sensor sampling instruction (shown by the Trace 1) and copying of a 16 bit value into a register (Trace 2). The former takes about 500 microseconds but the latter instruction takes only 10 microseconds. Please note that this time comparison also includes the time taken for toggling the I/O pins. As the Atmel ATMEGA1281 ($8MHz$) processor typically used on the sensor node has on-chip memory, a load instruction takes a maximum of 3 cycles that corresponds to 375 nanoseconds. A majority of the time consumed in the case of Trace 2

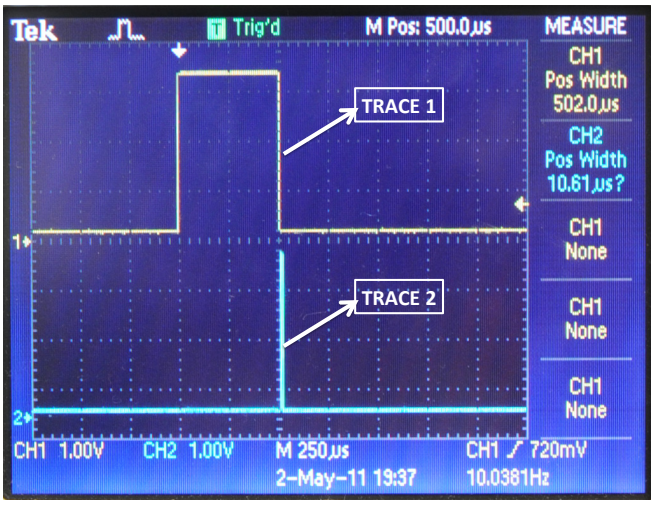


Figure 1. Oscilloscope screenshot showing two traces. Traces 1 (Yellow trace) shows the time taken to acquire one sample reading of the light sensor on the Firefly sensor platform running Nano-RK, and Trace 2 (blue) shows the time taken for executing a simple variable asignment instruction.

is because of the pin toggling. Hence, a sensor sampling instruction consumes up to $\frac{(500-10)\times 10^{-6}}{375\times 10^{-9}} = 1306$ times more power. This factor, which we refer to as *time-factor*, is specific to the platform and the operating system. However, the order of magnitude of the *time-factor* can be assumed to be similar over most of the common sensing systems.

III. Resource Optimization at Compile-Time

We assume that the users develop network-level sensing applications using a programming framework such as Nano-CF [8]. The application code written by the users can either be abstract network-level using a macro-programming language or node-specific virtual-machines (for example Matè [11]). In both the cases, the underlying framework creates node-level intermediate code based on the application logic specified by the user. In this paper, we describe our approach based on a machine-language like intermediate code, generally referred to as *bytecode*. The architecture of such a complete system is shown in Figure 2, where the user applications are converted into bytecode by a parser, such that each output instruction is either an indivisible subexpression or a special function for accessing the hardware (including sensing, GPIO access or packet transmission). Bytecode corresponding to each application is converted to a monolithic code by the *Redundancy Eliminator with Implicit Scheduler (REIS)* module. This monolithic code, which we call *REIS-bytecode* and ρ-code in short, is a merged sequence of all the applications but the redundancies are eliminated according to temporal overlap of sensing requests. REIS-bytecode is then sent over the wireless network to each sensor node where the applications are to be executed. A bytecode interpreter at the sensor node executes

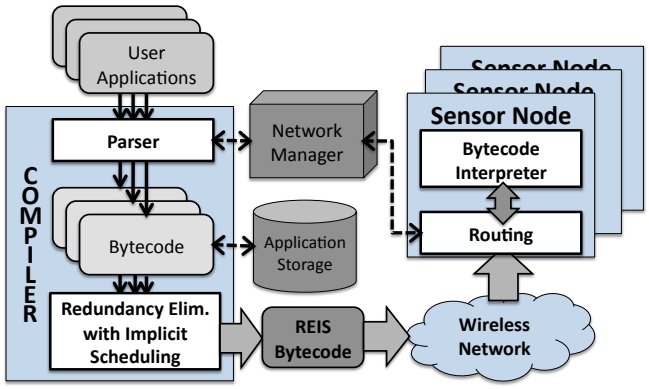
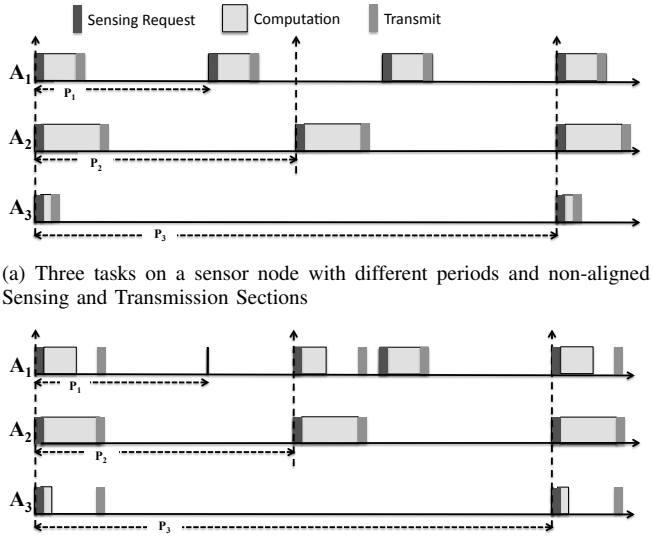


Figure 2. Overview of the approach for redundancy elimination among independent applications along with compiler-assisted scheduling.



(a) Three tasks on a sensor node with different periods and non-aligned Sensing and Transmission Sections

(b) Sensing and Transmission aligned across tasks

Figure 3. Aligning Sensing and Transmission requests from different tasks on a sensor node

the received REIS-bytecode. Further details of this compiler-assisted approach is provided in [12].

The approach assumes that a data link-layer and a suitable routing layer is already implemented on the sensor node and our solution is transparent to it as long as end-to-end packet delivery is supported. A network manager module handles the responsibility of dynamically updating the routing tables, and maintaining network topology information. As users issue applications to the system independently, our approach requires an application storage database to store application bytecode and merge them using the REIS module whenever a new application is submitted. The logic of the user applications is interleaved inside the REIS-bytecode to provide maximum sharing of sensing requests and radio transmissions. Bytecode from different applications share non-overlapping variable and address space, which removes any need for context switching between applications, and the interleaving of bytecode provides an implicit schedule of execution.

IV. Rate-Harmonized Scheduling

Each application deployed on the network has an associated period of execution that is specified by the user. Several tasks on sensor node with mismatching periods can create several shorter intervals of sleep duration instead of a longer one even in case where each task a small duty-cycle. Rate-Harmonized Scheduling clusters periodic tasks such that all task executions are grouped together in time to accumulate idle time durations in the processor schedule. This accumulation helps processor to go into the deep sleep state. This property can also be applied to packet transmissions, and sending bigger concatenated packets consumes less energy than sending multiple smaller packets more frequently. An example scenario where sensing requests and packet transmissions can be aligned across periodic tasks is shown in Figure 3.

The framework supports delaying the packet transmission and hence combining the packets together, which yeilds significant power savings by using the radio transceiver for shorter durations at low duty cycles. This effect is shown in Figure 4. The plot in the figure is obtained by estimating relative power savings for randomly generated packets with varying maximum packet size from 1 byte to 100 bytes. Every data-point shows average after 50 iterations. When 3 applications are used, the energy consumption related to packet delivery can be saved up to 35%. In addition, if we use 5 applications, the amount of energy saving is increased up to 50%. As the maximum packet size increases, the effect of saving energy is decreasing. It happens because large packets may not be merged anymore. In addition, we can obtain opportunities to save more energy due to high possibilty of clustering packets from more number of applications. Aggregating packets together helps in reducing the number of packets transmitted in the network, which in turn reduces the channel contention and packet loss due to collision.

V. Discussion and Future Work

We presented a brief description of our framework for programming sensor networks that employs packet scheduling optimizations and redundancy elimination at the compiler level to save energy at each sensor node. These optimizations are enabled by the network-level programming support provided to the users that not only abstracts away the low-level complexities of application development but also provides features for significant energy savings. The energy consumed at each node, and the possible savings in those are dependent on type of applications, and can significantly vary based on the individuals workload of sensing, computation and data-transmission. As radio is the single most power-

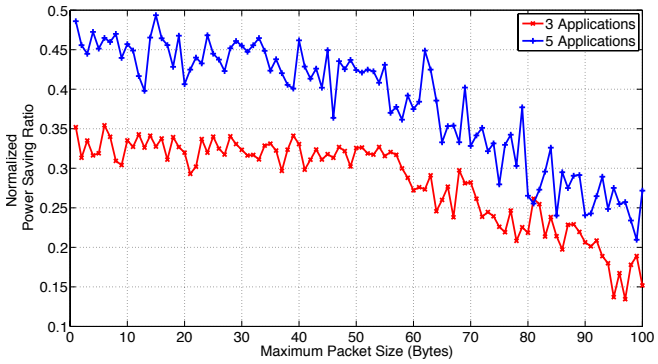


Figure 4. Applying RHS in packet delivery allows each sensor node save the energy by reducing the number of packets to be sent.

expensive component on a sensor node, and sampling a sensor can take 2-3 orders of magnitude more processor time than a simple instruction execution, optimization of a few transmission and sensing requests can achieve very significant power savings. Please note that we do not take into account the power usage because of the communication and computation for maintaining the network-routing and other basic technologies, as these are out of the scope of this work.

As future work, we would like to design further optimizations for a sensor network operating system. Traditional in-network programming approaches are designed as a middleware on top of the sensor node operating system, which generally acts as a virtual-machine interpreter. There is a significant scope of improvement in both the memory and computation resource usage by designing the operating system from the network-level programming point-of-view instead of other way around. As network-level programming is important for sensor networks to gain popularity as an infrastructure technology, we wish to design a hypervisor for sensor networks that is optimized for supporting in-network programming rather than for application development at the node level. The overhead of an interpreter runnning on top of an operating system can be significantly high, we aim to merge the functionality of those together in a single firmware.

ACKNOWLEDGEMENTS

This research is partially funded by Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) through Information and Communication Technologies Institute (ICTI), under the Carnegie Mellon Portugal Program.

REFERENCES

[1] K. Langendoen, A. Baggio, and O. Visser, "Murphy loves potatoes: experiences from a pilot sensor network deployment in precision agriculture," in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, april 2006, p. 8 pp.

[2] G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh, "Fidelity and yield in a volcano monitoring sensor network," in *Proceedings of the 7th symposium on Operating systems design and implementation*, ser. OSDI '06. Seattle, Washington: USENIX Association, 2006, pp. 381–396. [Online]. Available: http://portal.acm.org/citation.cfm?id=1298455.1298491

[3] N. Ramanathan, T. Harmon, L. Balzano, D. Estrin, M. Hansen, J. Jay, W. Kaiser, and G. Sukhatme, "Designing wireless sensor networks as a shared resource for sustainable development," in *Information and Communication Technologies and Development*, 2006.

[4] Y. Yu, L. J. Rittle, V. Bhandari, and J. B. LeBrun, "Supporting concurrent applications in wireless sensor networks," in *Proceedings of the 4th international conference on Embedded networked sensor systems*, ser. SenSys '06. Boulder, Colorado, USA: ACM, 2006, pp. 139–152. [Online]. Available: http://doi.acm.org/10.1145/1182807.1182822

[5] S. Bhattacharya, A. Saifullah, C. Lu, and G.-C. Roman, "Multi-application deployment in shared sensor networks based on quality of monitoring," *Proceedings of Real-Time and Embedded Technology and Applications Symposium, IEEE*, pp. 259–268, 2010.

[6] J. Steffan, L. Fiege, M. Cilia, and A. Buchmann, "Towards multi-purpose wireless sensor networks," in *Systems Communications, 2005. Proceedings*, aug. 2005, pp. 336 – 341.

[7] A. Tavakoli, A. Kansal, and S. Nath, "On-line sensing task optimization for shared sensors," in *IPSN '10: Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*. Stockholm, Sweden: ACM, 2010, pp. 47–57.

[8] V. Gupta, J. Kim, A. Pandya, K. Lakshamanan, R. Rajkumar, and E. Tovar, "Nano-cf: A coordination framework for macro-programming in wireless sensor networks," in *(to appear) Proceedings of the 8th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON), 2011*, ser. SECON-VIII. IEEE, 2011.

[9] A. Rowe, K. Lakshmanan, H. Zhu, and R. Rajkumar, "Rate-harmonized scheduling for saving energy," in *RTSS '08: Proceedings of the 2008 Real-Time Systems Symposium*. Barcelona, Spain: IEEE Computer Society, 2008, pp. 113–122.

[10] "Chipcon inc., chipcon cc2420 data sheet," 2003.

[11] P. Levis and D. Culler, "Matè: a tiny virtual machine for sensor networks," in *Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, ser. ASPLOS-X. San Jose, California: ACM, 2002, pp. 85–95. [Online]. Available: http://doi.acm.org/10.1145/605397.605407

[12] V. Gupta, E. Tovar, K. Lakshmanan, and R. R. Rajkumar, "Inter-application redundancy elimination in sensor networks with compiler-assisted scheduling," *CISTER Tech Report*. [Online]. Available: http://www.cister.isep.ipp.pt/docs/