From Automatic to Adaptive Data Acquisition

- towards scientific sensornets

Chang, Marcus

Publication date:
2009

Document version
Også kaldet Forlagets PDF

DEPARTMENT OF COMPUTER SCIENCE
FACULTY OF SCIENCE
UNIVERSITY OF COPENHAGEN

PhD thesis

Marcus Chang

From Automatic to Adaptive Data Acquisition
- towards scientific sensornets

November 2009

Submission: November 2009

Typesetted with $\LaTeX\ 2_\epsilon$

Preface

The following dissertation concludes my studies at the University of Copenhagen. It is a synopsis of five papers in the field of sensornets and ecological monitoring. Three of these papers have been published and two are currently under review. The journey has been long, with detours to both physics and astronomy, but not as long as the list of people I wish to thank:

My advisor, Philippe Bonnet, for guiding me through both rounds of graduate school and sending me on cool trips around the World.

Special thanks to Prof. Andreas Terzis, for hosting my visits at the Johns Hopkins University, and all the great grad students at the Wymann Park lab for all the fun and great food.

Kirsten S. Christoffersen for two awesome field trips to Greenland; I hope the animals will leave our equipment alone this season.

Our departmental PhD Secretary, Marianne Henriksen, for administrative support and always getting the job done. Be well.

Friends and family for encouragement and support. Especially, Henrik Juul Nielsen for proofreading and Anders Siegumfeldt for sparring.

My first lab mates at the physics department, for great times and friendly competition.

Finally, Jens Martin Knudsen for inspiring hopes and dreams, and Bo Bune Rasmussen for motivation and drive.

Marcus Chang
November 2009

Summary

Sensornets have been used for ecological monitoring the past decade, yet the main driving force behind these deployments are still computer scientists. The denser sampling and added modalities offered by sensornets could drive these fields in new directions, but not until the domain scientists become familiar with sensornets and use them as any other instrument in their toolbox.

We explore three different directions in which sensornets can become easier to deploy, collect data of higher quality, and offer more flexibility, and we postulate that sensornets should be instruments for domain scientists.

As a tool to ease designing and deploying sensornets, we developed a methodology to characterize mote performance and predict the resource consumption for applications on different platforms, without actually having to execute them. This enables easy comparison of different platforms.

In order to reduce the amount of faulty and missing measurements, we developed a mote-based anomaly detection framework lightweight enough to run alongside an actual data acquisition application. This allows faulty measurements to be detected immediately and not after the experiment has been concluded.

To increase the flexibility of sensornets and reduce the complexity for the domain scientist, we developed an AI-based controller to act as a proxy between the scientist and sensornet. This controller is driven by the scientist's requirements to the collected data, and uses adaptive sampling in order to reach these goals.

Contents

1 Introduction **1**

2 Automatic Data Acquisition **11**
2.1 Lessons from the Hogthrob Deployments
Marcus Chang, Cécile Cornou, Klaus S. Madsen, and Philippe Bonnet
2nd Intl. Workshop on Wireless Sensor Network Deployments, 2008 . . 15

3 Normalizing Resource Consumption **21**
3.1 Characterizing Mote Performance: A Vector-Based Methodology
Martin Leopold, Marcus Chang, and Philippe Bonnet
5th European conference on Wireless Sensor Network, 2008 25

4 Anomaly Detection **41**
4.1 Mote-Based Online Anomaly Detection Using ESN
Marcus Chang, Andreas Terzis, and Philippe Bonnet
5th Intl. Conf. on Distributed Computing in Sensor Systems, 2009 . . . 45
4.2 Mote-Based Anomaly Detection
Marcus Chang, Andreas Terzis, and Philippe Bonnet
ACM Transactions on Sensor Networks (submitted) 59

5 Adaptive Data Acquisition **83**
5.1 Meeting Ecologist Requirements w/ Adaptive Data Acquisition
Marcus Chang and Philippe Bonnet
Intl. Conf. on Information Processing in SN, 2010 (submitted) 87

6 Conclusion **99**

Bibliography **105**

Chapter 1

Introduction

The Scientific Method

In the field of natural science, experiments play an integral role in proving and refuting hypotheses. As early as 1021, Ibn al-Haytham established the importance of verifying hypotheses with experiments in his book *Book of Optics*. Until then, science was based on imagination and logical deduction rather than experiments. This is why Galileo Galilei in 1638 could reject Aristotele's theories on the movement of falling objects in his book *Two New Sciences* by performing rigorous experiments. This method of letting the experiment rule over deduction was later formalized in the *Rules of Reasoning in Philosophy* in Isaac Newton's 1726 edition of *Principia* which paved the way for the modern scientific method, which mainly consists of these four points:

1. Statement of problem
2. Formulation of hypothesis
3. Testing of hypothesis using experimentation
4. Analysis of experimental results

Where (3) and (4) can either prove or disprove the hypothesis formulated in (2). In the latter case, it is necessary to return to either (1) or (2) and reformulate the problem and/or the hypothesis by using the knowledge gained from the experiment.

Because of the importance of rigorous experiments, new discoveries and scientific fields are often linked together with the progress of measurement instruments. For example, the invention of the microscope paved the road for microbiology, the same way the telescope revolutionized astronomy, and each generation of particle accelerators have proved the existence of more elementary particles from the Standard Model.

How the MEMS Revolution Changed Experimental Science

With the MEMS[1] revolution, it has become possible, both technologically and economically, to sample modalities in places and on scales previously not feasible. As an example, we use the progress of bird monitoring to show the impact of technology.

Previously, knowledge of birds was gained primarily by manual observation with binoculars and telescopes. This passive sensing relied solely on the human observer and only information such as numbers, race, and gender were easily available.

This process was later augmented with bird tagging, which made it possible to distinguish individual birds from the flock and track the same bird throughout seasons. By combining observations from different regions, it also became possible to track bird migration patterns. However, this process still requires human intervention and relies solely on humans being in the field to observe the particular birds that are tagged. Obviously, because of this it is not possible to keep track of all the birds who have been tagged.

After the MEMS revolution, bird monitoring has taken a quantum leap forward from relying completely on passive observations to active tracking and monitoring. Birds have been tagged with *dataloggers*,[2] equipped with small GPS receivers to continuously track their location. After the datalogger's retrieval, the bird's entire flightpath can be downloaded.

By combining the datalogger with inexpensive radios, we essentially have the mote platform. By forming networks of sensors (sensornets) with these motes, it became possible to download the measurements without having

[1]Microelectromechanical systems

[2]Sensors combined with stable storage and just enough processing power to sample the sensors at fixed intervals and store the data.

to retrieve the dataloggers first. After initial deployment, data are automatically downloaded from the motes through the sensornet and stored in databases for later retrieval, completely removing the human element from the data acquisition process.

The low cost has made it possible to vastly improve the density of data samples, both in space and time, and the small size has made it possible to combine modalities with fields previously unheard of. For example, in the Great Duck Island bird monitoring project by Mainwaring et al. [5], modalities such as light, temperature, pressure, and humidity were measured inside bird nests.

However, the sensors with the greatest technologically breakthroughs are mainly those used in the automotive, health, and consumer electronics industry such as accelerometers, gyroscopes, microphones, and imaging devices. Other sensors, not needed by these industries, have yet to see the same exponential evolution. These are mainly chemical, gaseous, and optical sensors which are still bulky and expensive.

Problem Statement

Sensornets have already revolutionized the way experimental scientists can collect data by offering automatic sampling both temporally and spatially denser than manual sampling allows, and at the same time in places and with modalities previously unattainable.

However, experimental scientists themselves have yet to embrace sensornets and utilize them in the same manner as they would any other instrument in their toolbox. Field deployments, such as the volcano monitoring by Werner-Allen et al. [13] and soil monitoring by Musăloiu-E. et al. [6] clearly illustrate this by the fact that half the people involved were computer scientists.

One of the reasons for this discrepancy lies in the complexity still involved with successfully deploying a sensornet, as illustrated by Barrenetxea et al. [1]. In order for experimental scientists, without the necessary computer skills, to embrace sensornets, this gap must be bridged. Design and visualization tools, such as those by Burns et al. [2] and Nath et al. [7] can alleviate deployment and data management problems, but sensornet management still remains an open issue.

Another reason why sensornets have yet to replace experimental scientists' manual sampling in the field is the lack of flexibility. It is not enough just to sample more places and more often; sensornets must also be as flexible and adaptive as a scientist in the field can be.

The problem, however, lies in the flow of information. Previous sensornets, such as ZebraNet by Huang et al. [4], forest monitoring in Redwood by Tolle et al. [12], and the river monitoring project LUSTER by Selavo et al. [10] have all been one-way. Motes are deployed, and measurements are sampled, stored and forwarded to the gateway. In order for sensornets to give the domain scientists the ability to adapt the same way they would have, had they been in the field, information must flow both ways.

This retasking can be achieved with sensornets such as *Tenet* by Gnawali et al. [3], but it is only useful if the domain scientist is actually present and monitoring the measurements. Because of the data explosion from the denser sampling and continuous operation, processing this data fast enough to reap the benefits of automatic data acquisition, by reacting faster to episodic events, would either require significant human resources or automatic online processing.

Processing measurements as soon as they become available could also help remedy the problem with faulty and missing measurements. Sensornets without online processing have been plagued with faulty measurements, because typically these faults are only discovered long after the experiment is over, making it impossible to repair the faulty equipment or redo the measurements. The soil monitoring sensornet *Suelo* by Ramanathan et al. [8] uses online processing of measurements to detect faulty equipment, however, their solution is human intervention and not automatic recovery.

For example, as part of the river monitoring program at the Zackenberg[3] ecologial research station, water samples are taken and the water level is recorded twice a day. This interval is normally sufficient to capture variations because changes normally happen on this time scale. However, because the river is connected to high altitude lakes which can freeze and melt several times during spring and fall, sudden flooding can occur, doubling the water level for a day or two. For an ecologist in the field, already taking manual samples, it is easy to discover this sudden flooding, to increase the measurement frequency during the flooding, and return to the normal sampling frequency afterwards. It is also easy for the ecologist to quickly verify the validity of the measurements and redo the samples if some faults have been introduced.

Obviously, with a sensornet in place, measurements could be taken more

[3]`http://www.zackenberg.dk`

often and without involving the ecologist. However, with the current level of sensornets it is not possible for ecologists to deploy sensornets and achieve the same level of adaptability as described above. In order for sensornets to replace ecologists in the field, they must be deployable by the ecologists themselves and have the same level of adaptability.

To summarize, the problems we seek to solve in this thesis are: How do we turn sensornets into viable instruments for the domain scientist to deploy in the field? How do we increase the quality of the data by avoiding faulty and missing measurements? And, with the domain scientist's limited time, how do we make the sensornet able to adapt to the environment in the same way the domain scientist would have done had she been present?

Thesis

The main issue with sensornets being scientific instruments, lies in the interaction between the domain scientist and the sensornet. Because, deploying a sensornet is still too computer intensive for a non-computer scientist, and manually controlling a sensornet in real-time is not practical.

We tackle these problems on three fronts. First, we introduce a benchmarking methodology to help choose the right set of motes and applications for the deployment. Second, we introduce an anomaly detection framework to increase the quality of the collected data. Third, we introduce a controller to mediate between the domain scientist and the sensornet, thereby making it easier for the scientist to control the sensornet, and react to changes in the collected data.

For the mote benchmarking, we seek a methodology to predict the outcome of running different applications on a variety of mote platforms, without actually having to port the applications and execute them. We base our solution on real traces instead of simulations in order to incorporate nondeterministic interactions from the environment. Our solution, inspired by Seltzer et al. [11], is based on decomposing applications into unit actions and measure the time and energy consumption for each action on different platforms.

For the anomaly detection, we seek a framework lightweight enough to be incorporated into the data collection application already running on the mote, in order to detect anomalies in the measurements as soon as they are sampled, and thereby allowing the mote to react immediately. We base our detection algorithms on machine learning, to avoid misclassification problems between different types of anomalies.

Finally, to act as a middleman between the domain scientist and sensornet we introduce a controller based on the *Planning and Scheduling* [9] architecture known from the Artificial Intelligence community. This allows the domain scientist to define a set of overall *goals*, reflecting the scientist's requirements to the collected data. Based on a model and the actual state of the system (i.e., sensornet and environment) the controller plans a set of *actions* to achieve these goals.

However, for the controller to construct a plan that both satisfies the scientist's requirements and utilizes the sensornet's resources most efficiently, it is necessary to estimate the time and energy consumption of each plan under consideration. We base our resource model on deconstructing each plan into unit actions according to our benchmarking methodology above. With this model in place the controller can reason about whether a plan is both feasible and efficient, in terms of available time and energy.

Another consideration the controller must take into account, is the state of the environment. For the controller to be adaptive, it must be able to detect changes the domain scientist would find interesting and disregard irrelevant ones. Using the anomaly detection framework above, the controller becomes capable of distinguishing between what the scientist deems important and unimportant, and by processing the acquired data online, the plans the controller constructs will reflect the actual state of the environment and system.

Returning to the river monitoring example above, two goals based on the ecologist's requirements could be (1) take at least one water sample every hour, and (2) during flooding, take at least six water samples every hour if the energy reserve permits it. It is then the controllers responsibility to ensure that given unexpected events such as faulty measurements or rising water levels that the goals are achieved by either retaking water samples or increasing the sampling rate.

In summary, our thesis for this dissertation is:

Sensornets should become instruments that domain scientists can pick, deploy, and program to efficiently collect high quality datasets.

Contributions

This dissertation has the following three major contributions:

- A vector-based methodology to study mote performance. Our method divides resource consumption into two components, one characterizing the specific application and one characterizing the platform in general. By combining these vectors, we are able to both predict the energy and time consumption of the same application on different platforms, and different applications on the same platform.

- An anomaly detection framework based on machine learning algorithms. We show that the machine learning algorithms are superior to the simple heuristics used in previous sensornets and do not suffer from the same misclassification problems, where faults are mistaken for events and vice versa. Furthermore, with a TinyOS implementation we show that these algorithms are lightweight enough to perform online anomaly detection on a mote.

- A sensornet controller based on artificial intelligence. We capture the scientist's requirements in terms of data collection modes, consisting of a range of acceptable parameters, and use these to drive the controller. By utilizing our resource prediction methodology to reason about efficiency and feasibility, and our detection framework to track changes in the environment, the controller can choose the actions that best meets the scientist's goals given the state of the environment and available resources.

The rest of this dissertation is structured around our move towards scientific sensornets and the change from automatic to adaptive data acquisition. In Chapter 2 we look back at our initial sensornet deployments. We present the lessons we learned and the problems that motivated the development of our three front approach. In Chapter 3 we present the first cornerstone, the characterization of mote performance with the purpose of facilitating sensornet design and deployment. In Chapter 4 we present the second cornerstone, the machine learning based anomaly detection framework to increase the quality of the collected data. In Chapter 5 we present the third cornerstone, the AI-based sensornet controller to mediate between the domain scientist and sensornet. Finally, in Chapter 6 we offer our concluding remarks and directions for future work.

Bibliography

[1] G. Barrenetxea, F. Ingelrest, G. Schaefer, and M. Vetterli. The Hitchhiker's Guide to Successful Wireless Sensor Network Deployments. In *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 43–56, New York, NY, USA, 2008. ACM.

[2] R. Burns, A. Terzis, and M. Franklin. Software tools for sensor-based science. In *Proceedings of the Third Workshop on Embedded Networked Sensors (EmNets 2006)*, Feb. 2006.

[3] O. Gnawali, B. Greenstein, K.-Y. Jang, A. Joki, J. Paek, M. Vieira, D. Estrin, R. Govindan, and E. Kohler. The TENET Architecture for Tiered Sensor Networks. In *ACM SenSys 2006*, Nov. 2006.

[4] P. Huang, H. Oki, Y. Wang, M. Martonosi, L. Peh, and D. Rubenstein. Energy-efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet. In *Proceedings of the Tenth Internationla Conferece on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X)*, Oct. 2002.

[5] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 88–97, New York, NY, USA, 2002. ACM.

[6] R. Musăloiu-E., A. Terzis, K. Szlavecz, A. Szalay, J. Cogan, and J. Gray. Life Under your Feet: A Wireless Soil Ecology Sensor Network. In *EmNets Workshop*, May 2006.

[7] S. Nath, J. Liu, and F. Zhao. Sensormap for wide-area sensor webs. *Computer*, 40(7):90–93, 2007.

[8] N. Ramanathan, T. Schoellhammer, E. Kohler, K. Whitehouse, T. Harmon, and D. Estrin. Suelo: Human-assisted Sensing for Exploratory Soil Monitoring Studies. In *SenSys '09: Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, pages 197–210, New York, NY, USA, 2009. ACM.

[9] S. Russell and P. Norvig. *Artificial Intelligence A Modern Approach, 2nd Ed.* Prentice Hall, 2003.

[10] L. Selavo, A. Wood, Q. Cao, T. Sookoor, H. Liu, A. Srinivasan, Y. Wu, W. Kang, J. Stankovic, D. Young, and J. Porter. LUSTER: Wireless Sensor Network for Environmental Research. In *SenSys '07: Proceedings of the 5th international conference on Embedded networked sensor systems*, pages 103–116, New York, NY, USA, 2007. ACM.

[11] M. Seltzer, D. Krinsky, K. Smith, and X. Zhang. The Case for Application-Specific Benchmarking. In *Hot Topics in Operating Systems, 1999. Proceedings of the Seventh Workshop on*, pages 102–107, 1999.

[12] G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, and W. Hong. A Macroscope in the Redwoods. In *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 51–63, New York, NY, USA, 2005. ACM.

[13] G. Werner-Allen, K. Lorincz, M. Welsh, O. Marcillo, J. Johnson, M. Ruiz, and J. Lees. Deploying a Wireless Sensor Network on an Active Volcano. *IEEE Internet Computing*, 10(2):18–25, 2006.

Chapter 2

Automatic Data Acquisition

Due to the MEMS revolution, sensornets have made it possible to monitor ecological environments and animal life much denser in space and time and with modalities previously not possible. More importantly, this allows domain scientists to design new experiments and test theories previously not possible to perform in the field.

For example, the gender of the Box Turtles[1] is not determined at conception, but by the temperature in the nest during incubation. At least, this is what laboratory experiments show. But how do we determine if the parameters we observe in the lab also apply in nature as well? I.e., will temperatures between 27-28$°C$ result in male turtles and temperatures between 29-30$°C$ in females?

The only way to prove this, is to actually measure the temperature in a nest with eggs found in nature. This is exactly what the sensornet deployed by Szlavecz et al. [2] did. They put temperature and moisture sensors inside the Box Turtles' nests and used motes to sample these sensors and store the measurements. This data was then periodically forwarded to a gateway and stored in a database.

We faced a similar situation in the Hogthrob project where we used a sensornet to detect oestrus in household sows. The project was a three-party collaboration between (1) the swine production industry, which main goal is to maximize production by keeping the sows healthy and reproductive, (2) the veterinarians, who wish to understand the sows' behavior and build a model of this, and (3) the computer scientists, who develop and manage the sensornet.

[1]`http://en.wikipedia.org/wiki/Box_turtle`

Oestrus detection is important to the farmers, in order to maximize the sows' reproduction, and important to the veterinarians, as part of their behavior model. Traditionally, this detection has been done manually either by back pressure tests, close visual inspection, or measuring the rectal temperature.

Obviously, all three methods are slow and cumbersome and do not scale well for large households. As it turns out, the onset of oestrus is closely related to the sows' physical activity level, and a suitable modality to express this physical activity would be acceleration. For the veterinarian, this could also contribute to the behavioral model since each kind of activity has its own distinct acceleration pattern. Because accelerometers have become small, inexpensive, and power efficient enough to be equipped on a mote, it is possible to mount one on each sow using a neck collar. This allows us to acquire acceleration data, with minimal intrusion to the sows' daily routines and environment and with minimal manual labor for the farmers.

In "Lessons from the Hogthrob Deployments" we describe two field deployments of the sow monitoring sensornet and the development process leading up to each. The paper was presented at the *Second International Workshop on Wireless Sensor Network Deployments, 2008* and made in collaboration with Cécile Cornou, Klaus Madsen, and Philippe Bonnet.

We successfully acquired enough data for the veterinarian to build her sow behavioral model and we were also able to devise simple detection algorithms to determine whether or not a sow is in heat. Compared to a similar sow monitoring sensornet, *Wired Pigs* by McCauley et al. [1], we are able to achieve a higher throughput and thereby sustain a higher sampling rate.

However, we also discovered several fundamental problems which we since then have used to guide our research:

First, based on the problems we encountered during the first deployment we changed the hardware platform for the second experiment. But, choosing the right mote for the right job is not trivial. What we missed, was a systematic approach to predict the behavior of an application on a new platform without having to actually deploy it. This is especially important for resource constrained platforms, where applications might be too resource intensive to achieve the desired sampling rate and stay within the energy budget. More generally, we need a way to reason about system resources, such as energy and time, in order to utilize them optimally. Such resource awareness will not only help choosing the right platform but also help designing and adapting the application to maximize the sensornets' value. For example, in both deployments the batteries lasted longer than the actual experiments. This surplus energy could have been used on a higher duty-cycle which potentially could have reduced data loss.

Second, even with a simple one-hop star topology, the network connectivity varied greatly because of the changing environment, i.e., the sows themselves impacted the radio link quality when they moved around leading to periods with network outage. Although samples were stored for later transmission, these outages still resulted in memory shortage leading to loss of data. We did experiment with a degraded operation mode, but what we need is a more flexible mechanism to throttle the sampling rate given the state of the system in order to utilize the resources more efficiently and meet the consistency requirement of the domain scientist. In fact, this mechanism should also consider the sensor readings from the environment as well, since different situations might call for different sampling strategies. Because of network outages, this mechanism cannot solely rely on the more resourceful gateway but must also have a mote based component.

Last, we used the yield (fraction of acquired data to the theoretical maximum) as our performance factor (which seems to be the *de facto* standard in sensornets). From the computer scientist's perspective this seems reasonable, however, as it turns out this is not necessarily the case for the domain scientists. The problem is, yield only describes the amount of data missing and not the distribution of it. For the domain scientists, consistency is more important and missing data spread out evenly over time is preferred to clusters of missing data. What we need is a performance factor that reflects the domain scientist's requirements and a way to dynamically adapt the sampling strategy to optimize this factor.

Bibliography

[1] I. McCauley, B. Matthews, L. Nugent, A. Mather, and J. Simons. Wired Pigs: Ad-Hoc Wireless Sensor Networks in Studies of Animal Welfare. In *Proceedins of the Second IEEE Workshop on Embedded Networked Sensors (EmNetS-II)*, May 2005.

[2] K. Szlavecz, A. Terzis, R. Musaloiu-E., C.-J. Liang, J. Cogan, A. Szalay, J. Gupchup, J. Klofas, L. Xia, C. Swarth, and S. Matthews. Turtle Nest Monitoring with Wireless Sensor Networks. In *Proceedings of the American Geophysical Union*, 2007.

Lessons from the Hogthrob Deployments

Marcus Chang, Cécile Cornou, Klaus S. Madsen, Philippe Bonnet
University of Copenhagen

Abstract**—Today, even small-scale sensor networks need to be carefully designed, programmed and deployed to meet user goals, such as lifetime or quality of the collected data. Only by confronting the problems that do come up during actual deployments can we devise the tools, methods and abstractions that will help meet the requirements of a sensor network infrastructure. In this paper we report on the lessons we learned deploying two experimental sensor networks in a pig farm in the context of the Hogthrob project. We describe the design of the sensor networks whose goal was to collect time series of sow acceleration data over a menstrual cycle, i.e., circa 30 days, and we discuss the lessons we learned.**

Index Terms**—sensor network deployments, group housed sows, acceleration time series.**

I. Introduction

In the context of the Hogthrob project[1], we aim at developing an infrastructure for online monitoring of sows in production. Today, farmers use RFID based solutions to regulate food intake. Such systems do not allow the farmers to locate a sow in a large pen, or to monitor the life cycle of a sow (detect oestrus, detect injury, etc.). Our long term goal is to design a sensor network that overcomes these limitations and meets the constraints of the pig industry in terms of price, energy consumption and availability.

In a first phase, we focus on oestrus detection. We use a sensor network to collect the data that scientists can use to develop an oestrus detection model based on the sow's activity. Such sensor networks promise to be part of the experimental apparatus scientists can use to densely sample phenomena that were previously hard or impossible to observe in-situ, and then build appropriate models.

Automated oestrus detection has been mainly studied for dairy cattle and for individually housed sows. Group housing complicates the problem significantly as it may impair the oestrus behavior of subordinate sows. Possible methods for automated oestrus detection for group housed sows are reviewed in [CC06]. Our goals are (i) to assess whether it is feasible to detect the onset of oestrus using acceleration data and (ii) to devise a detection method that is appropriate for online monitoring.

We designed a sensor network that collects sow acceleration data during a complete oestrus cycle (21 days in which the sows show one oestrus). In this period, the nodes should remain unattended in order to avoid disturbing the sows. This is a tough challenge in terms of energy consumption as packaging constrains the size of the batteries and thus the energy budget.

We deployed our sensor network on a production herd in Sjælland, Denmark[2] in a 264 m^2 indoor pen (see Figure 1). Because of cost considerations (we need to refund the farmer for the missed opportunity to perform artificial insemination) we reserved a limited number of sows.

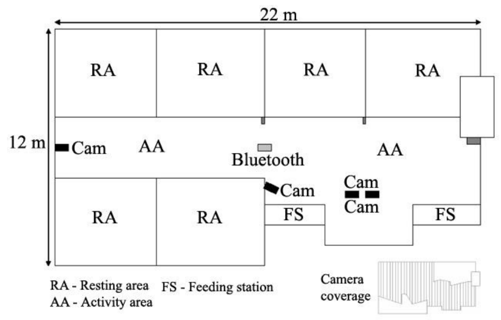


Figure 1: Plan of the Experimental Pen

We ran a first experiment during a four weeks period in February and March 2005 and a second in January and February 2007.

In this paper, we report our experience preparing, conducting and analyzing these field experiments:

1) We describe the design of the sensor network. Our first experiment focused on the power budget and on energy consumption in order to keep data collection running for 3 weeks. We relied on the lessons learned from previous deployments as they were related in the literature when planning the experiment [SITEX02][GDI02][GDI04]. We thus paid particular attention to packaging, battery selection, duty cycling, and replication in the back-end infrastructure. In our second experiment, we focused on improving the yield of accurate collected data with a goal of 90%.
2) We discuss the lessons learned and how our results can be applied in a larger context.

II. First Experiment

Our goal was to design a sensor network for the collection of sow acceleration data sets. We faced two key problems, typical of sensor networks deployed for scientific data collection:

1) What is our power budget? How do we duty cycle the sensor nodes to keep energy consumption within our budget?
2) How do we collect the data from the sensor nodes to the back-end infrastructure?

[1] A collaboration between DIKU, DTU, KVL, IO Technologies and NCPP (http://www.hogthrob.dk).

[2] www.askelygaard.dk

A. Sensor Network

Sensor Nodes. We decided to use BTnodes [Beutel] as a hardware platform. We used Btnodes rev2, equipped with an ATMega128 micro-controller, 64 KiB of external RAM, and an Ericsson ROK 101 007 Bluetooth module. Our goal was to use the high-bandwidth radio[3] of the BTnodes to transmit large amounts of data. This opens up for a model where the sensor data is stored locally on the node as it is captured, and periodically transmitted in large chunks to the base station. We thus leverage the radio's bandwidth in order to reduce its duty cycle. We developed an accelerometer board for the BTnode and designed a node packaging that could fit the sow's neck.

Sensor Data. In his experiment with individually housed sows, Geers [Geers95] used a sampling rate of 255 Hz. An experiment with the MIT LiveNet system used a 50Hz sampling rate to detect shivering [Sung04]. We estimated that a 4 Hz sampling rate would be good enough to capture the activity of sows.

Accelerometers. The most important parameter of an accelerometer is the range of acceleration it can measure. The acceleration experienced when a human walks is in the range of 0-2g. We assume that the acceleration of a sow walking will be in the same range, while the occasional fights could lead to higher accelerations. For the sake of redundancy and comparison, we equipped our board with a 2D analog accelerometer (ADXL320 from Analog Devices) as well as a 3D digital accelerometer (LIS3L02DS from STMicroelectronics).

Packaging. We needed an appropriate packaging in order to attach a BTnode to the sow's neck. We had to design a protective casing that (a) could contain a BTnode with the accelerometer board and batteries, (b) would fit the shape of a sow's neck and (c) would be robust enough to resist the hostile environment.

We designed a box in a carved oblong form that fits the shape of a sow's neck (Figure 2). The box is in effect air tight (23 screws to fix the top, with an o-ring underneath, to seal it). It is 135x33x80mm. Its walls are 5 mm thick to resist bites. The box can contain a BTnode equipped with the accelerometer board as well as a pack of 4 AA size batteries. Note that the protective casing turned out to be the most expensive part of the equipment. Attaching this box to the sow's neck was also a challenge. We finally devised a reliable method using medical tape and a sow collar.

Power Budget. In our experience, BTnodes exhibit unstable behavior with an input voltage of around 3V. We thus decided to use a 4x1.2V battery pack to make sure that the node could deplete the batteries fully before encountering brown-outs. For economic reasons, we decided to use rechargeable batteries. We decided against Lithium cells because of their sensitivity to shocks and their price. Instead we opted for NiMH batteries.

We experimented with three different cells: One from Panasonic with a nominal capacity of 2100 mAh, and two from Ansmann with nominal capacities of 2300 and 2400 mAh. A constant discharge experiment confirmed the capacity figures given by the manufacturers as well as the promised flat discharge curves. Our power budget was thus around 10000mWh.

Duty Cycling Model This limited power budget led us to define an aggressive duty cycling model. First, we should keep the power consumption of the BTnode to a minimum when not working. Our initial experiments showed a consumption of 54 mW for an idle BTnode. We succeeded in bringing this figure down to 2,4 mW by entering a deeper sleep mode and disabling the external memory.

Second, in order to transmit large chunks of data, we had to store the accelerometer samples. This was a bit of a challenge with the external memory disabled. We decided to use the unused parts of the program memory, i.e., the 128 KiB flash inside the ATMega128 micro-controller. Because 8 KiB were reserved for the boot loader and our code occupied around 20 KiB, we could use around 100 KiB of program memory to store sensor data which in turn corresponded to around 55 minutes of sampling. Thus, in our duty cycling model the radio was turned off until the memory was almost filled up.

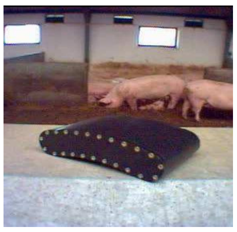


Figure 2: The sensor node box.

Back-end Infrastructure. For our experiment, the role of the back-end infrastructure was twofold. First, it should obtain and store data from the sensor nodes. Second, it should collect ground truth about sow activity, i.e., video recording of the pen.

For data collection we used a star topology with all the sensor nodes in direct communication range of the base station, which was a PC connected to an external Bluetooth dongle hanging from the ceiling in the middle of the pen. Its role was (a) to communicate with the sensor nodes and (b) to timestamp and store the received packets.

In order to check if the range of a single Bluetooth dongle provided sufficient coverage, we placed a BTnode (with its protective casing) in different corners of the pen and verified that connections could be opened and data transferred. This test did not reveal any problems.

The communication between the sensor nodes and the base station used the following protocol. The base station continually sends inquiries. Whenever required by the duty cycling model, a sensor node starts an inquiry scan (a scan requires less energy than sending inquiries [Leopold03]). When a base station detects a sensor node, it creates a connection. When the connection is established, the sensor node sends status information including the number of samples it contains. The server does the bookkeeping; it requests the samples that have not yet been transmitted and acknowledges their reception. The sensor node stores a

[3] 434 kbit/s bandwidth compared to (at that time) contemporary radios like the CC1000 which only had 76.8 kbit/s.

sample until its transmission has been acknowledged. When memory is full, new samples are simply dropped.

The key issues when designing the infrastructure were:

• Timestamping: In order to analyze the data sets, we needed to control that a sow was actually moving when the data set indicated some activity. The same PC was used to assign a global timestamp to both video frames and sensor data. The PC was connected to the Internet and synchronized with a remote NTP server.

The timestamping of the video frames was done by the frame grabbing software while the timestamping of the time series was done on each sensor node by attaching a logical timestamp (a counter) to the samples it collected. The PC then associated the logical timestamp with the global timestamp on each batch of samples it got from a particular sensor node (in the context of one Bluetooth connection). Global timestamps were attached to individual samples in a post-processing phase.

• Data storage: Each frame stored in PNG format occupied 8 KiB. We expected a stream of 32 KiB per second for each camera and a total of 80 GiB of data per camera for the duration of the experiment. We used an ADSL line with an uplink capacity of 512 KiB/sec to connect the base station PC to the Internet. We reserved 1TB of disk space on a remote file server to store the video and the time series.

• Reliability: We paid particular attention to reliability as it had been reported to be a problem in previous deployments [GDI02]. We chose a straightforward approach and basically replicated all the processing on two base station PCs. Both PCs were equipped with their own Bluetooth dongle. We splitted the coaxial cables so that each PC was connected to all the cameras. Both PCs ran NTP clients connected to the same NTP server. They proceeded in parallel with the timestamping of both video frames and time series. Both PCs shared the ADSL line connecting the farm to the Internet. A fail-over service monitored both servers and picked one of the PCs to upload data to the remote file server.

• Resilience to the hostile environment: Sows are aggressive animals and they will try to eat any equipment they can put their mouth on. The pen is a hostile environment as well mainly due to the corrosive ammonia. We decided to place the PCs in a container outside the pen for protection and used USB extenders, connected via Ethernet cables, to connect the PCs to the Bluetooth dongles.

B. *First Field Experiment*

The experiment took place from February 21st to March 21st 2005. Data was collected from the sensor network from Day 11 to Day 30. In this Section we report on the data collection process using our sensor network and we describe the collected data.

1) Hits and Misses

Sensor node lifetime was our main concern when designing the sensor network. Our lab experiment suggested a possible lifetime of around 60 days. In fact, 4 of the nodes could collect data for the duration of the experiment. The fifth sensor node lost its Bluetooth module a week before the end of the experiment and had to be replaced.

Battery drain information was measured by the BTnode and sent to the base station in the status packet that prefixed all transmissions. Figure 3 shows the voltage drain for one of the sensor nodes equipped with an Ansmann battery pack. We obtained a flat discharge curve very similar to the one we obtained with constant discharge. Note that the actual voltage of the cell was higher than the nominal 1.2V and we obtained around 5V for the duration of the experiment. From day 37 on, there was still enough power left in the batteries to turn the Bluetooth module on, receive a connection and send the initial packet. However during the send phase, the voltage of the batteries dropped so low that the node browned out.

The main problem we encountered during the experiment was missed connections between the base station and sensor nodes. We expected the sensor nodes to upload their data approximately once an hour, but we only observed an average of 16 connections a day. Once a connection was established, though, data transfer was successful. We observed that these missed connections occurred mostly while the sows were in the corners of the pen. It is most probably the water contained in the body of the sow and the iron contained in the internal alcoves of the pen that limit the communication range of the sensor node. As a result our time series contain numerous holes (see next Section).

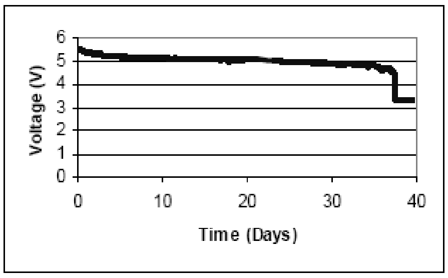


Figure 3: Voltage drain during the experiment for a sensor node equipped with an Ansmann battery pack.

Another problem was that sensor nodes rebooted every other day. The only node that did not reboot was equipped with a custom made battery pack with batteries welded together. As a result, the samples that had been collected but not transferred were overwritten. Also, the reboot caused the logical timestamp counter to reinitialize. Fortunately, we had programmed the nodes to connect to the base station only 10 minutes after a reboot so we avoided cascading effects.

We realized after the experiment was finished that our two PCs (connected to the same NTP server) were approximately 20 seconds out of sync. This problem combined with the numerous node reboots made the post-processing of the sensor data complicated and error prone.

2) Collected Data

We collected around 200MiB of sensor data per node[4].

[4] The data sets are available at http://hogthrob.42.dk/. Our agreement with the farmer does not allow us to release the video images.

We validated a posteriori the validity of the data collected from the analog and digital accelerometers by converting the raw measurements to the gravitational acceleration. This verification showed that most of the accelerations measured with the digital accelerometer were around 1g as expected.

The analog accelerometers however showed some weird behavior. A closer look at the sensor node application revealed that the analog accelerometers were turned on for too short a period (20ms while 80ms would have been needed). As a result the measurements from the analog accelerometers were unusable. Such a bug that slipped our lab experiments should have been detected and fixed while the experiment was running, not afterwards.

The time frame we use as reference for our analysis is 01/03/05 at 00:00 to the 21/03/05 at 00:00. Compared to a theoretical number of 6912000 samples[5] we obtained a yield of 53%, 62%, 61% and 71% respectively for each node. These figures are reasonable compared to previous sensor network deployments [Red05], but very low when compared to the actual available data. Looking back, it was a mistake not to specify the yield as a primary objective for our experiment.

We use the collected acceleration data to assess whether it can be used to detect oestrus. We want to distinguish between periods of activity where the relative acceleration is high, and periods of calm where the relative acceleration is close to null. For this purpose, we use the Omniburst stream processing system developed at NYU [Shasha04]. This system finds bursts (i.e., subsequences with abnormal aggregates) of various durations within a time series.

In our case, we are interested in short periods with intense accelerations as well as longer periods where a lot of possibly less intense accelerations occur. The periods where a sow is inactive will not contain bursts. We focus on short bursts (window size 100 samples) where 90% of the measurements are above a given threshold and long bursts (window size 1000 samples) where 60% of the measurements are above the threshold. We use a low threshold value for the relative acceleration (i.e., we only consider that a sow is inactive if the relative acceleration is close to zero).

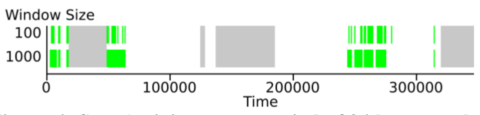


Figure 4: Sow Activity over a period of 24 hours at the start of the experiment (March 1st)

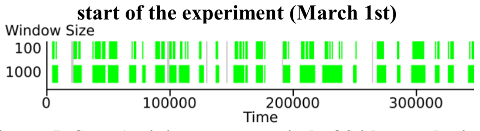


Figure 5: Sow Activity over a period of 24 hours during its heat period (March 15th)

Figure 4 shows the output of Omniburst on the time series corresponding to a day of measurement (outside the heat period). The green bars on the graph correspond to active periods, the white areas correspond to inactive periods and the grey areas correspond to holes in the time series due to missed connections. We validated on the video the alternation of active and inactive periods described by the graph. The long inactive period between logical timestamps 80000 and 230000 actually corresponds to a good night sleep. Figure 5 shows the activity of a sow during its heat period. The difference in the level of activity with respect to Figure 4 is striking, in particular at night. This is encouraging for the definition of a detection model.

III. Second Experiment

The goal we set for ourselves for the second experiment was to reduce the holes in the collected time series and reach a yield of 90%.

A. Sensor Network

We made some key improvements for the second experiment both in terms of hardware and software. In terms of hardware, we made the following adjustments:

- In the first experiment our aim was to leverage the high bandwidth of the Bluetooth radio. However, even if Bluetooth had a superior bandwidth there was also a higher overhead cost in terms of the time it took to establish a connection, often in the range of 30 seconds or more. Newer low power radios, like the Chipcon CC2420, offer 250 kbit/s bandwidth with little to none connection time, depending on the protocol in use. With this radio the process of offloading all the data could be done before the Bluetooth radio even established connection.
- The price on non-volatile storage has dropped significantly and has become widely available on sensor nodes. By adding external storage we increased the amount of samples stored and thus increased the time the node could be out of range without dropping measurements. This gave us both better duty-cycling and lowered the effect of black-out periods.
- We added directional antennas to the base station to increase signal strength. Since the movement of the sows was restricted to the ground, using directional antennas instead of omni-directional antennas would increase the signal strength with at least a factor 2. We could not go further and add antennas to the sensor nodes, as it would be impossible to keep the antennas aligned and operational – sows tend to roll over a lot.
- In order to reduce spontaneous reboots, we only relied on welded battery packs.

In terms of software, we introduced the following features:

- We increased the storage capacity by compressing the collected data.
- We defined a degraded mode for transmissions. In the first experiment, measurements could be dropped in case the flash was full. This time, we anticipated the lack of storage space and we defined a lossy compression scheme that allowed us to trade a lower data resolution for reduced space occupation.

Sensor Nodes. For the second experiment we decided to use the Sensinode Micro.4 as a hardware platform. It is equipped with a TI-MSP430 microcontroller, 512 KiB external FLASH, and a Chipcon CC2420 radio.

[5] Four times per second times 3600 seconds per hour times 24 hours a day times 20 days.

Accelerometers: The Sensinode Micro.4 came with an optional accelerometer board equipped with an analog 3D accelerometer (MMA7261Q from Freescale Semiconductors). The range of the accelerometer was set at ±2.5g and the analog output was converted to a 12-bit digital signal by an extra chip on the accelerometer board. This ensured that there were no timing issues with the microcontroller.

Power Budget: The Sensinode Micro.4 requires a supply voltage of 1.5-2.6 V, so a 2x1.2 V battery pack was sufficient. Again we opted for rechargeable NiMH batteries but since the first experiment, newer batteries with increased capacity had become available. Specifically, we chose the Panasonic 2600 mAh, which gave us an approximate power budget of 6000 mWh.

Note that the estimated power budget is significantly lower than the one used for the first experiment with the BTnode. Also note that the idle consumption for the Sensinode Micro.4 is 2.4 mW (same as the Btnodes) and that the Chipcon CC2420 radio has the same power consumption as the Bluetooth radio during send and receive. However, the Bluetooth radio has a mandatory inquiry phase each time a connection is established, that uses a staggering 165 mW for at least 10 seconds. The Chipcon CC2420 does not have this long discovery period and as a result overall energy consumption is much lower with the Micro than with the Btnode.

Duty Cycling Model: We adapted the same strategy as in the first experiment, however, with the increased storage and with additional data compression, we were able to store 5-10 hours of data (compared to the 55 minutes in the first experiment). Since the connection time was virtually instant, we also used a more relaxed offloading scheme, where we initiated the first offload attempt already when the node was half full. If this failed we tried again after a back off period, with each period defined by the halving of the remaining free space until a minimum period was reached. As the memory started to run out the back off period thus became shorter.

Compression: We note that in order to increase the yield, it is important not to run out of storage space, since this means new measurements will be dropped instantly. We thus added a lossless compression algorithm before storing the data. In extreme cases where repeated offload attempts had failed and the node was about to run out of space we shifted to lossy compression.

The lossless compression used is a simple differential algorithm with variable bit-rate. The data is divided into blocks, so that each block can be decompressed independently. The first measurement in each block is stored fully, together with the timestamp and the readings from all three axes, <x1,y1,z1>. The timestamp is only stored for the first data point, since the sampling rate is fixed. When the next measurement <x2, y2, z2> is about to be stored, the difference on each axis is calculated <x1-x2>, <y1-y2>, and <z1-z2>, and only this difference is stored.

To minimize the header size for each data point we choose four discrete bitrates to store the differences in: 4, 7, 9 or 12 bits, with the last value being no compression. Lab experiments [Madsen06] show that our custom compression algorithm outperforms (in terms of code size, compression ratio and power consumption) both Huffman and Lz77 compression, when used on this particular kind of data.

In the sow behavior statistical models (developed based on the data from the first experiment and not presented here because of lack of space [Cornou07]), measurements are grouped together and averaged. The least intrusive lossy compression is thus an average of several values. For our lossy compression scheme, we take the average of four measurements, which equals a second with our sampling rate, and stores it with the lossless algorithm mentioned above. Another bit in the header is used to indicate whether or not a lossy compression has been performed.

In summary, data points are thus stored as 15 bit, 24 bit, 30 bit, or 39 bit values.

Back-end Infrastructure: With the Chipcon CC2420 radio instead of Bluetooth, the communication protocol became slightly different. To minimize the time during which the radio was active, we changed the polling process of the base station into a push process of the sensor nodes.

After the connection had been made, we reused the offloading protocol from the first experiment. It should be noted that the discovery process takes less than a second to complete, making the process at least a factor 20 cheaper than with the Bluetooth radio.

In this experiment our base station was a PC connected to a Sensinode Micro.4 node, which acted as a bridge, meaning it handled the discovery and connection, but otherwise forwarded the data to the PC. The PC handled the offloading protocol, timestamping and decompression. Also, the node was equipped with a Cisco ceiling mounted directional antenna. As in the first experiment, we did a coverage survey of the pen which yielded no problems, as expected.

The 4 analog black/white video cameras from the first experiment were replaced by 2 digital color cameras mounted opposite each other in the ceiling. Each camera acted as a webcam and frames were grabbed by a standard PC. The cameras were NTP enabled and embedded a timestamp directly into each frame. The timestamping of the time series remained the same as for the first experiment.

Again we used redundancy to increase reliability. The only difference with the first experiment was that we used 2 PCs for storing the time series and 2 PCs for video recording. We also added a heart-beat-pulse between the base station PC and the bridge node, so it did not offer connections if the PC application was unavailable.

B. Second Field Experiment

The experiment took place from January 24th to February 24th 2007. 12 sows were selected and divided into two groups with 6 in each. Two different experiments were conducted. The first focused on oestrus detection and lasted 3 weeks with both groups of sows involved. The second was an activity observation experiment which lasted for a week and involved only one group. This group was reequipped with fresh batteries after the first experiment.

1) Hits and Misses

Although our power budget was more strained this time, none of the nodes ran out of power. The main problem we observed was that the soldering on one of the batteries broke off on one of the last days and thus killed the node for the rest of the experiment.

Compared to the first field experiment the amount of received data was significantly higher and we did not experience any spontaneous reboots. We did, however, experience data corruption at the flash level which resulted in duplicate timestamps and invalid values. Whether this is due to faulty hardware or a bug in the driver is unknown.

2) Collected Data

We collected around 170 MiB of raw sensor data per node. This is numerically less than the first experiment, because we only collected 3D measurements. After the corrupted data were cleaned up, timestamp expanded and calibrated to SI units, each data set occupied 450 MiB.

Table 1 shows the yield for each node. The column marked “% recv” is the percentage of received data samples out of the measured samples and the column “% clean” is the actual usable data after the corrupted data is removed. The average yield for the oestrus detection experiment is 92.7% and 89.3% for the received and cleaned data respectively. It should be noted that *node 11* is the one where the batteries fell off, which explains the lower yield. For the activity detection experiment the yield is 92.0% and 90.3% for the received and cleaned data respectively, which is consistent with the oestrus detection experiment.

These percentages are significantly higher than our first experiment and previous sensor network deployments in general. Looking at the reception percentage it is clear that we still have connection issues, similar to the first experiment. Specifically, blackouts occur when the sows are sleeping on top of the nodes, but since our nodes can actually hold more data than the average sleeping period of the sows, it should be possible to devise a strategy that is resilient against this problem, e.g., offload every hour or every other hour instead of every 5-6 hour as it is now.

Node id	1	2	3	4	5	6	7	8	9	10	11	12
% recv	92	93	94	94	94	90	96	95	95	96	80	93
% clean	86	88	94	94	89	80	95	94	94	95	75	86

Table 1. Yield for the 2nd Experiment.

IV. Lessons Learned

The sensor networks that we deployed in Hogthrob were quite simple with few sensor nodes, a single modality, and no multihop. Still we faced many challenges to meet our goals in terms of lifetime and yield, and we learned lessons that apply to sensor network-based data acquisition systems in general:

1. **Model aware vs. Model agnostic data acquisition.** In order to improve yield, it is necessary to design degraded operations modes that kick in to preserve data in case of communication failures, or lack of storage space. The design of appropriate degraded modes should be based on how the collected data is to be modeled.

2. **Mote characteristics.** The amount of storage available, the time during which the radio must be on to transmit data, and the power consumption in sleep mode are key characteristics that impact the duty cycling policy and thus should drive the decision of which mote to use for a given deployment.

3. **Offline vs. Online adjustments.** We collected data using a best effort approach, then analyzed it offline and proceeded to adjust our data collection efforts. Ideally, the sensor network should be deployed so that (a) the collected data is analyzed online, and (b) the data acquisition methods are adjusted to guarantee that user requirements are met (e.g., interesting events are caught and faults are compensated for or signaled).

4. **Post-processing**. The design of the data acquisition system should include both post-processing (cleaning, timestamping, decompression, calibration) as well as transformation and loading into the format used for data analysis.

V. Conclusion

We described the experience we gained deploying sensor networks to collect sow activity data sets. We believe that the sensor network we designed is representative of the experimental apparatus needed for initially exploring an application domain. The lessons we learned should be useful for future deployments.

VI. References

[Beutel] Jan Beutel. BTNode Project Page. http://www.tik.ethz.ch/~beutel/btnode.html

[CC06] C. Cornou, “Automated oestrus detection methods in group housed sows”, Livestock Science, 2006, ISSN: 1871-1413

[Cornou07] C. Cornou, “Automated Monitoring Methods For Group Housed Sowst” PhD Thesis. Faculty of Life Sciences. U. Copenhagen. 2007.

[GDI02] A. Mainwaring, J. Polastre, R.Szewczyk, D.Culler, J. Anderson “Wireless Sensor Networks for Habitat Monitoring”, WSNA *2002. GA, USA*

[GDI04] R.Szewczyk, J. Polastre, A. Mainwaring, D. Culler. “Lessons from a Sensor Network Expedition”. EWSN 2004.

[Leopold03] M.Leopold,M.Dydensborg, Ph.Bonnet, “Bluetooth and Sensor Networks: A Reality Check”, Sensys 2003.

[Geers95] R.Geers, S.Janseens, J.Spoorenberg, V.Goedseels, J.Noordhuizen, H.Ville, J.Jourquin, “Automated Oestrus Detection of Sows with Senors for Body Temperature and Physical Activity”, *Proceedings of ARBIP95, 1995* [Madsen06] K.Madsen “Experimental Sensor Networks: Lessons from the Hogthrob Project”. Msc Thesis 06-04-02. U.Copenhagen. 2006

[Red05] G.Tolle et al. “A Macroscope in the Redwoods”. Sensys 2005.

[Shasha04] D. Shasha, Y.Zhu, “High Performance Discovery in Time Series: Techniques and Case Studies”, Springer, 2004.

[SITEX02] Marco F. Duarte and Yu Hen Hu, “Vehicle Classification in Distributed Sensor Networks”, Journal of Parallel and Distributed Computing, Vol. 64 No. 7, pp. 826-838, 2004.

[Sung04] M.Sung, R. DeVaul, S.Jimenez, J.Gips, A.Pentland. “Shiver Motion and Core Body Temperature Classification for Wearable Soldier Health Monitoring Systems”. ISWC 2004.

Chapter 3

Normalizing Resource Consumption

The defining characteristic for sensornets, and the mote platform in particular, have been the limited resources: processing power, battery life, storage, bandwidth, etc. These limitations have spawned a variety of systems, network protocols, and algorithms with the purpose of minimizing resource consumption.

However, these minimizations often come as the results of a trade-off between resource consumption and performance. In the presence of energy harvesting, the trade-off between resource consumption and performance can become particularly fluent because of the variations in the amount of harvested energy. In order to minimize resource requirements, while still keeping the desired performance level, several *resource aware* systems have been developed.

For example, the Pixie operating system by Lorincz et al. [4] keeps track of all resources and utilizes a broker to allocate these to applications. This allows a more fine grained control between minimizing resources and optimizing performance at the operating system level.

Another example is the solar powered system developed by Fan et al. [1]. They used a centralized scheme to adapt the sampling strategy based on the available stored energy and the expected harvested energy, under the conditions that the energy reserve should never be depleted while at the same time as much of the free harvested energy should be used, thereby maximizing use of the harvested energy.

The two main concerns for resource aware systems, such as those described above, are knowing (1) the available resources and (2) the resource requirements for each action. The first can be estimated either by direct measurement of the remaining voltage or indirectly by keeping records of resources already spent. For the second part, Fan et al. used datasheets to estimate resource consumption while Lorincz et al. assumed the information to be available from other sources.

In "Characterizing Mote Performance: A Vector-Based Methodology" we present a systematic way to normalize an application's energy and time consumption by splitting it into unit actions and measuring the energy and time consumed by each. Our method is based on collecting the application's traces when executing in the field while simultaneously measuring the energy consumption and elapsed time. This allows us to incorporate non-deterministic real-world interactions which simulations such as *PowerTOSSIM* by Shnayder et al. [6] and *AEON* by Landsiedel et al. [3] lack. The paper was presented at the *5th European conference on Wireless Sensor Network, 2008* and made in collaboration with Martin Leopold and Philippe Bonnet.

Our approach is similar to *Quanto* by Fonseca et al. [2] who also collects application traces, but instead of only tracking resource usage we also consider resource prediction. Using linear algebra and a vector-based approach, inspired by Seltzer et al. [5], we devised a method to divide the energy and time consumptions into two independent contributions, one specific to the application and one specific to the platform. By combining application vectors with platform vectors we are able to predict the energy and time consumption of one application on different platforms, and different applications on the same platform. Cross-platform prediction is essential when comparing different platforms in preparation for a deployment. In the Hogthrob deployments, this would have been extremely valuable since we would have been able to consider a wider range of motes for our second deployment without having to actually deploy them.

Application predictions can be used in resource aware applications such as those mentioned above. Since each element in the application vector represents an action, it is also possible to construct artificial application vectors using real ones as template.

Bibliography

[1] K.-W. Fan, Z. Zheng, and P. Sinha. Steady and Fair Rate Allocation for Rechargeable Sensors in Perpetual Sensor Networks. In *SenSys '08: Pro-*

ceedings of the 6th ACM conference on Embedded network sensor systems, pages 239–252, New York, NY, USA, 2008. ACM.

[2] R. Fonseca, P. Dutta, P. Levis, and I. Stoica. Quanto: Tracking energy in networked embedded systems. In R. Draves and R. van Renesse, editors, *OSDI*, pages 323–338. USENIX Association, 2008.

[3] O. Landsiedel, K. Wehrle, and S. Gotz. Accurate Prediction of Power Consumption in Sensor Networks. In *EmNets '05: Proceedings of the 2nd IEEE workshop on Embedded Networked Sensors*, pages 37–44, Washington, DC, USA, 2005. IEEE Computer Society.

[4] K. Lorincz, B.-r. Chen, J. Waterman, G. Werner-Allen, and M. Welsh. Resource Aware Programming in the Pixie OS. In *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 211–224, New York, NY, USA, 2008. ACM.

[5] M. Seltzer, D. Krinsky, K. Smith, and X. Zhang. The Case for Application-Specific Benchmarking. In *Hot Topics in Operating Systems, 1999. Proceedings of the Seventh Workshop on*, pages 102–107, 1999.

[6] V. Shnayder, M. Hempstead, B. rong Chen, G. Werner-Allen, and M. Welsh. Simulating the Power Consumption of Large-Scale Sensor Network Applications. In *Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys'04)*, Nov. 2004.

Characterizing Mote Performance: A Vector-Based Methodology

Martin Leopold, Marcus Chang, and Philippe Bonnet

Department of Computer Science,
University of Copenhagen,
Universitetsparken 1, 2100 Copenhagen, Denmark.
`{leopold,marcus,bonnet}@diku.dk`

Abstract. Sensors networks instrument the physical space using motes that run network embedded programs thus acquiring, processing, storing and transmitting sensor data. The motes commercially available today are large, costly and trade performance for flexibility and ease of programming. New generations of motes are promising to deliver significant improvements in terms of power consumption and price — in particular motes based on System-on-a-chip. The question is how do we compare mote performance? How to find out which mote is best suited for a given application? In this paper, we propose a vector-based methodology for benchmarking mote performance. Our method is based on the hypothesis that mote performance can be expressed as the scalar product of two vectors, one representing the mote characteristics, and the other representing the application characteristics. We implemented our approach in TinyOS 2.0 and we present the details of our implementation as well as the result of experiments obtained on commercial motes from Sensinode. We give a quantitative comparison of these motes, and predict the performance of a data acquisition application.

1 Introduction

Sensor networks-based monitoring applications range from simple data gathering, to complex Internet-based information systems. Either way, the physical space is instrumented with sensors extended with storage, computation and communication capabilities, the so-called motes. Motes run the network embedded programs that mainly sleep, and occasionally acquire, communicate, store and process data. In order to increase reliability and reduce complexity, research prototypes [1, 2] as well as commercial systems[1] now implement a tiered approach where motes run simple, standard data acquisition programs while complex services are implemented on gateways. These data acquisition programs are either a black box (Arch Rock), or the straightforward composition of building blocks such as sample, compress, store, route (Tenet). This approach increases reliability because the generic programs are carefully engineered, and reused across

[1] See http://www.archrock.com

deployments. This approach reduces complexity because a system integrator does not need to write embedded programs to deploy a sensor network application.

Such programs need to be portable to accommodate different types of motes. First, a program might need to be ported to successive generations of motes. Indeed, hardware designers continuously strive to develop new motes that are cheaper, and more power efficient. Second, a program might need to be ported simultaneously to different types of motes, as system integrators need various form factors or performance characteristics.

Handzicki, Polastre et al.[5] address the issue of portability when they designed TinyOS 2.0 Hardware Abstraction Architecture. They defined a general design principle, that introduces three layers:

1. Mote Hardware: a collection of interconnected hardware components (typically MCU, flash, sensors, radio).
2. Mote Drivers: Hardware-specific software that exports a hardware independent abstraction (e.g., TinyOS 2.0 define such Hardware Independent Layer for the typical components of a mote).
3. Cross-Platform Programs: the generic data acquisition programs that organize sampling, storage and communication.

We rely on these three layers to reason about mote performance. Whether motes are deployed for a limited period of time in the context of a specific application (e.g., a scientific experiment), or in the context of a permanent infrastructure (e.g., within a building), power consumption is the key performance metric. Motes should support data acquisition programs functionalities within a limited power budget. We focus on the following questions:

1. What mote hardware to pick for a given program? The problem is to explore the design space and choose the most appropriate hardware for a given program without having to actually benchmark the program on all candidate platforms.
2. What is a mote hardware good for? The problem is to characterize the type of program that is well supported by a given mote hardware.
3. Is a driver implemented efficiently on a given hardware? The problem is to conduct a sanity check to control that a program performs as expected on a given hardware.

We are facing these questions in the context of the Hogthrob project, where we design a data acquisition infrastructure. First, because of form factor and cost, we are considering a System-on-a-Chip (SoC) as mote hardware. Specifically, we want to investigate whether Sensinode Nano, a mote based on Chipcon's CC2430 SoC, would be appropriate for our application. More generally, we want to find out what a CC2430 mote is good for, i.e., what type of applications it supports or does not support well. Also, we had to rewrite all drivers to TinyOS 2.0 on CC2430, and we should check that our implementation performs as well as TinyOS 2.0 core. Finally, we would like to use Sensinode Micro as a prototyping platform for our application as its toolchain is easier and cheaper to use (see

Section 3.2 for details). We would like to run our application on the Micro, measure performance, and predict the performance we would get with the Nano.

In this paper, we propose a vector-based methodology to study mote performance. Our hypothesis is that energy consumption on a mote can be expressed as the scalar product of two performance vectors, one that characterize the mote (hardware and drivers), and one that characterize the cross-platform application. Using this methodology, we can compare motes or applications by comparing their performance vectors. We can also predict the performance of an application on a range of platforms using their performance vectors. This method will enable sensor network designers answer the questions posed above. Specifically, our contribution is the following:

1. We adapt the vector-based methodology, initially proposed by Seltzer et al.[4], to study mote performance in general and TinyOS-based motes in particular (Section 3).
2. We conduct experiments with two types of motes running TinyOS 2.0: Sensinode Micro and CC2430. We ported TinyOS to these platforms (see Section 4).
3. We present the results of our experiments (Section 5). First, we test the hypothesis underlying our approach. Second, we compare the performance of the Micro and CC2430 motes using their hardware vectors. Finally, we predict the performance of generic data acquisition programs from the Micro to the CC2430.

2 Related Work

Typically, analytical models, simulation or benchmarking are used to study the performance of a program [3]. In our opinion, simulation is best suited for reasoning about the performance and scalability of protocols and algorithms, not to reason about the performance of an application program on a given mote hardware. Indeed, simulators are best suited when they abstract the details of the hardware and driver layers. Standard benchmarks fall into two categories: application benchmarks (SPEC, TPC), or microbenchmarks (lmbench)[2]. There is no such standard benchmark for sensor networks. Micro benchmarks have been defined for embedded systems (EEMBC), but they focus at the automotive and consumer electronics markets – they do not tackle wireless networking or sensing issues.

The vector-based methodology proposed by Setlzer et al.[4] has been used to characterize the performance of web servers, OS utilities and Java Virtual Machines. Our paper is the first to propose this methodology in the context of sensor networks.

Performance estimation is of the essence for real-time embedded systems. The focus there is on timing analysis, not so much on energy consumption. We share a same goal of integrating performance estimation into system design [8].

[2] See http://www.tpc.org, http://www.spec.org, http://www.bitmover.com/lmbench, and http://www.eembc.org/ for details about these benchmarks.

In the context of sensor network, our work follows-up on the work of Jan Beutel that defined metrics for comparing motes[9]. Instead of using data sheets for comparing mote performance, we propose to conduct application-specific benchmarks.

Our work is a first step towards defining a cost model for applications running on motes. Such cost models are needed in architectures such as Tenet [1] or SwissQM [2] where a gateway decides how much processing motes are responsible for. Defining such a cost model is future work .

3 Vector-Based Methodology

The vector-based methodology[4], consists in expressing overall system performance as the scalar product of two vectors:

1. A system-characterization vector, which we call **mote vector** and denote $\underline{MV}$. Each component of this vector represents the performance of one primitive operation exported by the system, and is obtained by running an appropriate microbenchmark.
2. An application-characterization vector, which we call **application vector** and denote $\underline{AV}$. Each component of this vector represents the application's utilization of the corresponding system primitives, and is obtained by instrumenting the API to the system primitive operations.

Our hypothesis is that we can define those vectors such that mote performance can be expressed as their scalar product:

$$Energy = \underline{MV} \cdot \underline{AV}$$

Our challenge is to devise a methodology adapted to mote performance. The issues are (i) to define the mote vector components, and the microbenchmarks used to populate them, and (ii) to define a representative application workload, to collect a trace from the instrumented system API, and to convert an application trace into an application vector.

3.1 Mote Vector

We consider a system composed of the mote hardware together with the mote drivers. The primitive operations exported by such a system are:

- CPU duty cycling: the network embedded programs that mainly sleep and process events need to turn the CPU on and off[3].
- Peripheral units: controlled through the hardware-independent functions made available at the drivers interface.

[3] Note that we assume that the mote hardware relies on a single CPU to control all peripheral units. Peripheral units such as digital sensors might include their own micro-controller. Our assumption simply states that a mote program is run on a single CPU.

We choose this system because its interface is platform-independent. This has two positive consequences. First, we can use mote vectors to compare two different motes. Second, the application vector is platform-independent. We can thus use our vector-based methodology to predict the performance of an application across motes.

The mote vector components correspond to the CPU (when active or idle), and the peripheral units (as determined by the driver interfaces). Throughout the paper, we use an associative array notation to denote the mote (and application) vector components, e.g., $\underline{MV}[active]$ corresponds to CPU execution, $\underline{MV}[idle]$ corresponds to CPU sleep, $\underline{MV}[PUi]$, correspond to peripheral units primitives where PUi is for example ADC sample, flash read, flash write, flash erase, radio transmit, radio receive.

We need to define a metric for the vector components. The two candidates are energy and time. We actually need both: (a) energy to compute the scalar product with the application vector and thus obtain mote performance, and (b) time to derive the platform-independent characteristics of an application (see Section 3.2). We thus need to define a microbenchmark for each mote vector component for which we measure time elapsed and energy spent. We distinguish between the energy mote vector, noted $\underline{MV_e}$, and the time mote vector, noted $\underline{MV_t}$.

The microbenchmarks must capture the performance of the system's primitive operations. The first problem is to represent CPU performance. The most formidable task for the CPU in a sensor network application is to sleep. This is why we distinguish sleep mode from executing mode in the mote vector. For the applications we consider, a single sleep mode is sufficient. Defining a microbenchmark to define the energy spent in sleep mode is trivial. However, we wish to use the time mote vector to compare the time spent in sleep mode by different motes. Intuitively, the time spent in sleep mode is a complement of the time spent processing. As an approximation, we thus consider that $\underline{MV_t}[idle]$ is the complement of $\underline{MV_t}[active]$ with respect to an arbitrary time period (fixed for all mote vectors), and that $\underline{MV_e}[CPUsleep]$ corresponds to the energy spent in sleep mode during that time.

The second problem is to define an appropriate representation of CPU performance (in executing mode). Unlike peripheral units, for which drivers define a narrow-interface, the CPU has a rich instruction set. It is non-trivial to estimate the CPU resources used by a given application as it depends on the source code and on the way the compiler leverages the CPU instruction set. We choose a simple approach where we use a microbenchmark as a yardstick for the compute-intensive tasks of an application. We thus represent CPU performance using a single vector component. There is an obvious pitfall with this approach: we assume that the distribution of instructions used by the microbenchmark is representative of the instructions used by the application. This is unlikely to be the case. We use this simple approach, despite its limitation, as a baseline for our methodology because we do not expect CPU utilization to have a major impact on energy consumption. Our experiments constitute a first test of this assumption. Obviously much more tests are needed, and devising a more precise estimation of CPU utilization is future work.

The third problem related to the microbenchmarks is that driver interfaces often provide a wide range of parameters that affect their duration and energy consumption. Instead of attempting to model the complete range of parameters, we define microbenchmarks that fix a single set of parameters for each peripheral unit primitive. Each peripheral unit microbenchmark thus corresponds to calling a system primitive with a fixed set of parameters, e.g., a microbenchmark for radio transmit will send a packet of fixed length, and a microbenchmark for ADC sampling will sample once at a fixed resolution. We believe that this models the behavior of sensor network application that typically use a fixed radio packet length or a particular ADC resolution. This method can trivially be expanded by defining a vector component per parameters (e.g., replacing radio transmit with two components radio transmit at packet *length_1* and radio transmit at packet *length_2*).

For the sake of illustration, let us consider a simplistic mote with a subset of the TinyOS 2.0 drivers, that only exports two primitives: ADC sample and radio transmit (tx). The associated time mote vectors will be of the form:

$$\underline{MV_t} = \begin{bmatrix} active \\ idle \\ adc \\ tx \end{bmatrix}$$

Where the mote vector components correspond to the time spent by the mote running the CPU microbenchmark, to the time spent in sleep mode (the complement of the time spent running the CPU benchmark with respect to an arbitrary time period that we set to 20 s), to the time spent running the ADC benchmark, and to the time spent running the transmit benchmark.

In order to express mote performance as the scalar product of the energy mote vector and the application vector, we need the components of the mote vectors to be independent. This is an issue here, because CPU is involved whenever peripheral units are activated. Our solution is to factor CPU usage in each peripheral unit component. As a consequence, the mote vector component corresponding to CPU performance (*active*) must be obtained without interference from the peripheral units. Another consequence is that we need to separate the CPU utilization associated to peripheral units from the pure computation, when deriving the platform-independent characteristics of an application. We thus register CPU time when benchmarking each peripheral unit primitive. We denote them as $CPU[PUi]$ for each peripheral unit primitive PUi.

We detail in the next Section, how we use those measurements when deriving the application vector from a trace.

3.2 Application Vector

Our goal is to characterize how an application utilizes the primitives provided by the underlying system. The first issue is to define a workload that is representative of the application. In the context of sensor networks, workload characterization is complicated (i) because motes interact with the physical world and (ii) because the network load on a mote depends on its placement with respect to the gateway, and (iii) because different motes play different roles in the sensor network (e.g., in a multihop network a mote located near the gateway deals with more network traffic than a mote located at the periphery of the network).

We consider that a sensor network application can be divided into representative epochs that are repeated throughout the application lifetime. For example, the application we consider in the Hogthrob project consists of one data acquisition epoch[4], where an accelerometer is sampled at 4 Hz, the samples are compressed, stored on flash when a page is full, and transmitted to the gateway when the flash is half-full. While sampling is deterministic, such an epoch is non-deterministic as compressing, storing or transmitting depends on the data being collected, and on the transmission conditions. Obviously, tracing an application throughout several similar epochs will allow us to use statistics to characterize these non-deterministic variations.

For each epoch, we trace how the application uses the CPU and the peripheral units. More precisely the trace records the total time spent by the mote in each possible mote state, defined by the combination of active mote vector components (*active* that represents the compute-intensive operations, *idle* that represents the CPU in sleep mode, and *PUi* that represents a peripheral unit interface call). We thus represent the trace as a vector, denoted $\underline{T}$. $\underline{T}$ is of dimension 2^m, where m is the dimension of the mote vector. Some of the mote states will not be populated because they are mutually exclusive (e.g., *active* and *idle*), or because the driver interfaces prevent a given combination of active peripheral units.

Let us get back to the simple example we introduced in the previous section. The trace vector for an epoch will be of the form:

$$\underline{T} = \begin{bmatrix} active \\ idle \\ adc \\ tx \\ adc \ \& \ tx \\ active \ \& \ adc \\ active \ \& \ tx \\ active \ \& \ adc \ \& \ tx \\ \cdots \end{bmatrix}$$

Now the problem is to transform, for each epoch, the trace vector into a platform-independent application vector. The application vector, denoted $\underline{AV}$,

[4] A sensor network deployed for collaborative event detection will typically consist of two epochs: one where motes are sampling a sensor and looking for a given pattern in the local signal, and one where motes are communicating once a potential event has been detected.

has same dimension m as the mote vector, and each application vector component corresponds to the utilization of the system resource as modeled in the mote vector. The application vector components have no unit, they correspond to the ratio between the total time a system primitive is used in an epoch, by the time spent by this system primitive in the appropriate microbenchmark (as recorded in the time mote vector $\underline{MV_t}$). Note that if the driver primitive is deterministic, then the ratio between the total time spent calling this primitive in an epoch and the microbenchmarking time is equal to the number of times this primitive has been called. However, drivers typically introduce non-determinism,because the scheduler is involved or because drivers embed control loops with side effects (e.g., radio transmission control that results in retransmissions).

We use a linear transformation to map the trace vector onto the application vector. This transformation can be described in three steps:

1. We use an **architecture matrix** to map the trace into a vector of dimension m, the **raw total time vector**, where each component correspond to the total utilization of the CPU and peripheral units. The architecture matrix encodes the definition of each state as the combination of active mote vector components. Note that this combination depends on the architecture of the mote. For example, a SPI bus might be shared by the radio and the flash. In this case, the time spent in a state corresponding to radio transmission and flash write is spent either transmitting packets or writing on the flash (there is no overlap between these operations). We assume fair resource arbitration and consider that both components get half the time recorded in the trace. In case of overlap between operations, both get the total time recorded in the trace.
   In our simplistic example, assuming that a SPI resource is shared between the radio and the ADC, the architecture matrix will be of the form:
$$\mathbf{AM} = \begin{bmatrix} 1\,0\,0\,0\,0\,1\,1\,1 \\ 0\,1\,0\,0\,0\,0\,0\,0 \\ 0\,0\,1\,0\,\frac{1}{2}\,1\,0\,\frac{1}{2} \\ 0\,0\,0\,1\,\frac{1}{2}\,0\,1\,\frac{1}{2} \end{bmatrix}:$$
2. We use a **CPU matrix** to factor out of the *active* component the time spent by the CPU controlling the peripheral units. The CPU matrix, of dimension $m \times m$, is diagonal except for the column corresponding to the *active* component. This column is defined as 1 on the diagonal, 0 for the *idle* component, and $-\underline{CPU}[k]/\underline{MV}[k]$ for all other components. When multiplying the total time vector with the CPU matrix, we obtain a **total time** vector where the *active* component corresponds solely to the compute-intensive portion of the application.
   Using again our running example, we have a CPU matrix of the form:
$$\mathbf{CPU} = \begin{bmatrix} 1\,0 & -\frac{CPU[adc]}{\underline{MV_t}[adc]} & -\frac{CPU[tx]}{\underline{MV_t}[tx]} \\ 0\,1 & 0 & 0 \\ 0\,0 & 1 & 0 \\ 0\,0 & 0 & 1 \end{bmatrix}$$

3. We use the time mote vector to derive the application vector. The basic idea is to express the application utilization of the system primitive as the ratio between total time per component, and the time spent running a benchmark. We define the inverse mote vector, $\underline{MV^{-1}}$, as a vector of dimension m where each component is the inverse of the time mote vector component (this inverse is always defined as the time mote vector components are always non zero). We define the application vector as the Hadamard product of total time vector with the inverse mote vector.
   With our running example, we obtain the equation:
$$\begin{bmatrix} totalactive/\underline{MV_t}[active] \\ totalidle/\underline{MV_t}[idle] \\ totaladc/\underline{MV_t}[adc] \\ totaltx/\underline{MV_t}[tx] \end{bmatrix} = \begin{bmatrix} totalactive \\ totalidle \\ totaladc \\ totaltx \end{bmatrix} \circ \begin{bmatrix} 1/\underline{MV_t}[active] \\ 1/\underline{MV_t}[idle] \\ 1/\underline{MV_t}[adc] \\ 1/\underline{MV_t}[tx] \end{bmatrix}$$

More generally, we derive the application vector from the trace vector using the following linear transformation:

$$\underline{AV} = (\mathbf{CPU} \times (\mathbf{AM} \times \underline{T})) \circ \underline{MV^{-1}}$$

And we obtain the mote performance as the scalar product of the application vector with the energy mote vector:

$$E = \underline{AV} \cdot \underline{MV_e}$$

4 Implementation in TinyOS 2.0

We applied our vector-based methodology to two motes: Sensinode Micro, a Telos-like mote, and CC2430, which is the basis for a new generation of commercial motes[5]. We ported TinyOS 2.0 on both platforms.

4.1 CC2430 and Sensinode Micro

As a SoC Chipcon's CC2430[6] has a small form factor (7x7 mm) and promises to be mass-produced at a lower price than complex boards. Motes built around the CC2430 might constitute an important step towards reducing the price of sensor networks. The CC2430 is composed of the 8051 MCU with a wide range of common on-chip peripherals as well as an 802.15.4 radio very similar to the Texas Instruments CC2420. We run the system at 32 MHz. The CC2430 differs from the platforms on which TinyOS has been implemented so far in two important ways: the system architecture and the interconnect to the radio.

The Intel 8051 MCU architecture was designed in the early eighties and many oddities from the era remain. Not only is it an 8 bit, CISC style processor with a

[5] We experimented with a CC2430 development kit. Using commercial systems based on CC2430, such as Sensinode Nano, is future work.

[6] For details, see CC2430 data sheet: http://focus.ti.com/lit/ds/symlink/cc2430.pdf

Harvard architecture[7], but the main memory is further subdivided into separate address spaces that differ in size, are addressed differently and vary in access time. Simply put, the 8051 defines a fast memory area limited to 256 bytes, and a slow memory area of 8 KiB. In addition to variables, the fast access area contains the program stack. This limits the program stack to less than 256 bytes depending on the amount of variables in this area. Commonly, activation records of functions are placed on the stack, thus potentially limiting the call depth critically. To circumvent this problem, the compiler places stack frames in the slow data area, which imposes a high cost for storing and retrieving arguments that do not fit in registers when calling a function. The slow access RAM also penalizes dynamic memory allocation, and context switches and thus favor an event-based OS with static memory allocation such as TinyOS.

Because CC2430 is a SoC, there is no bus between the MCU and the radio. The MCU controls the radio via special function registers (instead of relying on a SPI bus as it is the case on Telos and Micro motes for example). The other peripheral units (ADC, UART, timers, flash, and pins) are accessed in the 8051 MCU as in other micro-controllers such as the MSP or Atmega.

The Sensinode Micro is built around the 16 bit, RISC style MSP430 MCU with combined code and memory spaces (Von Neuman). The platform can run up to 8 MHz, but we choose 1 MHz in our experiments. Apart from the built in common peripherals of the MSP, it features the Texas Instruments CC2420 radio which is connected though an SPI bus.

4.2 TinyOS 2.0 on CC2430 and Micro

TinyOS 2 has been designed to facilitate the portability of applications across platforms. First, it is built using the concept of components that use and provide interfaces. TinyOS is written in nesC, an extension of C that supports components and their composition. Second, TinyOS implements the Hardware Abstraction Architecture[5]. For each hardware resource, a driver is organized in three layers: the Hardware Presentation Layer (HPL) that directly exposes the functions of the hardware component as simple function calls, the Hardware Abstraction Layer (HAL) that abstracts the raw hardware interface into a higher-level but still platform dependent abstraction, and the Hardware Independent Layer (HIL) that exports a narrow, platform-independent interface. The TinyOS 2.0 core working group has defined HIL for the hardware resources of typical motes: radio, flash, timer, ADC, general IO pins, and UART.

Porting TinyOS 2.0 on CC2430 consisted in implementing these drivers[8]. For the timers, pins, UART and ADC we used the TinyOS HIL interfaces, however for the Radio and Flash diverge from the common interfaces.

Radio We export the radio using a straightforward *SimpleMac* interface. This interface is well suited for the 802.15.4 packet-based radios of the CC2430.

[7] Code and data are located in separate memory space

[8] For details, see http://www.tinyos8051wg.net

It allows to send and receive packets, and set various 802.15.4 parameters as well as duty cycling the radio. Note that we depart from the Active Message abstraction promoted by the TinyOS 2.0 core working group. Our SimpleMac implementation supports simple packet transmission, but does not provide routing, or retransmission. Implementing Active Messages is future work.

Flash We export the flash using the *SimpleFlash* interface that allows to read and write an array of bytes, as well as delete a page from flash. Note that this interface is much simpler than the abstractions promoted by the TinyOS 2.0 core working group (volumes, logging, large and small objects). We adopted this simple interface because it fits the needs of our data acquisition application. Implementing the core abstractions as defined in TEP103 is future work.

Timer The timers are exported using the generic TinyOS Timer interfaces *Alarm* and *Counter*. These two interfaces give applications access to hardware counters and allows the use of the TinyOS components to extend the timer width from 16 bit to 32 bit. Note that on the pre-release CC2430 chips we used for our experiments, timers do not work properly[9].

ADC The Analog-to-Digital Converter is accessed through the core *Read* interface that allows to read a single value. In order to read multiple values, an application must issue multiple read calls or use DMA transfers.

Pins The General IO pins are exported through the core *GeneralIO* interface, that allows to set or clear a pin, make it an input or an output.

UART The UART is exported using the core *SerialByteComm* interface (that sends and receives single bytes from the UART) and *StdOut* interfaces (that provides a `printf`-like abstraction on top of SerialByteComm.

Note that we did not need to change the system components from TinyOS 2.0. However, supporting a sleep mode on the CC2430 requires implementing a low-frequency timer. On the pre-release CC2430 chips we used for our experiments, timers do not work properly. This is work in progress, as a consequence our experiments are conducted without low-power mode on the CC2430.

The main challenges we faced implementing TinyOS 2.0 drivers on CC2430 were to (i) understand the TEP documents that describe the core interfaces as we were the first to port TinyOS 2.0 on a platform that was not part of the core, and (ii) to define an appropriate tool chain. Indeed, the code produced by the nesC pre-compiler is specific to gcc, which does not support 8051. We had to (a) choose another C compiler (Keil), and (b) introduce a C-to-C transformation step to map the C file that nesC outputs into a C file that Keil accepts as input (e.g., Keil does not support inlining, the definition of interrupt handlers is different in Keil and gcc, Keil introduces compiler hints that are specific to the 8051 memory model). The details of our toolchain are beyond the scope of this paper, see [6] for details.

Because the Micro has many similarities with the Telos mote, on which TinyOS 2.0 was originally developed, porting porting TinyOS 2.0 was a sim-

[9] The timers miss events once in a while. This error is documented on a ChipCon errata, which is not publically available.

ple exercise. However, the wiring of the radio does not feature all of the signals available on the Telos mote, meaning that the radio stack could not be reused. We implemented the simple MAC layer, *SimpleMac*, and simple flash layer *SimpleFlash* described above.

4.3 Mote Vectors and Benchmarks

The vector component are chosen by analyzing the components used by the applications. As a result, we choose the following components for their mote vectors: *active*, *idle*, *adc*, *radio_receive*, *radio_transmit*, $flash\_read$, $flash\_write$, and $flash\_erase$. Doing so, we leave some of the peripheral unit primitives out of the mote vector (e.g., the primitives to set or get the channel on the 802.15.4 radio) and unused peripherals. The time spent executing primitives left out are factored as CPU execution time, while the unused peripherals are only considered to contribute the idle power consumption. We also leave timers, UART and general IO pins out of the mote vector. The time spent in the timers is factored in the CPU idle component. We leave general IO pins out because we do not use LEDs, or digital sensors. Similarly, we do not use the UART. Note that we do not consider a specific sensor connected to the ADC.

The benchmarks we defined for these mote vector components are:

– A compression algorithm to characterize CPU execution. This component contains a mix of integer arithmetic with many loads and stores and some function calls. Using this algorithm is a baseline approach.
– Simple function calls with a fixed parameter for each peripheral unit primitive[10]. Note that benchmarks, in particular for the radio and flash, contain some buffer manipulation. These are measured as $CPU[PUi]$ (see Section 2.1).

4.4 TinyOS API Instrumentation

We need to implement the CPU and peripheral units to collect the traces that are the basis for the application vectors. We implemented the following mechanisms:

– For the peripheral units, we introduce a platform-independent layer between the component that provides the driver interface and the component that uses it. As an example consider reading a value from the ADC using the TinyOS 2.0 *Read* interface. This interface starts an ADC conversion with a *Read* command and returns with a *readDone*, We insert a layer that records the time elapsed between the *Read* command is called and the *readDone* event is received. This is obviously an approximation of the time during which the ADC is actually turned on.
– For the CPU, we leverage the fact that TinyOS has a simple task scheduler that puts the CPU into sleep mode when the task queue is empty. The microprocessor is awoken via interrupts generated from internal or external

[10] The source code is available through the TinyOS 2 contribution section

peripherals. We record the time elapsed between the CPU enters sleep mode and the woke-up interrupt handler is executed as *idle* and the rest of the time as *active*.

In order to collect this trace, we encode each state as a combination of bits (our mote vector is of dimension 8) we thus use 8 bits to encode the states. Collecting this trace could be done internally on the mote being investigated, but this introduces a management overhead. Instead we output each bit of the state as an IO pin, using a second mote, which we call *LogRecorder*, that records the state transitions. This mechanism is very similar to the monitoring techniques devised for deployment-support networks[7].

4.5 Data Acquisition Applications

We use simple data acquisition applications as workload for our experiments. We build them from building blocks: sample, compress, store, and send. We create 4 applications that increase the parallel behavior of these tasks from isolation to parallel sample and transmission:

SampleCompressStore is a simple state machine, that runs each step in isolation. As each sample is retrieved, it is then compressed, and once 10 samples are retrieved they are stored to flash. This cycle is repeated 9 times.

DataAcquisition extends the state machine from SampleCompressStore to retrieve the data from flash and transmit it. Again, each step in isolation.

SampleStoreForward is similar to DataAcquisition, except without the compression step.

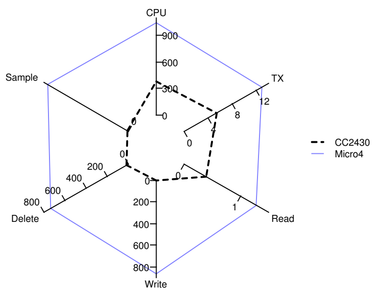
DataAcquisitionAdv performs the same tasks as DataAcquisition, but interleaves the sample and transmit processes. Store is done in isolation.

For our first experiments, we want a deterministic workload that exhibits reproducible results. One important source of variance in a sensor network applications is the environment. We choose a simple network topology and transmission scheme. Data is transmitted in 384 byte chunks (data and padding). The transmission does not expect acknowledgment that a packet is received, but only wait for the channel to be cleared (CCA) before sending. Sampling is at 10Hz and for compression we use the Lz77 algorithm.

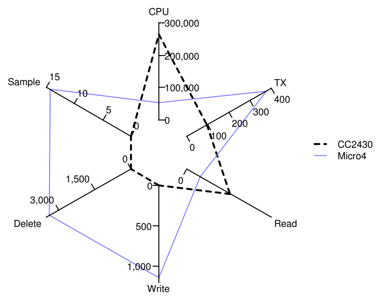
5 Experimental Results

5.1 CC2430 and Micro

We ran the benchmarks described in the previous section on both the Micro and CC2430 motes. The time and energy mote vectors we obtain are shown in Figure 1 as spider charts. The results are somewhat surprising. CC2430 is much faster than the Micro when running the benchmarks and transmitting packets. Slow memory accesses is compensated by the high clock rate and direct access to



(a) Time Mote Vectors



(b) Energy Mote Vectors

Fig. 1. Time and energy mote vectors for CC2430 and Micro

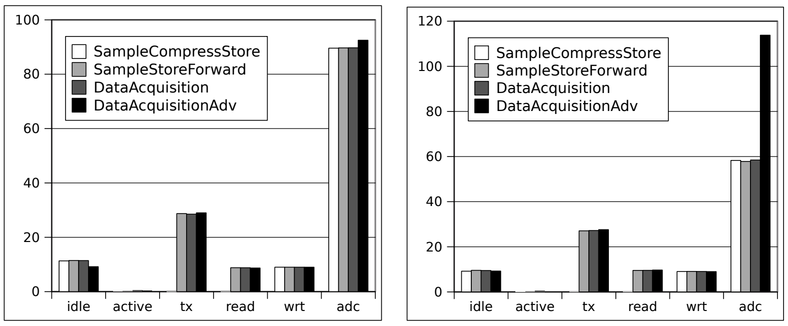
the radio speeds up packet transmission. It means that the CC2430 can complete its tasks quickly, and thus be aggressively duty cycled. In terms of energy, we observe that:

1. CPU operations are two to three orders of magnitude more expensive on the CC2430 than on the Micro. This is due to the high clock rate (which guarantees fast execution) and to the overhead introduced by the slow access RAM.
2. Flash operations are much more expensive on the Micro than on the CC2430. These results led us to check our driver implementation (which is a positive results in itself). We could not find any bug. We believe that the difference in performance can be explained by the difference in clock rate between both platforms (1 MHz for the Micro vs. 32 MHz for the CC2430) and with the fact that the CC2430 driver is hand coded in assembler and the Micro's is not.

5.2 Performance Prediction

We used our methodology to derive the application vectors for the four data acquisition applications described in the previous Section. The results are shown in Figure 2.

The profiles we get for the applications correspond to what we expect. Indeed, the application vector components for the ADC, flash and radio operations correspond roughly to the number of samples, flash and radio operations issued by the applications. The application vector is designed to be platform-independent. We thus expect that the application vectors derived from the CC2430 and Micro are similar. The good news is that they are at the exception of the ADC component. This is either a measurement error, a software bug in the driver, or a hardware



(a) Micro (b) CC2430

Fig. 2. Application vectors for CC2430 and Micro

bug. We focused on this issue and observed that the time it takes to obtain a sample on CC2430 varies depending on the application. Two different programs collecting the same data through the same ADC driver experience different sampling times. We observed as much as 50% difference between two programs. We believe that this is another hardware approximation on the CC2430.

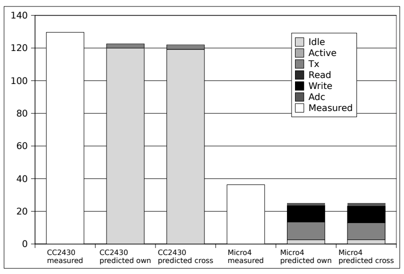


Fig. 3. Energy measurements and estimates

Our initial hypothesis is that the energy spent by an application on a mote can be estimated using the scalar product of the application vector with the mote vector. We computed the energy estimate for the *DataAcquisitionAdv* application and we compared them to the measurements we conducted directly on the motes (using an oscilloscope). The results are shown in Figure 3.

The estimations are well into an order of magnitude from the actual energy consumption. This is rather positive. As expected, the contribution from the CPU in active mode is insignificant. The poor performance of the CC2430 is due to the fact that we did not implement sleep mode support on the CC2430. Much more work is needed to test our methodology. This experiment, however, shows that we can use our method to prototype a data acquisition application with the Micro and predict how much energy the CC2430 would have used in the same conditions.

6 Conclusion

We described a vector-based methodology to characterize the performance of an application running on a given mote. Our approach is based on the hypothesis that mote energy consumption can be expressed as the scalar product of two vectors: one that characterize the performance of the core mote primitives, and one that characterizes the way an application utilizes these primitives. Our experiments show that our methodology can be used for predicting the performance of data acquisition applications between Sensinode Micro and a mote based on the CC2430 SoC. Much more experimental work is needed to establish the limits of our approach. Future work includes the instrumentation of an application deployed in the field in the context of the Snowths project, and the development of a cost model that a gateway can use to decide on how much processing should be pushed to a mote.

References

1. O. Gwanali, KY. Jang, J. Paek, M. Vieira, R. Govindan, B.Greenstein, A. Joki, D. Estrin, E.Kohler. The Tenet Architecture for Tiered Sensor Networks. *Proc ACM Intl. Conference on Embedded Networked Sensor Systems (Sensys 2006).*
2. Rene Mueller, Gustavo Alonso, Donald Kossmann. SwissQM: Next Generation Data Processing in Sensor Networks. *Third Biennial Conference on Innovative Data Systems Research.*
3. Victor Shnayder, Mark Hempstead, Bor-rong Chen, and Matt Welsh. Power-TOSSIM: Efficient Power Simulation for TinyOS Applications. *Proc ACM Intl. Conference on Embedded Networked Sensor Systems (Sensys 2004).*
4. Margo Seltzer, David Krinsky, Keith Smith, Xiaolan Zhang. The Case for Application-Specific Benchmarking. *Workshop on Hot Topics in Operating Systems 1999.*
5. Vlado Handziski, Joseph Polastre, Jan-Hinrich Hauer, Cory Sharp, Adam Wolisz and David Culler. Flexible Hardware Abstraction for Wireless Sensor Networks. *Proc. 2nd European Workshop on Wireless Sensor Networks (EWSN'07).*
6. Martin Leopold. Creating a New Platform for TinyOS 2.x *Technical Report 07/09, Depth. of Computer Science, University of Copenhagen, 2007*
7. J. Beutel, M. Dyer, M. Ycel and L. Thiele. Development and Test with the Deployment-Support Network. *Proc. 4th European Conference on Wireless Sensor Networks (EWSN 2007).*
8. L. Thiele and E. Wandeler. Performance Analysis of Distributed Embedded Systems. In *The Embedded Systems Handbook*. CRC Press, 2004.
9. J. Beutel. Metrics for Sensor Network Platforms. *Proc. ACM Workshop on Real-World Wireless Sensor Networks (REALWSN'06).*

Chapter 4

Anomaly Detection

In experimental science, consistency in the data collection process is key to a successful deployment. Equipment errors and faulty measurements must be detected quickly in order to avoid data loss. However, discovering faulty equipment is not trivial because faulty measurements can seem perfectly valid and pass through the sensornet the same way as valid data, and not be detected until post-processing.

A comprehensive list of measurement faults typically encountered in sensornets have been presented by Ni et al. [5]. Generally, measurement faults can be divided in two categories: permanent and temporary. For the permanent faults, all subsequent measurements from the same sensor or mote will result in faulty measurements and the only remedy is to replace the defective component. For temporary faults subsequent measurements from the same sensor and mote might be completely valid. Depending on the frequency of these faults, faulty measurements can be remedied by immediately resampling the sensor once the error has been detected.

The ability to collect consistent data thus depends on how quickly missing or faulty measurements can be identified and how fast this can be remedied. However, not all sensornets allow a swift response. Especially gateway based sensornets with a high duty-cycling or bad network connectivity are particularly vulnerable.

For example, in the *Life Under Your Feet* project, by Musăloiu-E. et al. [4], a sensornet collects temperature and moisture measurements, stores them locally, and periodically offloads them to a gateway. This sensornet is highly duty-cycled, with motes being out of contact with the gateway for weeks. Hence, the time to detect faulty measurements can be of this order of magnitude, and resampling faulty sensors is not an option even though the fault might have been temporary.

The zebra monitoring project *ZebraNet* by Huang et al. [3] was also exposed to this flaw. However, in this sensornet the latency was due to missing network connectivity because of the zebras' movement and not duty-cycling.

In order to reduce the impact of temporary faulty equipment it is necessary to detect faults as quickly as possible. Preferably, measurements should be checked right after they are sampled on the mote and a *watchdog* should ensure that faulty measurements are resampled immediately, in case they are temporary. Otherwise, an alert should be raised in order for the defect component to be replaced.

The question is, how do we detect faulty measurements on the mote? In "Mote-Based Online Anomaly Detection Using Echo State Networks" we present a machine learning based data processing framework, where measurements that deviate significantly from a training set is characterized as anomalies. This framework is based on a Neural Network variant, small and lightweight enough to be implemented on a mote. The paper was presented at the *5th IEEE International Conference on Distributed Computing in Sensor Systems, 2009.*

We subsequently augmented this framework with a Bayesian Network based machine learning algorithm, which proved to be superior in terms of execution speed and storage requirements. We present our findings in "Mote-Based Anomaly Detection" which has been submitted to the *ACM Transactions on Sensor Networks* journal and is currently under review. Both papers have been made in collaboration with Andreas Terzis and Philippe Bonnet.

This approach of combining faults and events into a unified anomaly detection framework is different from detection techniques previously used in sensornets, such as those presented by Greenstein et al. [1] and Sharma et al. [7], which rely on a set of heuristics tailored to specific faults and events.

Although these heuristics are efficient in terms of processing speed and storage requirements, and effective on their respective fault or event, Gupchup et al. [2] pointed out that they are also susceptible to misclassification as well, where faults are misclassified as events and vice versa. We avoid this misclassification problem by basing our framework on machine learning algorithms, which have the ability to generalize over the training data and classify everything else as anomalies.

The caveat of using machine learning is the availability of training data. In deployments where historic data is available training sets can easily be constructed, however, in new deployments where measurements have yet to be obtained the options are limited. In this case, one have to settle with artificial training sets based on models, bootstrapping the training during the deployment, or using human assisted learning similar to *Suelo* by Ramanathan et al. [6].

In the Hogthrob deployments, such a watchdog could have been used to minimize data loss due to the storage being full. This could have been done by defining inactivity periods caused by the sow sleeping to be anomalous and let the watchdog action be not to store this data.

Bibliography

[1] B. Greenstein, C. Mar, A. Pesterev, S. Farshchi, E. Kohler, J. Judy, and D. Estrin. Capturing high-frequency phenomena using a bandwidth-limited sensor network. In *SenSys '06: Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 279–292, New York, NY, USA, 2006. ACM.

[2] J. Gupchup, A. Sharma, A. Terzis, R. Burns, and A. Szalay. The Perils of Detecting Measurement Faults in Environmental Monitoring Networks. In *DCOSS*, 2008.

[3] P. Huang, H. Oki, Y. Wang, M. Martonosi, L. Peh, and D. Rubenstein. Energy-efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet. In *Proceedings of the Tenth Internationla Conferece on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X)*, Oct. 2002.

[4] R. Musăloiu-E., A. Terzis, K. Szlavecz, A. Szalay, J. Cogan, and J. Gray. Life Under your Feet: A Wireless Soil Ecology Sensor Network. In *Em-Nets Workshop*, May 2006.

[5] K. Ni, N. Ramanathan, M. N. H. Chehade, L. Balzano, S. Nair, S. Zahedi, E. Kohler, G. Pottie, M. Hansen, and M. Srivastava. Sensor Network Data Fault Types. *ACM Trans. Sen. Netw.*, 5(3):1–29, 2009.

[6] N. Ramanathan, T. Schoellhammer, E. Kohler, K. Whitehouse, T. Harmon, and D. Estrin. Suelo: Human-assisted Sensing for Exploratory Soil Monitoring Studies. In *SenSys '09: Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, pages 197–210, New York, NY, USA, 2009. ACM.

[7] A. Sharma, L. Golubchik, and R. Govindan. On the Prevalence of Sensor Faults in Real-World Deployments. In *IEEE SECON*, pages 213–222, 2007.

Mote-based Online Anomaly Detection using Echo State Networks

Marcus Chang[1], Andreas Terzis[2], and Philippe Bonnet[1]

[1] Dept. of Computer Science, University of Copenhagen, Copenhagen, Denmark
[2] Dept. of Computer Science, Johns Hopkins University, Baltimore MD, USA

Abstract. Sensor networks deployed for scientific data acquisition must inspect measurements for faults and events of interest. Doing so is crucial to ensure the relevance and correctness of the collected data. In this work we unify fault and event detection under a general *anomaly detection* framework. We use machine learning techniques to classify measurements that resemble a training set as *normal* and measurements that significantly deviate from that set as *anomalies*. Furthermore, we aim at an anomaly detection framework that can be implemented on motes, thereby allowing them to continue collecting scientifically-relevant data even in the absence of network connectivity. The general consensus thus far has been that learning-based techniques are too resource intensive to be implemented on mote-class devices. In this paper, we challenge this belief. We implement an anomaly detection algorithm using Echo State Networks (ESN), a family of sparse neural networks, on a mote-class device and show that its accuracy is comparable to a PC-based implementation. Furthermore, we show that ESNs detect more faults and have fewer false positives than rule-based fault detection mechanisms. More importantly, while rule-based fault detection algorithms generate false negatives and misclassify events as faults, ESNs are *general*, correctly identifying a wide variety of anomalies.

Keywords: Anomaly detection, Real-time, Wireless Sensor Networks

1 Introduction

Sensor networks deployed to collect scientific data (e.g., [1–3]) have shown that field measurements are plagued with measurement faults. These faults must be detected to prevent pollution of the experiment and waste of network resources. At the same time, networks should autonomously adapt to sensed events, for example by increasing their sampling rate or raising alarms. Events in this context are measurements that deviate from "normal" data patterns, yet they represent features of the underlying phenomenon. One such example would be rain events in the case of soil moisture.

The problem is that algorithms which classify measurements that deviate from the recent past as faulty tend to misclassify *events* as faults [4]. This behavior is undesirable because, unlike faults which must be discarded, events are the most important data that a mote collects, as they inform scientists about the characteristics of the observed environment. Furthermore, detection algorithms tailored to specific types of faults lead to false positives when exposed to multiple types of faults [4].

In this work we unify fault and event detection under a more general *anomaly detection* framework, in which online algorithms classify measurements that significantly deviate from a learned model of data as anomalies. By including punctuated, yet infrequent events in the training set we avoid the misclassification problem mentioned above thus allowing the system to distinguish faults from events of interest. More importantly, this learning-based technique can effectively detect measurement sequences that contain multiple categories of anomalies that do not exist in the training data.

Obviously anomaly detection can and should also be done on a gateway that correlates data from multiple sensors. Nonetheless, we claim that online detection on motes is also very much relevant. We motivate this need through an example derived from one of our ongoing projects [5]. Consider a set of motes deployed under the surface of a lake with limited physical access. These motes are connected to a floating buoy via acoustic modems which can be non-functional over long periods of time, either because the buoy is out of communication range or due to background noise in the lake. The motes should be able to autonomously alter their sensing behavior depending on whether the collected measurements are seemingly faulty or correspond to interesting events. For example, faulty measurements should be replaced in a timely manner by new measurements while interesting events should trigger sampling rate increases.

In summary, the contributions of this paper are as follows: **(1)** we develop an anomaly detection framework based on the Echo State Network (ESN) [6]. **(2)** We implement this framework on a mote-class device. **(3)** We quantitatively compare the ESN with two rule-based fault detection techniques. Specifically, we show that an ESN small enough to function alongside a fully-functional environmental monitoring mote application, is still more sensitive to subtler faults and generates fewer false positives than the two rule-based fault detection techniques.

2 Related Work

Anomaly characterization and detection has received significant attention in the sensor network community, yielding a broad range of algorithmic approaches. Probabilistic Principal Component Analysis [7], geometric algorithms [8], and Support Vector Machines [9], detect anomalies by partitioning data into subsets and subsequently identifying outliers. However, the temporal relations among data points are lost in such partitioning. We seek a solution that not only considers each data point in isolation, but also the context in which it appears.

Rashidi et al. recast the problem above as a pattern recognition problem and built a framework for pattern mining and detection [10], while Röemer used conditional rules to define anomalies [11] . However, neither of these solutions operate directly on raw measurements. Rather they rely on simple rules and thresholds to annotate the data with descriptive labels. The accuracy of both methods thereby depends on those labeling algorithms.

Sensor networks have extensively used rule- and threshold-based anomaly detection schemes due to their simplicity. For example, Werner-Allen et al. used threshold rules over Exponentially Weighted Moving Averages (EWMA) to detect seismological events [12], while analogous threshold techniques have been used to detect Cane toads [13] and vehicles [14]. In the context of environmental monitoring, Sharma et al. proposed two rules to detect faults commonly observed by such applications: Short faults, defined as drastic differences between two sequential measurements, and Noise faults, defined as periods during which measurements exhibit larger than normal variations [15]. To detect the former, the Short rule compares two adjacent data points and classifies the more recent as faulty when the difference is above a certain threshold. To detect the latter, the Noise rule considers a sliding window of measurements and flags all measurements in the window as faulty if the standard deviation is above a certain threshold. While these detection schemes are very resource efficient, their effectiveness is limited. For example, Werner-Allen et al. estimated the accuracy of their detection technique to be as low as 5%-29% [12]. Moreover, these schemes also suffer from inherent misclassification problems [4]. We thus seek a solution based on machine learning.

The use of machine learning as an anomaly detection tool has been proposed in the context of WSNs. For example, Echo State (neural) Networks [16] and Bayesian Networks [17] have been proposed for offline gas monitoring, while Kalman filters have been used for offline sow monitoring [18]. Bokareva and Bulusu used a Competitive Learning Neural Network (CLNN) for online classification [19]. However, the neural network was implemented on a Stargate gateway rather than a mote-class device. We bridge the gap between online detection and machine learning by implementing a learning-based technique on a mote.

3 Machine Learning

We propose a classification mechanism that accepts measurements matching a model as valid and rejects everything else as *anomalies*, where we define anomalies as the measurements that significantly deviate from learned data. We rely on machine learning techniques to define our classification model and focus on supervised learning because our scientific partners are able to provide training sets that correspond to the data they expect. Such data sets include previously collected measurements and synthetic data generated by analytical models.

What learning technique should we choose to achieve both accurate anomaly detection and efficient implementation on a mote-class device? We rule out Kalman filters, because they base each prediction on a sliding window of observed values instead of a compact model of learned data. Likewise, a Bayesian network's graph reduction operation (which is NP-complete) and the modeling of probability functions (typically Gaussians), discourage its use on resource constrained devices. Consequently, we decided to use ESN to meet our requirements in terms of classification efficiency (i.e., minimize false classifications) and resource use (i.e., minimize CPU, RAM, ROM, and energy usage).

3.1 Neural Networks

A neural network can be informally considered as an approximation function. Specifically, when presented with a subset of the original function's value pairs during the *training* stage, the neural network generalizes over these data and approximates the outcome of the original function in the *prediction* stage. Formally, a neural network is a weighted directed graph where each vertex represents a neuron. We consider discrete-time networks consisting of K input neurons, N hidden neurons, and L output neurons. The input neurons act as sources and the output neurons as sinks. The value of neuron j is given by: $v_j = \mathrm{A}(\sum w_{ij} v_i)$, where v_i is the output of neuron i, w_{ij} is the weight of the edge connecting neuron i to j, and A() is the *activation function.* This function is typically tanh() or a similar function. The training stage consists of adjusting the network's weights to approximate its output signal to the training signal.

Echo State Networks. In an ESN, all neurons are interconnected (but can have zero-weighted edges) meaning cycles involving one or more neurons are allowed. This gives each neuron the capability to remember, adding memory to the network as a whole. All the neurons' connections, directions, and weights are generated randomly and do not change, except for the output weights which are changed during training. The neurons thus act as a black box referred to as the Dynamic Reservoir (DR). This property reduces the learning algorithm to a simple linear regression. According to the Echo State property [6], the DR contains a set of basis states and by adjusting the output weights it is possible to capture the 'echoes' of real states as linear combinations of these basis states. Although the DR is randomly generated, Jaeger proved that it is possible to ensure that the DR indeed has the Echo State property by enforcing certain conditions [6] . One such condition is that the DR must be sparsely connected, i.e., 10% of all possible connections are actually active.

Anomaly Detection. We use ESNs to determine whether sensor readings are anomalous by comparing the ESN predictions to the actual measurements. In order to quantify the prediction error we look at the absolute differences between the measurements ($\boldsymbol{M}$) and the predictions ($\boldsymbol{P}$), i.e., $\boldsymbol{\delta} = |\boldsymbol{M} - \boldsymbol{P}|$. This difference should ideally be close to zero for *normal* data, while *anomalous* data should result in large differences (peaks). In other words, the ESN transforms the original time series into one whose values are ~ 0 most of the time, corresponding to the expected data. Anomaly detection thus reduces to recognizing the peaks in the transformed signal. We can then use pattern matching algorithms based on simple thresholds that have been proven to be both efficient and effective for such simple signals.

3.2 Discussion

The decoupling of the DR from the output weights enables several optimizations that fit WSNs particularly well. For example, the same DR can be used for multiple tasks by storing task-specific output weights and post-deployment updating can be done without transmitting the entire DR. Also, the requirement that the DR is sparsely connected, combined with the use of sparse matrix

	$\boldsymbol{\delta}_{lab/tanh}$ (%)	$\boldsymbol{\delta}_{mote/tanh}$ (%)	$\boldsymbol{\delta}_{lab/tl}$ (%)	$\boldsymbol{\delta}_{mote/tl}$ (%)
NRMSD	0.15448	0.15470	0.5855014	0.5855008

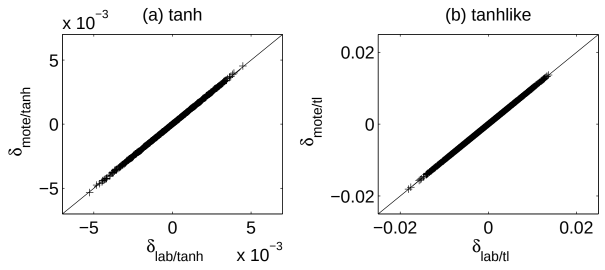


Fig. 1. (a) Q-Q plot of $\boldsymbol{\delta}_{lab/tanh}$ and $\boldsymbol{\delta}_{mote/tanh}$. (b) Q-Q plot of $\boldsymbol{\delta}_{lab/tl}$ and $\boldsymbol{\delta}_{mote/tl}$.

algebra, allows the implementation of ESNs that are larger than regular Feed Forward networks.

A limitation that ESNs share with all learning algorithms is their dependence on training data. In particular, if these data do not represent what domain scientists deem as “normal”, the predictions will be useless. Therefore, the choice of training sets, and more interestingly the choice of classification technique based on the available training sets is a very interesting open problem, which is beyond the scope of this paper. We just note that a key issue in successfully deploying an ESN lies in the choice and availability of training data. For example, adjusting the sampling rate in an adaptive sampling environment can change the properties of the measurement time series and thus possibly invalidate the training set. This issue can however be remedied, by storing different output weights for each sampling rate, or by disregarding higher sampling rates when applying the ESN detection. On the positive side, ESNs have the ability to generalize over the training data. In other words, ESNs base their predictions on the *trends* of the presented data rather than *exact* values. This feature allows motes deployed in similar regions to share the same training data instead of requiring mote-specific training sets.

4 ESN on a Mote

4.1 Implementation

While we create and train the ESNs offline, a complete ESN (including the network’s activation function, output weights, and the DR) is included in the application that runs on the mote. We use TinyOS 2.x to ensure portability to a broad range of mote class devices. Our implementation, publicly available for download at [20], focuses on feasibility and efficiency: the ESN must be able to fit in memory and the algorithm must be fast enough to maintain the desired sampling rate. We present the following three optimizations to improve performance along these two axes.

Sparse Matrix Algebra. The size of the DR’s weight matrix grows quadratically with the number of neurons in the reservoir n. However, only 10% of these

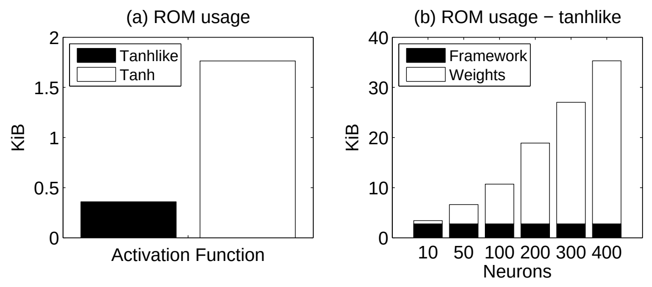


Fig. 2. (a) ROM footprints for the tanh() and *tanhlike* functions. (b) Total ROM footprint for an ESN using the custom *tanhlike* activation function.

elements are non-zero because the DR must possess the Echo State property. We leverage this feature by storing the matrix using Compressed Row Storage [21], which only stores the non-zero elements and the layout of the matrix. This reduces the necessary storage from $O(n^2)$ to $O(2n_z + n + 1)$, where n_z is the number of non-zero elements. This technique also reduces the number of operations needed to perform matrix multiplications by a similar factor since only non-zero elements are considered.

Single Floating Point Precision. Most mote-class devices rely on software emulated floating point operations due to lack of dedicated hardware. This contributes to both the storage and runtime overheads. At the cost of reduced floating point precision we select to store and compute all values using single instead of double floating point precision. Doing so halves the size of all the weight matrices and reduces the number of emulated floating point operations needed. As we later show, the resulting loss of precision is tolerable.

Tanhlike Activation Function. Because the activation function has to be applied to all the neurons in every iteration, it is important to choose an efficient function. At the same time, choosing a suboptimal activation function can significantly degrade the ESN's output quality. The algorithm for the often used hyperbolic tangent, tanh(), has high complexity requiring both large amounts of storage and a significant processing time. Because of these shortcomings, [22] proposed the approximate function:

$$TL(x) = \text{sign}(x)\left[1 + \frac{1}{2^{\lfloor 2^n |x| \rfloor}}\left(\frac{2^n |x| - \lfloor 2^n |x| \rfloor}{2} - 1\right)\right]$$

where $n \in \mathbb{Z}$ determines the steepness of the function. This *tanhlike* function has properties similar to tanh() (when $n = 1$) but with far lower complexity. However, it is also a non-differentiable, piecewise-linear function because of the rounding operations ($\lfloor \cdot \rfloor$). Therefore, we expect the quality of the ESN's output to be lower than when using tanh(), because small changes in input will result in large changes in output if these changes happen across a linear junction.

4.2 Evaluation

We verify that our ESN implementation indeed performs well on a mote-class device by comparing its output to a reference ESN running on a PC. We consider

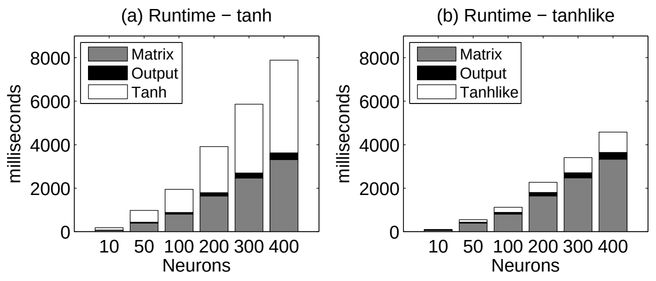


Fig. 3. Total execution cost of one ESN iteration divided to three components. (a) using the GCC built-in tanh() activation function. (b) using the custom tanhlike activation function.

ESNs which consist of two input signals (with one of the input signals held at a constant bias value in order to improve performance [23]), a 10-400 neuron reservoir, and one output signal (i.e., $K = 2, N = 10 - 400$, and $L = 1$). All mote experiments are carried out on a TelosB mote [24], running TinyOS 2.x with the clock frequency set to the default speed of 4 MHz [25]. Data sets are stored in ROM with measurements read with a fixed frequency to simulate sensor sampling. We use Matlab R2007a with the *Matlab toolbox for ESNs* [26] as our reference implementation. We use the Mackey-Glass (MG) time series with a delay $\tau = 17$ [27] to evaluate our ESN implementation. This system is commonly used to benchmark time series prediction methods because of its chaotic nature.

Sanity Check. We created a MG time series with 4,000 samples and used the first 2,000 samples to train a 50 neuron ESN, the next 1,000 samples for initialization, while the last 1,000 samples were used as the prediction vector $\boldsymbol{MG}$. Both the tanh() and *tanhlike* activation functions were used resulting in four different predictions: $\boldsymbol{P}_{lab/tanh}$, $\boldsymbol{P}_{mote/tanh}$, $\boldsymbol{P}_{lab/tl}$, and $\boldsymbol{P}_{mote/tl}$. We compute the four prediction errors and normalized root-mean-squared deviations (NRMSD).

Figure 1 presents the Q-Q plots [28] of the prediction errors grouped by activation function. Since the NRMSDs from the same activation function are almost identical and the points in the Q-Q plots lie on a straight line with slope one, we conclude that the TelosB ESN implementation has the same accuracy as the one in Matlab. Also, with an NRMSD less than 1% we see that the 50-neuron ESN is indeed capable of tracking the MG time series. However, the choice of activation function has a significant impact on the accuracy of the predictions, with tanh() being four times more accurate than the *tanhlike* function. This supports our claim that the piecewise-linearity of the *tanhlike* function degrades performance.

In order to compare the double precision floating point in Matlab with that of the single precision floating point on the TelosB, we look at the differences between predictions from the former with the latter when using the same activation function, i.e., $\boldsymbol{\delta}_{tanh} = \boldsymbol{P}_{lab/tanh} - \boldsymbol{P}_{mote/tanh}$ and $\boldsymbol{\delta}_{tl} = \boldsymbol{P}_{lab/tl} - \boldsymbol{P}_{mote/tl}$. We compute the NRMSDs for both error distributions: $NRMSD(\boldsymbol{\delta}_{tanh}) = 6.6 \cdot 10^{-3}$ % and $NRMSD(\boldsymbol{\delta}_{tl}) = 1.3 \cdot 10^{-4}$ %. Because $NRMSD(\boldsymbol{\delta}_{tanh}) < NRMSD(\boldsymbol{\delta}_{lab/tanh})$

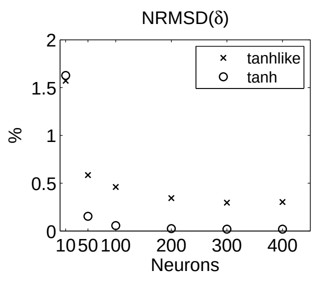


Fig. 4. *NRMSD*($\boldsymbol{\delta}$) for different reservoir sizes and activation functions.

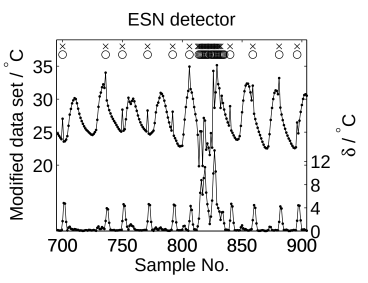


Fig. 5. Relation between measurements (middle plot), prediction errors (bottom plot), and injected/detected anomalies (top X/O markers).

and $NRMSD(\boldsymbol{\delta}_{tl}) < NRMSD(\boldsymbol{\delta}_{lab/tl})$ the errors caused by using single precision floating point are smaller than the errors caused by the ESN predictions. Thus, using single precision floating point on the TelosB is sufficient.

Performance. In order to explore the implementation's characteristics, such as ROM footprint, runtime speed, and accuracy, we vary the number of neurons in the DR. The ROM usage can be divided into two components: (1) *Framework*, the ESN algorithm used for prediction. (2) *Weight Matrices*, the DR and output weights. Whereas (1) is constant, (2) depends on the number of neurons in the reservoir. Figure 2a presents the ROM size difference for the two activation functions and Figure 2b shows the ROM footprint of the aforementioned components (using *tanhlike*). We observe that the memory contribution from the reservoir grows linearly, confirming the storage requirement of the Compressed Row Storage ($O(2n_z+n+1)$). Also, the ROM footprint is 1,806 bytes for tanh() and 368 bytes for *tanhlike*, making the former five times larger than the latter.

Next we measure the runtime cost of the ESN implementation. For each iteration, the ESN prediction algorithm performs the following set of operations: (1) *Matrix*, matrix-vector multiplication. (2) *Activation Function*, application of the activation function. (3) *Output*, vector-vector multiplication. Figure 3 summarizes the execution time of one prediction step and the contributions from each of the three operations. Surprisingly, the tanh() activation function is the most expensive operation and not the matrix-vector multiplication. It takes 28% longer to run than the matrix-vector multiplication and 453% longer than the *tanhlike* activation function.

Finally, we look at the prediction error as a function of reservoir size and activation function. We compare against the MG time series and find the *NRMSD*($\boldsymbol{\delta}$) for the six reservoirs and two activation functions used above. Figure 4 presents the results of this comparison. As expected, the prediction error decreases as the reservoir size increases and the *tanh*() activation function leads to more accurate predictions in general. Upon closer inspection, there appear to be three distinct regions relative to the reservoir size: small (10 neurons), medium (50-300 neurons), and large (300-400 neurons). In the small region, the prediction error is dominated by the small size of the reservoir and the choice of activation function

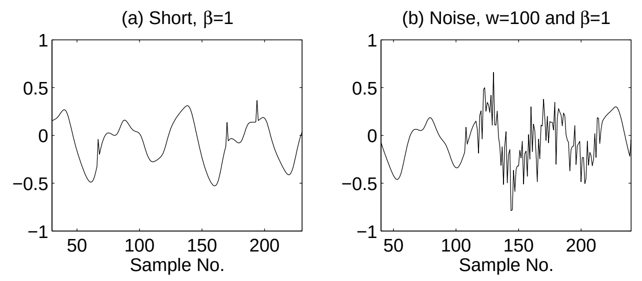


Fig. 6. Two types of injected anomalies: (a) Short faults and (b) Noise faults.

becomes less important. In the medium region there is a diminishing, yet clear reduction of the prediction error as the reservoir size increases. Finally, in the large region the prediction error does not decrease by adding neurons to the reservoir. Interestingly, the largest contribution to the prediction error comes from the activation function, with no overlap of prediction errors for the 50-400 neuron reservoirs. In fact, even the 50 neuron tanh() reservoir outperforms the 400 neuron *tanhlike* reservoir.

5 Evaluation

5.1 Experimental Design

The results from the previous section suggest that an ESN can be accurate, small, and fast enough to be incorporated with an existing data collection application that has been actively deployed for the past three years [1]. Motes in these sensor networks collect soil temperature and soil moisture readings every 20 minutes and store them to their onboard flash memory. All measurements are periodically offloaded over the network and persistently stored in a database. This environmental sensing application uses 40,824 bytes of ROM and 3,928 bytes of RAM, leaving 8,328 bytes of available ROM and 6,312 bytes of free RAM on the TelosB. From the previous section we know that a 50-neuron ESN using the *tanhlike* activation function has a ROM footprint of 6,788 bytes and a prediction time of 572 ms for each measurement. Thereby such an ESN complies with both the storage and computation constraints of the application and will be used for the remainder of this section.

Anomaly Types. We focus on two types of random anomalies: Short and Noise. These were defined in [15] and presented in Section 2. Samples of these faults can be seen in Figure 6. For brevity we only present these two random anomalies; for the detection of a systematic anomaly and further results we refer to our technical report [29]. For Short anomalies, we use two parameters to control their injection: the sample error rate and the amplification factor, β. For each anomalous measurement, $\tilde{m}_i$, we multiply the standard deviation of the original signal, σ, with β to obtain: $\tilde{m}_i = m_i + \beta\sigma$, where m_i is the true measurement.

For Noise anomalies, we use three parameters: the sample error rate, the period length, w, and the amplification factor, β. For each noisy period, we

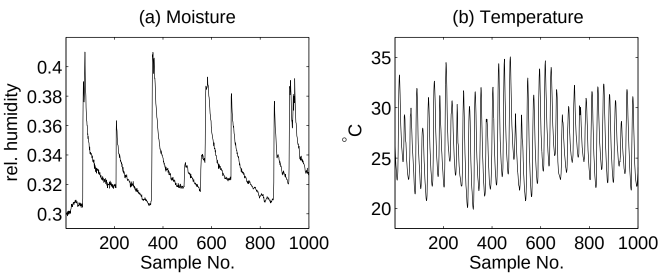


Fig. 7. Environmental sensing data sets. (a) Soil moisture and (b) Soil temperature.

calculate the standard deviation of the underlying signal and multiply it with β to create a random normal distribution with zero mean and $\beta\sigma$ standard deviation (i.e., $N(0, \beta\sigma)$). We then add samples from this distribution to each of the true measurements within that period.

Detection Algorithms. We use the two rule-based anomaly detection algorithms defined by [15] and summarized in Section 2 to detect the two anomalies mentioned above. We use these algorithms as reference as they are directly related to the anomalies we inject and their complexity is comparable to that of currently deployed fault detection algorithms. Our strategy for setting the thresholds is to minimize the number of false positives when the detection algorithms are applied to data sets with no anomalies.

Data Sets. For each of the soil moisture and soil temperature modalities that we use, we obtain a training and a test data set from the experiment's database [1]. Each of the four data sets consists of 1,000 data points. Figure 7 illustrates two such data sets. The data has been automatically sanitized by the database as a standard procedure for removing anomalies, following the methods proposed by [15]. By using this preprocessed data (instead of raw data) our results will not be biased by any anomalies already present in the data stream. Instead, we can assume that the only anomalies in the data are the ones we explicitly inject, thereby establishing the ground truth for evaluation purposes.

5.2 Results

Figure 5 illustrates the operation of the ESN anomaly detection algorithm by presenting the relation between the injected anomalies (Short $\beta = 1$; Noise $\beta = 1$ and $w = 20$), the temperature measurements – including the artificially added anomalies, the prediction error $\boldsymbol{\delta}$, and the detected anomalies. Notice that the prediction error is indeed an almost constant signal overlaid with large peaks coinciding with the injected faults. When not injected with anomalies we find that $NRMSD(\boldsymbol{\delta}_{Temp}) = 2.4\%$ and $NRMSD(\boldsymbol{\delta}_{Moist}) = 4.4\%$ for the temperature and moisture data set respectively. This accuracy is of the same order of magnitude as the one [16] found when tracking gas measurements, meaning that our online implementation is indeed comparable to the offline counterpart.

We use a 5% sample error rate (i.e., 5% of the measurements are polluted with errors) for each fault type and a period $w = 10$ for Noise faults. The

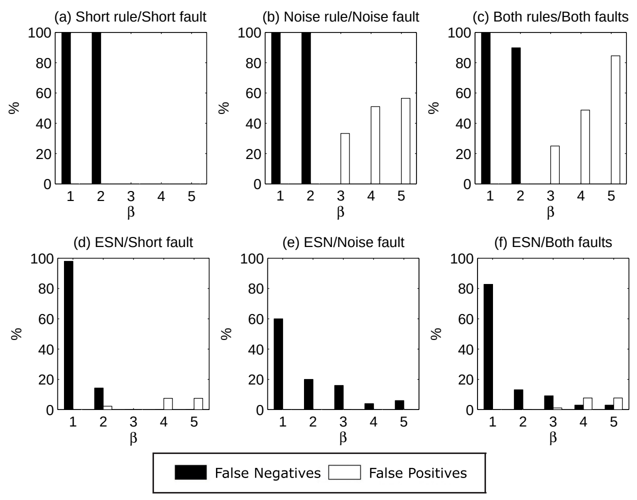


Fig. 8. Short rule, Noise rule, and ESN detection applied to the moisture data set.

amplifications used for the evaluation are: $1 \leq \beta \leq 5$. Figure 8 compares the three algorithms, in the case of moisture data, when applied to the Short faults, Noise faults, and a combination of both faults (5% Short and 5% Noise faults). We only apply each rule to its own domain fault since this is the optimal scenario. The challenge of this data set is the similarity between the onset of rain events and Short faults. In order to avoid false positives the thresholds must be set high enough to avoid triggering the Short rule during the rain events.

In the left column, Figure 8(a,d), we compare the Short rule with the ESN detection when applied to Short faults. Not surprisingly the Short rule performs well on this type of fault when $\beta \geq 3$. However, for lower β values the Short rule cannot distinguish between rain events and faults, and detects none of the latter. The ESN is effective for $\beta \geq 2$ but at the cost of more false positives at higher βs. In the middle column, Figure 8(b,e), we compare the Noise rule with the ESN detection when applied to Noise faults. Interestingly, the Noise rule does not perform well on its corresponding faults. At $\beta \geq 3$ we see the same trend as before with no false negatives, however, we also see a significant number of false positives. This behavior is caused by the aggressiveness of the Noise rule, marking the entire window as faulty rather than individual points. For low β values we still see the ambiguity between events and faults, leading to no positive detections. The ESN detector, however, has no false positives, and a significantly lower number of false negatives for $\beta \leq 2$. Finally, for higher β values the number of false negatives is also significantly smaller than the number of false positives of the rule-based algorithm.

Judging by these results, we conclude that the ESN can match up with the rule based detectors. There is although a trade-off between false positives and false negatives, since decreasing one often leads to the increase of the other. However, in a real deployment it is not possible to choose what algorithm to use on which faults and we must assume that all faults can appear at any time. In

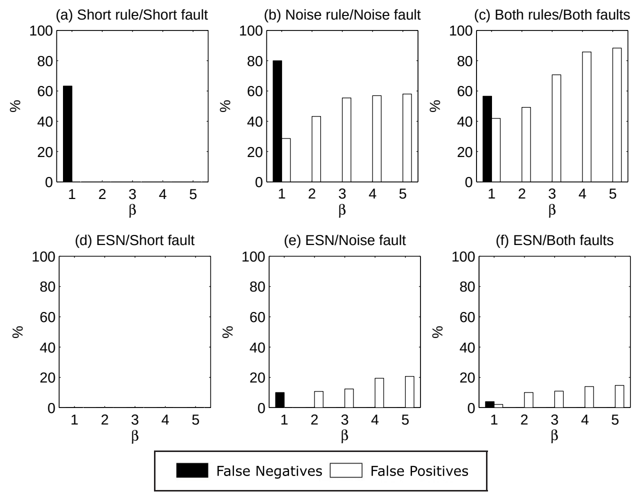


Fig. 9. Short rule, Noise rule, and ESN detection applied to the temperature data set.

the right column, Figure 8(c,f), we thus compare a hybrid detector using both the Short rule and the Noise rule at the same time on a data set injected with both types of faults. We see that the hybrid detector has the same behavior as the Noise rule, with either high number of false negatives or false positives. On the other hand, the ESN detector is performing significantly better across all β values, illustrating the strength of the learning algorithm's ability to detect what is not *normal*.

Next, we perform the same analysis on the temperature data set, using the same parameters to inject errors. The challenge of this data set, from the perspective of a detection algorithm, is the high temperature variance, caused by the diurnal pattern, that resembles noise faults. As before, the Short rule and faults are in the left column, Figure 9(a,d), Noise rule and faults in the middle column, Figure 9(b,e), and the hybrid detector on both types of faults in the right column, Figure 9(c,f). One can see that the overall accuracy improves significantly, with more faults being detected. Also note that the Noise rule generates a large number of false positives, supporting the claim that the diurnal temperature patterns in the data set can be misclassified as Noise faults. Again, when used on both faults simultaneously we see that the false positives is the biggest drawback with the hybrid detector. The ESN detector, however, does not misclassify to the same extent, again clearly showing the ESN's ability to distinguish between *normal* and *anomalous* data.

5.3 Discussion

We have shown that, for the modalities we tested, the ESN is capable of detecting low-amplitude anomalies better than specific rule-based anomaly detectors. At the same time, it is equally effective over multiple anomaly types, as it has the ability to detect a wide range of features deviating from the training set.

There are, however, several factors that limit the applicability of ESNs. We identify three key issues: (1) As we saw in Section 4.2 the prediction time for each iteration is in the order of seconds. For environmental monitoring, where changes happen on the scale of minutes, this prediction speed is acceptable. However, this technique might not be feasible for high data rate applications. (2) For deployments in which no historical data are available, the training data will have to be constructed (e.g., from models, experience, etc.) or learned during the deployment. Neither options are desirable, because an artificial training set will lack the details encountered in the field. (3) Because the ESN is an approximation function, its quality is highly dependent on the size of the dynamic reservoir (DR). In the case of soil moisture and temperature a DR of 50 neurons suffices for anomaly detection. However, given a different set of constraints the DR might not be large enough to encode the dynamics of the underlying modality.

6 Conclusion

This paper unifies fault and event detection in sensor networks under the general framework of anomaly detection. We show that online anomaly detection is feasible on mote-class devices by implementing an Echo State Network (ESN) on a TelosB mote. This network performs as well as a PC-based ESN of the same size, proving that it is feasible to implement sophisticated pattern recognition algorithms on motes. Indeed, the ESN is small and fast enough to function alongside an environmental monitoring application, detecting measurement anomalies in real-time. Depending on the amplitude of the injected anomalies, the ESN provides equivalent or higher detection accuracy compared to rule-based detectors customized to specific faults. However, the most significant feature of the ESN detector is its generality since it is capable of detecting all features not present in the training set.

In our future work we will explore the feasibility of implementing other machine learning techniques, such as Bayesian Networks, on mote-class devices and compare their performance to ESNs. With different methods available, the challenge becomes how to choose the best supervised learning method for mote-based online classification when given a particular training set from the domain scientists.

References

1. Musăloiu-E., R., Terzis, A., Szlavecz, K., Szalay, A., Cogan, J., Gray, J.: Life Under your Feet: A WSN for Soil Ecology. In: EmNets Workshop. (May 2006)
2. Selavo, L., Wood, A., Cao, Q., Sookoor, T., Liu, H., Srinivasan, A., Wu, Y., Kang, W., Stankovic, J., Young, D., Porter, J.: LUSTER: Wireless Sensor Network for Environmental Research. In: ACM SenSys. (November 2007)
3. Tolle, G., Polastre, J., Szewczyk, R., Turner, N., Tu, K., Buonadonna, P., Burgess, S., Gay, D., Hong, W., Dawson, T., Culler, D.: A Macroscope in the Redwoods. In: ACM SenSys. (November 2005)
4. Gupchup, J., Sharma, A., Terzis, A., Burns, R., Szalay, A.: The Perils of Detecting Measurement Faults in Environmental Monitoring Networks. In: DCOSS. (2008)

5. MANA: Monitoring remote environments with Autonomous sensor Network-based data Acquisition systems. `http://mana.escience.dk/`
6. Jaeger, H.: The echo state approach to analysing and training recurrent neural networks. Technical Report GMD Report 148, German National Research Center for Information Technology (2001)
7. Omitaomu, O.A., Fang, Y., Ganguly, A.R.: Anomaly detection from sensor data for real-time decisions. In: Sensor-KDD, Las Vegas, Nevada, USA (August 2008)
8. Wu, E., Liu, W., Chawla, S.: Spatio-temporal outlier detection in precipitation data. In: Sensor-KDD, Las Vegas, Nevada, USA (August 2008)
9. Kaplantzis, S., Shilton, A., Mani, N., Sekercioglu, A.: Detecting selective forwarding attacks in wsn using support vector machines. In: ISSNIP. (2007)
10. Rashidi, P., Cook, D.J.: An adaptive sensor mining framework for pervasive computing applications. In: Sensor-KDD, Las Vegas, Nevada, USA (August 2008)
11. Römer, K.: Distributed mining of spatio-temporal event patterns in sensor networks. In: EAWMS at DCOSS. (Jun 2006)
12. Werner-Allen, G., Lorincz, K., Johnson, J., Lees, J., Welsh, M.: Fidelity and yield in a volcano monitoring sensor network. In: OSDI. (2006)
13. Pister, K.: Tracking vehicles with a UAV-delivered sensor network. Available at http://robotics.eecs.berkeley.edu/ pister/29Palms103/ (March 2001)
14. Hu, W., Tran, V.N., Bulusu, N., Chou, C.T., Jha, S., Taylor, A.: The design and evaluation of a hybrid sensor network for cane-toad monitoring. In: IPSN. (2005)
15. Sharma, A., Golubchik, L., Govindan, R.: On the Prevalence of Sensor Faults in Real-World Deployments. IEEE SECON (2007)
16. Obst, O., Wang, X.R., Prokopenko, M.: Using echo state networks for anomaly detection in underground coal mines. In: IPSN. (April 2008)
17. Wang, X.R., Lizier, J.T., Obst, O., Prokopenko, M., Wang, P.: Spatiotemporal anomaly detection in gas monitoring sensor networks. In: EWSN. (2008) 90–105
18. Cornou, C., Lundbye-Christensen, S.: Classifying sows' activity types from acceleration patterns. Applied Animal Behaviour Science **111**(3-4) (2008) 262 – 273
19. Bokareva, T., Bulusu, N., Jha, S.: Learning sensor data characteristics in unknown environments. In: IWASN. (2006)
20. Chang, M., Terzis, A., Bonnet, P. `http://www.diku.dk/~marcus/esn/`
21. Bai, Z., Demmel, J., Dongarra, J., Ruhe, A., van der Vorst, H.: Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide. SIAM (2000)
22. Marra, S., Iachino, M., Morabito, F.: Tanh-like activation function implementation for high-performance digital neural systems. Research in Microelectronics and Electronics 2006, Ph. D. (June 2006) 237–240
23. Jaeger, H.: Tutorial on training recurrent neural networks, covering bppt, rtrl, ekf and the echo state network approach. Technical Report GMD Report 159, German National Research Center for Information Technology (October 2002)
24. Polastre, J., Szewczyk, R., Culler, D.: Telos: Enabling Ultra-Low Power Wireless Research. In: IPSN/SPOTS. (April 2005)
25. Moteiv Corporation: Tmote Sky. `http://www.moteiv.com/`
26. Herbert Jaeger: Matlab toolbox for ESNs. Available at `http://www.faculty.jacobs-university.de/hjaeger/pubs/ESNtools.zip` Last checked: 2008-08-31
27. Mackey, M.C., Glass, L.: Oscillation and Chaos in Physiological Control Systems. Science **197**(287) (1977)
28. Wolfram Research, Inc.: Quantile-Quantile Plot. `http://mathworld.wolfram.com/Quantile-QuantilePlot.html`
29. Chang, M., Terzis, A., Bonnet, P.: Mote-based online anomaly detection using echo state networks. Technical report, U. Copenhagen, `http://www.diku.dk/OLD/publikationer/tekniske.rapporter/rapporter/09-01.pdf` (2009)

Mote-Based Anomaly Detection

MARCUS CHANG
University of Copenhagen
ANDREAS TERZIS
Johns Hopkins University
and
PHILIPPE BONNET
IT University of Copenhagen

Sensor networks deployed for scientific data acquisition must inspect measurements for faults and events of interest. Doing so is crucial to ensure the relevance and correctness of the collected data. In this work we unify fault and event detection under a general *anomaly detection* framework. We use machine learning techniques to classify measurements that resemble a training set as *normal* and measurements that significantly deviate from that set as *anomalies*. Furthermore, we aim at an anomaly detection framework that can be implemented on motes, thereby allowing them to continue collecting scientifically-relevant data even in the absence of network connectivity. The general consensus thus far has been that learning-based techniques are too resource intensive to be implemented on mote-class devices. In this paper, we challenge this belief. In fact, we show that the accuracy of learning-based detection algorithms on a mote-class device is comparable to their PC-based implementation. Moreover, we compare two techniques based on learning algorithms – Echo State Networks (ESN) and Bayesian Networks (BN) – to rule-based detection algorithms. We find that learning-based techniques are more sensitive to subtle faults and generate fewer false positives than rule-based fault detection. Finally, the BN is as effective as the ESN even though it requires a third of the memory and decreases execution time by an order of magnitude.

Categories and Subject Descriptors: I.5.5 [**Computing Methodologies**]: Pattern Matching—*Implementation*; C.2.4 [**Computer Systems Organization**]: Computer-Communication Networks—*Distributed Systems*

General Terms: Design

Additional Key Words and Phrases: Machine-Learning, Sensor Networks, Bayesian Networks, Echo State Networks, Watchdog

1. INTRODUCTION

Sensor networks deployed to collect scientific data (e.g., [Musăloiu-E. et al. 2006; Selavo et al. 2007; Tolle et al. 2005]) have shown that field measurements are plagued

Author's address: M. Chang, Dept. of Computer Science, University of Copenhagen, Universitetsparken 1, 2100 Copenhagen, Denmark; e-mail: marcus@diku.dk. A. Terzis, Dept. of Computer Science, Johns Hopkins University, 3100 Wyman Park Drive Wyman Park Building, 21211 Baltimore MD, USA; e-mail: terzis@cs.jhu.edu. Ph. Bonnet, IT University of Copenhagen, Rued Langgaards Vej 7, 2300 Copenhagen, Denmark; email: phbo@itu.dk.

with measurement faults. These faults must be detected to prevent pollution of the experiment and waste of network resources. At the same time, networks should autonomously adapt to sensed events, for example by increasing their sampling rate or raising alarms. Events in this context are measurements that deviate from "normal" data patterns, yet they represent features of the underlying phenomenon. One such example would be rain events in the case of soil moisture measurements.

The issue is that detection algorithms tailored to specific types of faults lead to false positives when exposed to multiple types of faults [Gupchup et al. 2008]. More importantly, algorithms which classify measurements that deviate from the recent past as faulty tend to misclassify events as faults [Gupchup et al. 2008]. This misclassification is particularly undesirable because measurements from episodic events are the most important data for domain scientists and should be given the highest priority, while faults have no value and should be discarded thus improving the quality of the underlying sensing instrument.

In this work we unify fault and event detection under a more general *anomaly detection* framework, in which online algorithms classify measurements that significantly deviate from a learned model of data as anomalies. By including punctuated, yet infrequent events in the training set we avoid the misclassification problem mentioned above thus allowing the system to distinguish faults from events of interest. More importantly, these learning-based techniques are *general* in the sense of detecting measurement sequences that contain multiple categories of anomalies that do not exist in the training data.

Obviously anomaly detection can and should also be done on a gateway that correlates data from multiple sensors. Nonetheless, we claim that online detection on motes is also very much relevant. We motivate this need through an example derived from one of our ongoing projects [MANA]. Consider a set of motes deployed under the surface of a lake with limited physical access. These motes are connected to a floating buoy via acoustic modems which can be non-functional over long periods of time, either because the buoy is out of communication range or due to background noise in the lake. The motes should be able to autonomously alter their sensing behavior depending on whether the collected measurements are seemingly faulty or correspond to interesting events. For example, faulty measurements should be replaced in a timely manner by new measurements while interesting events should trigger sampling rate increases.

In this paper, we propose using learning-based anomaly detection on mote-class devices. More specifically, our contributions are the following: **(1)** We develop an anomaly detection framework based on Echo State Networks as well as Bayesian Networks; **(2)** we implement these frameworks on a mote-class device and show that their accuracy is comparable to a PC-based implementation; **(3)** we quantitatively compare these frameworks with two rule-based, fault-specific detection algorithms. We show that learning-based techniques are more sensitive to subtle faults and generate fewer false positives than the rule-based fault detection techniques. Finally, we show that the Bayesian Networks framework is as effective as the Echo State Network framework, while occupying a third of the memory footprint and running ten times faster.

The remainder of this paper is organized as follows. In Section 2 we present related work on anomaly detection. In Section 3 we discuss machine learning algorithms in general and in particular how Echo State Networks and Bayesian Networks can be used for anomaly detection. In Section 4 and Section 5 we present the implementation of the former and the latter, respectively. In Section 6 we evaluate the performance of the two detectors when applied to data collected from an actual deployment and finally in Section 7 we offer our conclusions.

2. RELATED WORK

Anomaly characterization and detection have received significant attention in the sensor network community. In particular, rule- and threshold-based detection schemes have been used extensively due to their simplicity and low resource requirements. For example threshold-based detection has been used in a broad variety of deployments such as seismological event detection [Werner-Allen et al. 2006], Cane toad monitoring [Hu et al. 2005], and vehicle tracking [Pister 2001].

In the context of environmental monitoring, [Sharma et al. 2007] proposed two rules to detect faults commonly observed by such applications: Short faults, defined as drastic differences between two sequential measurements, and Noise faults, defined as periods during which measurements exhibit larger than normal variations. To detect the former, the Short rule compares two adjacent data points and classifies the more recent as faulty when the difference is above a certain threshold. For the latter fault type, the Noise rule considers a sliding window of measurements and flags all of them as faulty if the standard deviation is above a certain threshold. While these detection schemes are very resource efficient, their effectiveness is limited. For example, [Werner-Allen et al. 2006] estimated the accuracy of their detection technique to be as low as 5%-29%. Moreover, these schemes suffer from inherent misclassification problems in which events are mistaken for faults [Gupchup et al. 2008]. We seek a solution that can improve the detection accuracy and avoid misclassification while still be efficient enough to run online on a mote.

More sophisticated algorithms such as pattern recognition [Rashidi and Cook 2008; Römer 2006] and data partitioning [Omitaomu et al. 2008; Wu et al. 2008; Kaplantzis et al. 2007] have been used for anomaly detection in wireless sensor networks. However, the former did not consider raw measurements but relied on meta-tags and the latter removed the temporal information of the data points. We seek a solution that processes the raw data points directly and utilizes the information inherent in the temporal relation among them.

Machine learning techniques have been proposed for anomaly detection in WSNs. For example, Kalman filters have been used for offline sow monitoring [Cornou and Lundbye-Christensen 2008] while Echo State (neural) Networks [Obst et al. 2008] and Bayesian Networks [Wang et al. 2008] have been proposed for offline gas monitoring. [Bokareva et al. 2006] used a Competitive Learning Neural Network (CLNN) for online classification but the neural network was implemented on a Stargate gateway rather than a mote-class device. [Osborne et al. 2008] used a multi-output Gaussian process to predict missing sensor readings, but they also relied on a PC-class device. We bridge the gap between online detection and machine learning by implementing two learning-based techniques on a mote.

In [Chang et al. 2009] we presented an anomaly detection framework based on

Echo State Networks. This neural network variant was used to predict the next measurement value as a function of all the previous ones. The prediction error could subsequently be used for anomaly detection. The evaluation showed the framework to be significantly more accurate than the rule-based counterparts. At the same time ESNs did not suffer from increased numbers of false positives when faced with different types of faults. Although efficient enough to run on a mote alongside a production-level environmental monitoring application, the combined code consumed the mote's entire ROM, while the algorithm's execution speed limited the sampling rate to 1 Hz. In this paper, we extend this framework to Bayesian Networks, on the premise that this data structure captures the temporal relation between data points more efficiently.

3. MACHINE LEARNING

We build our anomaly detection framework on a classification mechanism that accepts measurements matching a learned model as valid and rejects measurements that significantly deviate from learned data as *anomalies*.

We rely on machine learning techniques to build our classification model and focus on supervised learning because our scientific partners are able to provide training sets that correspond to the data they expect. Such data sets include previously collected measurements and synthetic data generated by analytical models.

What learning technique should we choose to achieve both accurate anomaly detection and efficient implementation on a mote-class device? We rule out Kalman filters, because they base each prediction on a sliding window of observed values instead of a compact model of learned data. Initially we thought that a Bayesian Network's learning and inference operations (which are NP-hard [Dagum and Luby 1993]) and the modeling of probability functions (typically Gaussians), would preclude its use on resource constrained devices. Consequently, we decided to use Echo State Networks to meet our requirements on classification efficiency (i.e., minimize false classifications) and resource use (i.e., minimize CPU, memory, and energy usage). However, as we later show inference for certain Bayesian Network types can in fact be done in polynomial time. Furthermore, by performing the learning step in a pre-processing phase we were able to use Bayesian Networks as well.

We continue by describing how Echo State Network and Bayesian Networks can be used for anomaly detection.

3.1 Neural Networks

A neural network can be informally considered as an approximation function. Specifically, when presented with a subset of the original function's value pairs during the *training* stage, the neural network generalizes over these data and approximates the outcome of the original function in the *prediction* stage. Formally, a neural network is a weighted directed graph where each vertex represents a neuron. We consider discrete-time networks consisting of K input neurons, N hidden neurons, and L output neurons. The input neurons act as sources and the output neurons as sinks. The value of neuron j is given by: $v_j = \mathrm{A}(\sum w_{ij} v_i)$, where v_i is the output of neuron i, w_{ij} is the weight of the edge connecting neuron i to j, and A() is the *activation function*. This function is typically tanh() or a similar function. The training stage consists of adjusting the network's weights to approximate its output signal to the training signal.

Echo State Networks. In an ESN, all neurons are interconnected (but can have zero-weighted edges) meaning cycles involving one or more neurons are allowed. This gives each neuron the capability to remember, adding memory to the network as a whole. All the neurons' connections, directions, and weights are generated randomly and do not change, except for the output weights which are changed during training. The neurons thus act as a black box referred to as the Dynamic Reservoir (DR). This property reduces the learning algorithm to a simple linear regression. According to the Echo State property [Jaeger 2001], the DR contains a set of basis states and by adjusting the output weights it is possible to capture the 'echoes' of real states as linear combinations of these basis states. Although the DR is randomly generated, [Jaeger 2001] proved that it is possible to ensure that the DR indeed has the Echo State property by enforcing certain conditions. One such condition is that the DR must be sparsely connected, i.e., 10% of all possible connections are actually active.

The decoupling of the DR from the output weights enables several optimizations that fit mote-class devices particularly well. For example, the same DR can be used for multiple tasks by storing task-specific output weights and post-deployment updating can be done without transmitting the entire DR. Also, the requirement that the DR is sparsely connected combined with the use of sparse matrix algebra, allows us to implement ESNs that are larger than regular Feed Forward networks.

Anomaly Detection. We use ESNs to determine whether sensor readings are anomalous by comparing the ESN predictions to the actual measurements. In order to quantify the prediction error we look at the absolute differences between the measurements ($\vec{M}$) and the predictions ($\vec{P}$), i.e., $\vec{\delta} = |\vec{M} - \vec{P}|$. This difference should ideally be close to zero for *normal* data, while *anomalous* data should result in large differences (peaks). In other words, the ESN transforms the original time series into one whose values are ~ 0 most of the time, corresponding to the expected data. Anomaly detection thus reduces to recognizing the peaks in the transformed signal. We can then use pattern matching algorithms based on simple thresholds that have been proven to be both efficient and effective for such simple signals.

3.2 Bayesian Networks

Informally, a Bayesian Network is a probabilistic model where given some *evidence* (measurements) it can estimate the *expectation value* (most probable outcome) and the *likelihood* (probability) of both the evidence and expectation value. Formally, a Bayesian Network is a directed acyclic graph in which each node represents a random variable and every directed arc represents a dependency relation between the parent node and the child node. Each edge includes a conditional probability distribution (CPD) $P(A = a|B = b)$, where A is the random variable of the child node and B is the random variable of the parent. In each node a joint CPD can be constructed using the CPDs of the connected edge(s). CPDs are usually implemented as tables when the random variables are discrete and as Gaussian probability distributions when the random variables are continuous.

When given the values of some of the random variables the expectation values and probability distributions in both children and parent nodes change. This *evi-*

dence, which can be from observations or direct measurements, fixes the value of the now *observed* nodes. Using Bayes rule and the CPDs, probabilities are propagated through the network to the rest of the *hidden* nodes. This process of computing the posterior probability for query purposes is called *inference*. The values with the highest probabilities become the *expectation values* and the corresponding probability the *likelihood*.

The structure of the Bayesian Network are either constructed using a priori knowledge, e.g., rain causes wet grass, or learned from a training set. Similarly, the CPDs are also constructed using knowledge or learned from a training set using statistical methods [Russell and Norvig 2003].

Anomaly Detection. Each node in our Bayesian Network represents a measurement. As a consequence, random variables are continuous. As measurements are obtained sequentially in time, the structure of our Bayesian Network is a subset of a fully connected DAG, where each edge points forward in time. The CPDs are not derived from the laws of nature, but learned from the training set.

By entering the measured data as evidence we can use the CPDs to infer the likelihood of any particular sequence of measurements. When measurements are similar to the training set the likelihood value will be high, while the likelihood will drop significantly when an anomalous data point is introduced. Anomaly detection can thus be recast as a pattern recognition problem on this greatly simplified signal.

3.3 Discussion

As approximation functions, neural networks can learn the structure and correlation between data points. However, this information is not stored very efficiently because the network is essentially created by memorizing patterns of data points. On the other hand, Bayesian Networks are structured from the physical dependencies in the data and the CPDs are tailored to each individual node and not the entire network as a whole. Hence, we expect that the same amount of information that can be stored in a neural network can be stored in a much smaller Bayesian Network, reducing both the memory usage and computations needed in each iteration.

A limitation that Bayesian Networks share with all learning algorithms, including neural networks, is their dependency on training data. In our case, this limiting factor is transformed into the accuracies of the CPDs, which we construct empirically from the training data. If these data do not represent what the domain scientists consider to be “normal”, the resulting likelihood will be useless. For deployments in which no historical data are available, the training data will have to be constructed (e.g., from models, experience, etc.) or learned during the deployment. Neither options are desirable, because an artificial training set will lack the details encountered in the field. Therefore, the choice of training sets is a very interesting open problem, which is beyond the scope of this paper. We just note that a key issue in successfully deploying a Bayesian Network based on learning lies in the choice and availability of training data.

For example, adjusting the sampling rate in an adaptive sampling environment can change the properties of the measurement time series and thus possibly invalidate the training set. This issue can however be remedied, by storing different CPDs for each sampling rate, or by disregarding higher sampling rates when entering the evidence into the Bayesian Network. On the positive side, Bayesian Networks with continuous CPDs (such as Gaussian probability distributions) have the ability to generalize over the training data. By this we mean that data similar to the training data will yield similar likelihoods even though they are not exactly identical. This feature allows motes deployed in similar regions to share the same training data instead of requiring mote-specific training sets.

4. ESN ON A MOTE

4.1 Implementation

While we create and train the ESNs offline, a complete ESN (including the network's activation function, output weights, and the DR) is included in the application that runs on the mote. We use TinyOS 2.x to ensure portability to a broad range of mote class devices. Our implementation, publicly available for download[1], focuses on feasibility and efficiency: the ESN must be able to fit in memory and the algorithm must be fast enough to maintain the desired sampling rate. Next, we present three optimizations to improve performance along these two axes.

Sparse Matrix Algebra. The size of the DR's weight matrix grows quadratically with the number of neurons in the reservoir n. However, only 10% of these elements are non-zero because the DR must possess the Echo State property. We leverage this feature by storing the matrix using Compressed Row Storage [Bai et al. 2000], which only stores the non-zero elements and the layout of the matrix. Doing so reduces the necessary storage from $O(n^2)$ to $O(2n_z+n+1)$, where n_z is the number of non-zero elements. This technique also reduces the number of operations related to matrix multiplications by a similar factor by considering only non-zero elements.

Single Floating Point Precision. Most mote-class devices rely on software emulated floating point operations due to lack of dedicated hardware. This contributes to both the storage and runtime overheads. At the cost of reduced floating point precision we select to store and compute all values using single instead of double floating point precision. Doing so halves the size of all the weight matrices and reduces the number of emulated floating point operations needed. As we later show, the resulting loss of precision is tolerable.

Tanhlike Activation Function. Because the activation function has to be applied to all the neurons in every iteration, it is important to choose a function that can be implemented efficiently. At the same time, choosing a suboptimal activation function can significantly degrade the ESN's output quality. The algorithm for the often used hyperbolic tangent, tanh(), has high complexity requiring both large amounts of storage and a significant processing time. Because of these shortcomings, [Marra et al. 2006] proposed the approximate function:

$$TL(x) = \operatorname{sign}(x)\left[1 + \frac{1}{2^{\lfloor 2^n |x| \rfloor}}\left(\frac{2^n |x| - \lfloor 2^n |x| \rfloor}{2} - 1\right)\right]$$

where $n \in \mathbb{Z}$ determines the steepness of the function. This *tanhlike* function has properties similar to tanh() (when $n = 1$) but with far lower complexity. However,

[1] `http://www.diku.dk/~marcus/esn/`

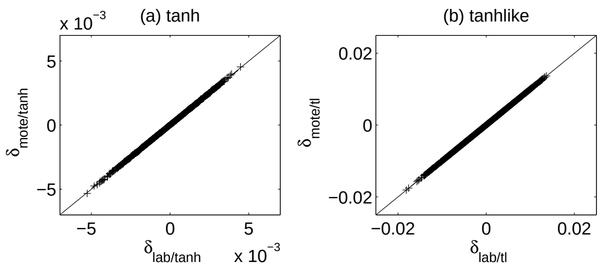


Fig. 1. (a) Q-Q plot of $\vec{\delta}_{lab/tanh}$ and $\vec{\delta}_{mote/tanh}$. (b) Q-Q plot of $\vec{\delta}_{lab/tl}$ and $\vec{\delta}_{mote/tl}$.

it is also a non-differentiable, piecewise-linear function because of the rounding operations ($\lfloor\cdot\rceil$). Therefore, we expect the quality of the ESN's output to be lower than when using tanh(), because small changes in input will result in large changes in output if these changes happen across a linear junction.

4.2 Evaluation

We verify that our ESN implementation indeed performs well on a mote-class device by comparing its output to a reference ESN running on a PC. We consider ESNs which consist of two input signals (with one of the input signals held at a constant bias value in order to improve performance [Jaeger 2002]), a 10-400 neuron reservoir, and one output signal (i.e., $K = 2, N = 10 - 400$, and $L = 1$). All mote experiments are carried out on a TelosB mote [Polastre et al. 2005], running TinyOS 2.x with the clock frequency set to the default speed of 4 MHz [Moteiv Corporation]. Data sets are stored in ROM with measurements read with a fixed frequency to simulate sensor sampling. We use Matlab R2007a with the *Matlab toolbox for ESNs* [Herbert Jaeger] as our reference implementation. We use the Mackey-Glass (MG) time series with a delay $\tau = 17$ [Mackey and Glass 1977] to evaluate our ESN implementation. This system is commonly used to benchmark time series prediction methods because of its chaotic nature when $\tau \geq 17$ [Müller et al. 1995; Wang and Fu 2005].

4.2.1 *Sanity Check.* We create a MG time series with 5,000 samples and use the first 2,000 samples to train a 50 neuron ESN, the next 100 samples for initialization, and the last 2,900 samples as the prediction vector $\vec{MG}$. Both the tanh() and *tanhlike* activation functions are used resulting in four different predictions: $\vec{P}_{lab/tanh}$, $\vec{P}_{mote/tanh}$, $\vec{P}_{lab/tl}$, and $\vec{P}_{mote/tl}$. We compute the four prediction errors and normalized root-mean-squared deviations (NRMSD):

	$\vec{\delta}_{lab/tanh}$ (%)	$\vec{\delta}_{mote/tanh}$ (%)	$\vec{\delta}_{lab/tl}$ (%)	$\vec{\delta}_{mote/tl}$ (%)
NRMSD	0.15448	0.15470	0.5855014	0.5855008

Figure 1 presents the Q-Q plots (Quantile-Quantile-plot) [Wolfram Research] of the prediction errors grouped by activation function. Since the points in the Q-Q plots lie on a straight line with slope one they pairwise belong to the same distribution. Together with the NRMSDs from the same activation functions being almost

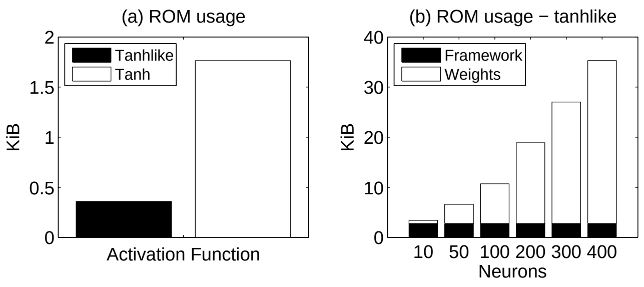


Fig. 2. (a) ROM footprints for the tanh() and *tanhlike* functions. (b) Total ROM footprint for an ESN using the custom *tanhlike* activation function.

identical we conclude that the TelosB ESN implementation has the same accuracy as the one in Matlab. Also, with an NRMSD less than 1% we see that the 50-neuron ESN is indeed capable of tracking the MG time series. However, the choice of activation function impact the accuracy of the predictions significantly, with tanh() being four times more accurate than the *tanhlike* function. This supports our claim that the piecewise-linearity of the *tanhlike* function degrades performance.

In order to compare the double precision floating point in Matlab with that of the single precision floating point on the TelosB, we look at the differences between predictions from the former with the latter when using the same activation function, i.e., $\vec{\delta}_{tanh} = \vec{P}_{lab/tanh} - \vec{P}_{mote/tanh}$ and $\vec{\delta}_{tl} = \vec{P}_{lab/tl} - \vec{P}_{mote/tl}$. We compute the NRMSDs for both error distributions:

	$\vec{\delta}_{tanh}$ (%)	$\vec{\delta}_{tl}$ (%)
NRMSD	$6.6 \cdot 10^{-3}$	$1.3 \cdot 10^{-4}$

Because $NRMSD(\vec{\delta}_{tanh}) < NRMSD(\vec{\delta}_{lab/tanh})$ and $NRMSD(\vec{\delta}_{tl}) < NRMSD(\vec{\delta}_{lab/tl})$ the errors caused by using single precision floating point are smaller than the errors caused by the ESN predictions. Thus, using single precision floating point on the TelosB is sufficient.

4.2.2 *Performance.* In order to explore the implementation's characteristics, such as ROM footprint, runtime speed, and accuracy, we vary the number of neurons in the DR. The ROM usage can be divided into two components: (1) *Framework*, the ESN algorithm used for prediction. (2) *Weight Matrices*, the DR and output weights. Whereas (1) is constant, (2) depends on the number of neurons in the reservoir. Figure 2a presents the ROM size difference for the two activation functions and Figure 2b shows the ROM footprint of the aforementioned components (using *tanhlike*). We observe that the memory contribution from the reservoir grows linearly, confirming the storage requirement of the Compressed Row Storage ($O(2n_z + n + 1)$). Also, the ROM footprint is 1,806 bytes for tanh() and 368 bytes for *tanhlike*, making the former five times larger than the latter.

Next we measure the runtime cost of the ESN implementation. For each iteration, the ESN prediction algorithm performs the following set of operations: (1) *Matrix*, matrix-vector multiplication. (2) *Activation Function*, application of the

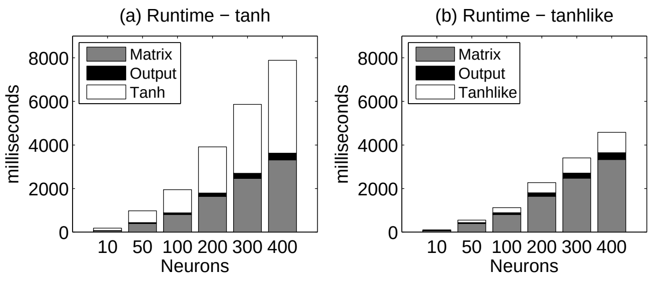


Fig. 3. Total execution cost of one ESN iteration divided into three components. (a) using the GCC built-in tanh() activation function. (b) using the custom tanhlike activation function.

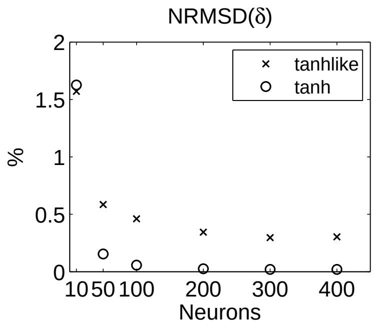


Fig. 4. $NRMSD(\vec{\delta})$ for different reservoir sizes and activation functions.

activation function. (3) *Output*, vector-vector multiplication. Figure 3 summarizes the execution time of one prediction step and the three operations' contributions. Surprisingly, the tanh() activation function is the most expensive operation and not the matrix-vector multiplication. It takes 28% longer to run than the matrix-vector multiplication and 453% longer than the *tanhlike* activation function.

Finally, we look at the prediction error as a function of reservoir size and activation function. We compare against the MG time series and find the $NRMSD(\vec{\delta})$ for the six reservoirs and two activation functions used above. Figure 4 presents the results of this comparison. As expected, the prediction error decreases as the reservoir size increases and the $tanh()$ activation function leads to more accurate predictions in general. Upon closer inspection, there appear to be three distinct regions relative to the reservoir size: small (10 neurons), medium (50-300 neurons), and large (300-400 neurons). In the small region, the prediction error is dominated by the small size of the reservoir and the choice of activation function becomes less important. In the medium region there is a diminishing, yet clear reduction of the prediction error as the reservoir size increases. Finally, in the large region the prediction error does not decrease by adding neurons to the reservoir. Interestingly, the largest contribution to the prediction error comes from the activation function, with no overlap of prediction errors for the 50-400 neuron reservoirs. In fact, even the 50 neuron tanh() reservoir outperforms the 400 neuron *tanhlike* reservoir.

5. BAYESIAN NETWORK ON A MOTE

5.1 Implementation

Similar to the ESN implementation in Section 4.1, we create and train the Bayesian Network offline in a pre-processing stage, while the actual inference of the expectation value and likelihood is done repeatedly online on the mote for the purpose of anomaly detection. The implementation in TinyOS 2.x is publicly available for download[2] and is tailored specifically for anomaly detection in order to ensure feasibility and efficiency. We present the following optimizations to accomplish these goals.

Inference. The anomaly detection, presented in Section 3.2, is based on finding the likelihood value of a series of measurements, by entering them as evidence and inferring the corresponding likelihood. This implies that all nodes are observable and inferring the likelihood reduces to finding the probability of the given evidence. Thus, it is not necessary to perform probability propagation (which is NP-complete) and the CPDs reduce to univariate Gaussian Probability Density Function.

Log-likelihood. We apply the natural logarithm to the likelihood value, because only the relative value is needed for the detection algorithm and not the absolute one. This leads to several key benefits. First, the Gaussian Probability Density Function [Weisstein] reduces to

$$P(X=x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \Leftrightarrow p_{\ln}(x) = -\ln(\sigma\sqrt{2\pi}) + \frac{-1}{2\sigma^2}(x-\mu)^2$$

where μ and σ are the mean and standard deviation respectively. This removes the need to implement the exponential function in software. Second, because both $-\ln(\sigma\sqrt{2\pi})$ and $\frac{-1}{2\sigma^2}$ are normalization constants that can be calculated during pre-processing we do not need to implement the logarithm function either. Third, since we are working with the log-likelihood, combining probabilities reduces to performing simple additions instead of the more expensive multiplications.

Single Floating Point Precision. As in Section 4.1, we store and compute all values using single floating point precision. We later show that both the precision of the Bayesian Network and the range of the log-likelihood values are less than the precision and range of single floating points.

5.2 Evaluation

Similar to the evaluation of the ESN in Section 4.2, we verify that our Bayesian Network implementation indeed performs well on a mote-class device by comparing its output to a reference Bayesian Network running on a PC. We consider Bayesian Networks with the structure described in Section 3.2 and a network size of 3-40 nodes. Again, all mote experiments are carried out on a TelosB mote [Polastre

[2] `http://www.diku.dk/~marcus/bn/`

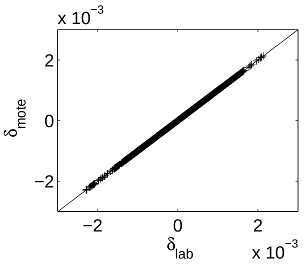


Fig. 5. Q-Q plot of $\vec{\delta}_{lab}$ and $\vec{\delta}_{mote}$

et al. 2005], running TinyOS 2.x with the clock frequency set to the default speed of 4 MHz [Moteiv Corporation]. Data sets are stored in ROM and measurements are read one data point at the time to simulate sensor sampling. We use Matlab R2009a with the *Bayesian Net Toolbox for Matlab 1.0.4* [Kevin Patrick Murphy et al.] as our reference implementation. As our prediction value, we use the expectation value from the Bayesian Network to track the Mackey-Glass (MG) time series from before.

5.2.1 *Sanity Check.* We use the first 2,000 samples of a MG time series with 5,000 samples to train a five node Bayesian Network and the last 3,000 samples as our measurements ($\vec{M}$). We look at the absolute differences between the measurements and the predictions from the Bayesian Network ($\vec{P}$), i.e., $\vec{\delta} = |\vec{M} - \vec{P}|$, when executed on both the mote ($\vec{\delta}_{mote}$) and in Matlab ($\vec{\delta}_{lab}$). From the prediction errors we calculate the normalized root-mean-square deviations (NRMSD) to be (when rounded to nearest single floating points precision):

	$\vec{\delta}_{mote}$ (%)	$\vec{\delta}_{lab}$ (%)
NRMSD	0.083599985	0.083600049

Next, we compare the distribution of these prediction errors against each other in a Q-Q-plot in Figure 5 and conclude that both $\vec{\delta}_{mote}$ and $\vec{\delta}_{lab}$ belong to the same distribution because the points lie on a straight line with slope one [Wolfram Research]. Coupled with the almost identical NRMSDs, we deduce that the accuracy of the TelosB Bayesian Network implementation has the same accuracy as the one in Matlab (i.e., within an acceptable margin of error). Also, with an NRMSD less than 1 ‰ we conclude that the 5-node Bayesian Network is indeed capable of tracking the MG time series.

In order to compare the double precision floating point in Matlab with that of the single precision floating point on the TelosB, we compute the differences between predictions from the former with the latter, i.e., $\vec{\delta}_P = \vec{P}_{lab} - \vec{P}_{mote}$. We compute the NRMSD to be:

	$\vec{\delta}_P$ (%)
NRMSD	$7.3 \cdot 10^{-6}$

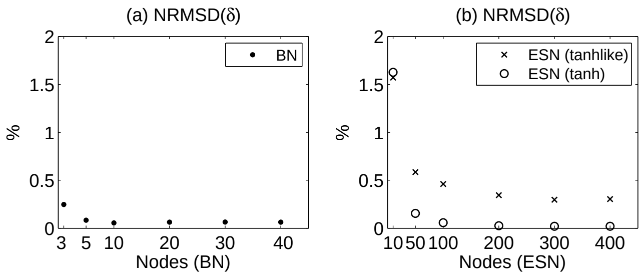


Fig. 6. *NRMSD*($\vec{\delta}$) for (a) BN and (b) ESN at different network sizes.

Because *NRMSD*($\vec{\delta}_P$) $<$ *NRMSD*($\vec{\delta}_{lab}$) the errors caused by using single precision floating point are smaller than the errors caused by the Bayesian Network predictions. Thus, using single precision floating point on the TelosB has no noticeable effect on the Bayesian Network inference.

5.2.2 *Performance.* As before, we vary the number of nodes in the Bayesian Network in order to explore the implementation's characteristics, such as ROM footprint, runtime speed, and accuracy. We compare with the ESN benchmarks from Section 4.2.2 while using the *tanhlike* activation function, since this is the most efficient of the two. However, instead of comparing equal size networks we compare networks with similar accuracy.

To determine the accuracy for different sizes Bayesian and Echo State Networks we find the prediction errors for the MG time series, *NRMSD*($\vec{\delta}$), and show the results in Figure 6. As expected, the network size required for a Bayesian Network to achieve a precision of the same order of magnitude as an Echo State Network is significantly smaller; in fact there is an order of magnitude in difference.

Common to both methods is that increased network size improves accuracy – a result that one would expect. However, this trend is only true up to a certain point after which increasing the network size has no influence on accuracy. The reason for this behavior lies in the chaotic nature of the MG time series which becomes harder to predict the farther away from the initial position one gets. In other words, adding more "history" in the form of more nodes do not improve the accuracy.

Next we compare the memory footprint for these particular network sizes. The ROM usage for both the Bayesian Network and the Echo State Network can be divided into two components: (1) *Framework*, the machine learning algorithm, which is constant. (2) *Network*, which depends on the number of nodes. Figure 7 presents the ROM size difference for the two algorithms. The framework sizes are almost equally large mainly due to the software emulation of floating point operations, which is common for both and constitutes half of their size. Although the size of a single Bayesian Network node is ten times larger than the size of an Echo State Network node, when comparing networks with similar accuracy the total size of the Bayesian Network is significantly smaller. Specifically, the 5-node Bayesian Network is only 38 % the size of the 50-node Echo State Network.

Finally, we measure the runtime cost of the Bayesian Network implementation.

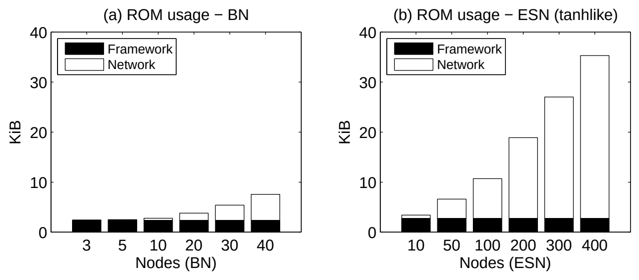


Fig. 7. Total ROM footprint for the (a) BN and (b) ESN.

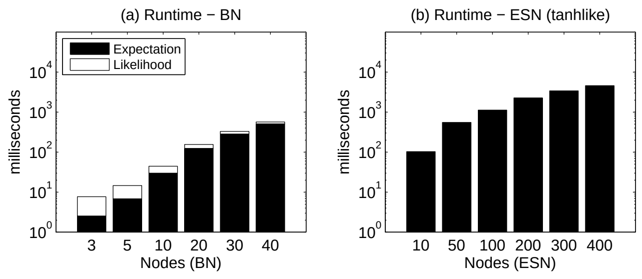


Fig. 8. Total execution cost of one iteration for (a) BN and (b) ESN.

For each iteration, the Bayesian Network's inference algorithm performs the following two operations: (1) *Expectation*, finding the expectation value given the evidence. (2) *Likelihood*, estimating the probability of the evidence when compared to the expectation value. Figure 8 summarizes the execution time of one iteration on a logarithmic scale and compares the result with similar measurements of the Echo State Network. Not surprisingly, there is an order of magnitude in difference because of the larger Echo State Network. However, when comparing the 10 node Bayesian Network with the 10-node Echo State Network, the Bayesian Network is still twice as fast using only 46 ms compared to the Echo State Network's 105 ms.

6. EVALUATION

6.1 Experimental Design

In [Chang et al. 2009] we showed that a 50 node Echo State Network is small enough to be incorporated to an existing data collection application and still be accurate enough to provide anomaly detection. Motes in these sensor networks collect soil temperature and soil moisture readings every 20 minutes and store them to their onboard flash memory. All measurements are periodically offloaded over the network and persistently stored in a database. This environmental sensing application has been actively deployed for the past four years [Musăloiu-E. et al. 2006]. The application uses 40,824 bytes of ROM, leaving 8,328 bytes available on the TelosB. The 50 node Echo State Network uses 6,788 bytes of ROM and has a prediction time of 572 ms for each measurement.

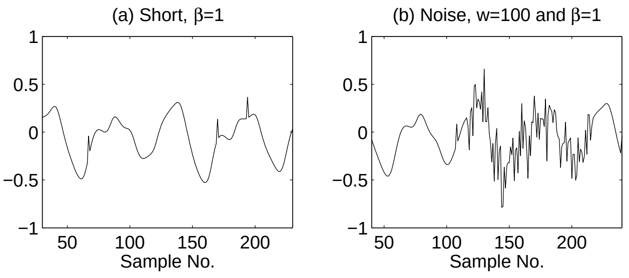


Fig. 9. Two types of injected anomalies: (a) Short faults and (b) Noise faults.

The results from the previous section suggest that a 5-node Bayesian Network, which is both smaller and faster, should still provide higher accuracy than the Echo State Network. This Bayesian Network, with a size of 2,580 bytes and an inference time of 15 ms, will be used for the remainder of this section.

6.1.1 *Anomaly Types.* We focus on two types of random anomalies: Short and Noise. These were defined by [Sharma et al. 2007] and summarized in Section 2. Figure 9 provides samples of these faults. We use two parameters to control the injection of Short anomalies: the sample error rate and the amplification factor, β. For each anomalous measurement, $\tilde{m_i}$, we multiply the standard deviation of the original signal, σ, with β to obtain: $\tilde{m}_i = m_i + \beta\sigma$, where m_i is the true measurement.

For Noise anomalies, we use three parameters: the sample error rate, the period length, w, and the amplification factor, β. For each noisy period, we calculate the standard deviation of the underlying signal and multiply it with β to create a random normal distribution with zero mean and $\beta\sigma$ standard deviation (i.e., $N(0, \beta\sigma)$). We then add samples from this distribution to each of the true measurements within that period.

6.1.2 *Detection Algorithms.* We use the two rule-based anomaly detection algorithms defined by [Sharma et al. 2007] and summarized in Section 2 to detect the two anomalies mentioned above. We use these algorithms as reference as they are directly related to the anomalies we inject and their complexity is comparable to that of currently deployed fault detection algorithms. Our strategy for setting the thresholds is to minimize the number of false positives when the detection algorithms are applied to data sets with no anomalies.

6.1.3 *Data Sets.* For each of the soil moisture and soil temperature modalities that we use, we obtain a training and a test data set from the experiment's database [Musăloiu-E. et al. 2006]. Each of the four data sets consists of 1,000 data points. Figure 10 illustrates two such data sets. The data has been automatically sanitized by the database as a standard procedure for removing anomalies, following the methods proposed by [Sharma et al. 2007]. Using preprocessed rather than raw data prevents any bias caused by anomalies already present in the data stream. Instead, we assume that the only anomalies in the data are the ones we explicitly inject, thereby establishing the ground truth for evaluation purposes.

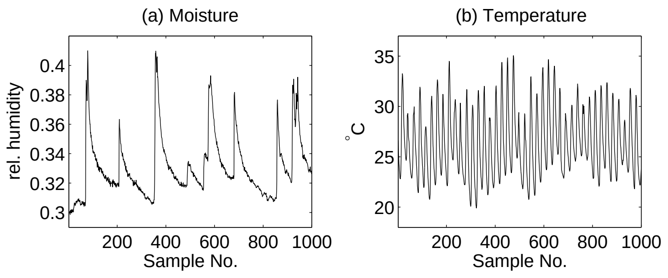


Fig. 10. Environmental sensing data sets. (a) Soil moisture and (b) Soil temperature.

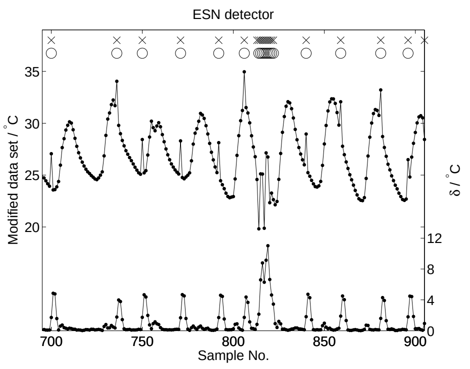


Fig. 11. Correlation between temperature measurements (middle plot), prediction error (bottom plot), and injected/detected anomalies (top X/O markers). (Short $\beta = 1$; Noise $\beta = 1$ and $w = 10$).

6.2 Results

Figures 11 and 12 illustrate the operation of the Echo State Network anomaly detection algorithm, while Figures 13 and 14 illustrate the operation of the Bayesian Network anomaly detection algorithm. All figures show the relation between injected anomalies, measurements, and the detected anomalies.

Notice that the Echo State Network's prediction error is indeed an almost constant signal overlaid with sharp peaks coinciding with the injected faults. Similarly, the Bayesian Network's log-likelihood has sharp drops coinciding with the injected faults. When not injected with anomalies we find that $NRMSD(\vec{\delta}_{Temp}) = 2.0\%$ and $NRMSD(\vec{\delta}_{Moist}) = 3.8\%$ for the Bayesian Network and $NRMSD(\vec{\delta}_{Temp}) = 2.4\%$ and $NRMSD(\vec{\delta}_{Moist}) = 4.4\%$ for the Echo State Network.

We use a 5% sample error rate (i.e., 5% of the measurements are polluted with errors) for each fault type and a period $w = 10$ for Noise faults. The amplifications used for the evaluation are: $1 \leq \beta \leq 5$.

Figure 15, 16, and 17 compare the four algorithms, in the case of moisture data polluted with Short faults, Noise faults, and a combination of both faults (5% Short and 5% Noise faults) respectively. We only apply each rule to its own domain fault since this is the optimal scenario. The challenge of this data set is the similarity

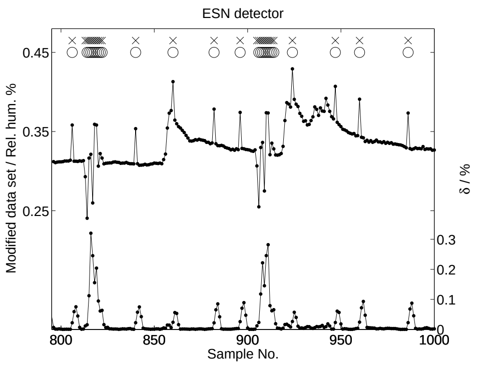


Fig. 12. Correlation between moisture measurements (middle plot), prediction error (bottom plot), and injected/detected anomalies (top X/O markers). (Short $\beta = 2$; Noise $\beta = 2$ and $w = 10$).

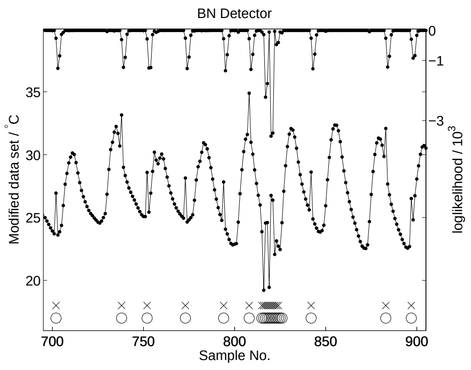


Fig. 13. Correlation between temperature measurements (middle plot), log-likelihood (top plot), and injected/detected anomalies (bottom X/O markers). (Short $\beta = 1$; Noise $\beta = 1$ and $w = 10$).

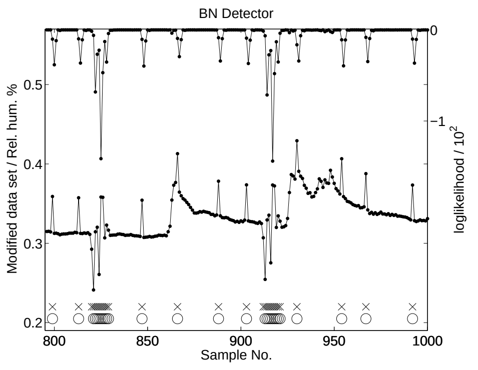


Fig. 14. Correlation between moisture measurements (middle plot), log-likelihood (top plot), and injected/detected anomalies (bottom X/O markers). (Short $\beta = 2$; Noise $\beta = 2$ and $w = 10$).

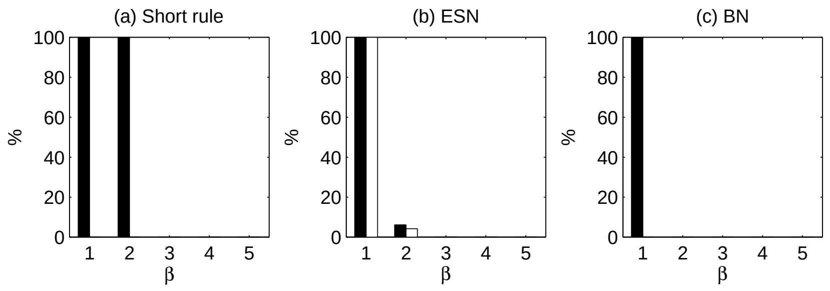


Fig. 15. Short rule, ESN detection, and BN detection applied to the moisture data set with short faults present. Black and white bars represent false negatives and false positives, respectively.

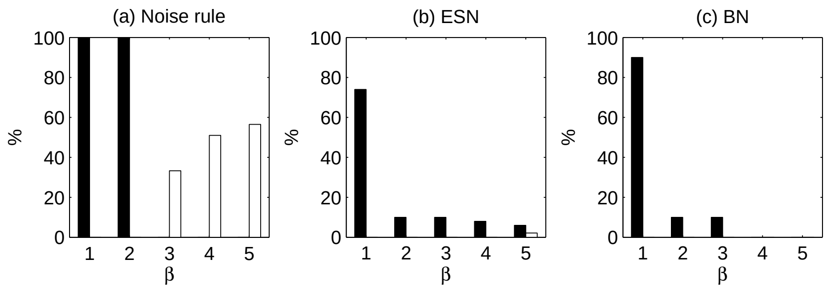


Fig. 16. Noise rule, ESN detection, and BN detection applied to the moisture data set with noise faults present. Black and white bars represent false negatives and false positives, respectively.

between the onset of rain events and Short faults. In order to avoid false positives the thresholds must be set high enough to avoid triggering the Short rule during the rain events.

In Figure 15, we compare the Short rule and ESN with the BN detection when applied to Short faults. Not surprisingly the Short rule performs well on this type of fault when $\beta \geq 3$. However, for lower β values the Short rule cannot distinguish between rain events and faults, and detects none of the latter. Meanwhile, ESN is effective for $\beta \geq 2$ with very few false positives and negatives for $\beta = 2$. BN on the other hand is completely effective for $\beta \geq 2$ without any false positives and negatives at all.

In Figure 16, we compare the Noise rule and ESN with the BN detection when applied to Noise faults. Interestingly, the Noise rule does not perform well on its corresponding faults. At $\beta \geq 3$ we see the same trend as before with no false negatives, however, we also see a significant number of false positives. This behavior is caused by the aggressiveness of the Noise rule, marking the entire window as faulty rather than individual points. For low β values we still see the ambiguity between events and faults, leading to no positive detections. The ESN detector has a significantly lower number of false negatives for $\beta \leq 2$ and for higher β values the number of false negatives is also significantly smaller than the number of false positives of the rule-based algorithm. Here the BN yields a slightly higher amount

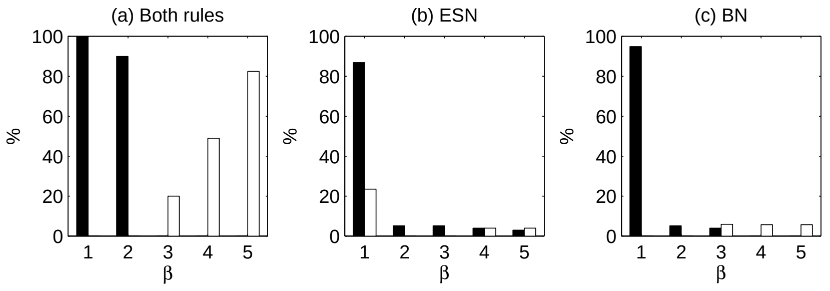


Fig. 17. Both rules, ESN detection, and BN detection applied to the moisture data set with both faults present. Black and white bars represent false negatives and false positives, respectively.

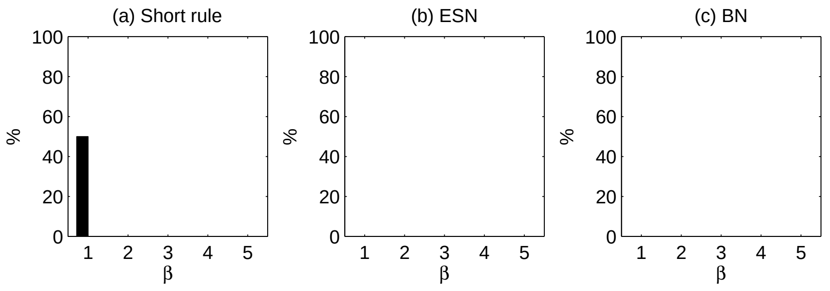


Fig. 18. Short rule, ESN detection, and BN detection applied to the temperature data set with short faults present. Black and white bars represent false negatives and false positives, respectively.

of false negatives for $\beta = 1$ when compared to the ESN, however, at $\beta \geq 4$ the BN has neither false positives nor negatives.

Because it is not possible in practice to selectively use different detection algorithms based on the fault type, we must assume that all faults can appear at any time. For this reason, we compare a hybrid detector using both the Short rule and the Noise rule at the same time on a data set injected with both types of faults.

It is evident from the results in Figure 17 that the hybrid detector behaves similarly to the Noise rule, having either high number of false negatives or false positives. On the other hand, the ESN and BN detectors perform significantly better across all β values, illustrating the learning algorithms' ability to detect what is not *normal*. More interestingly, both algorithms perform equally well. Judging from these results, we conclude that the smaller and faster BN can match up with the ESN and that both are superior to the rule based detectors.

Next, we perform the same analysis on the temperature data set, using the same parameters to inject errors. The challenge of this data set, from the perspective of a detection algorithm, is the high temperature variance, caused by the diurnal pattern, that resembles noise faults.

The Short rule and faults can be seen in Figure 18, Noise rule and faults in Figure 19, and the hybrid detector on both types of faults in Figure 20.

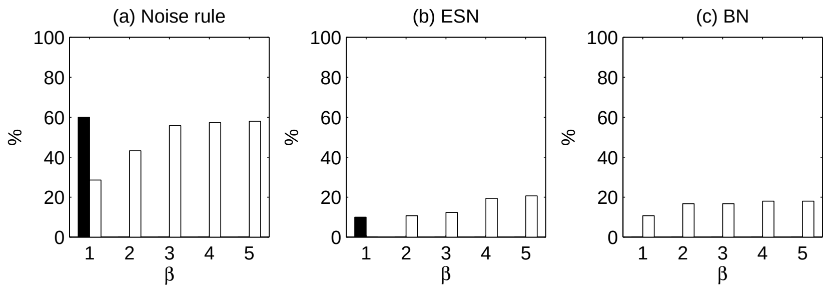


Fig. 19. Noise rule, ESN detection, and BN detection applied to the temperature data set with noise faults present. Black and white bars represent false negatives and false positives, respectively.

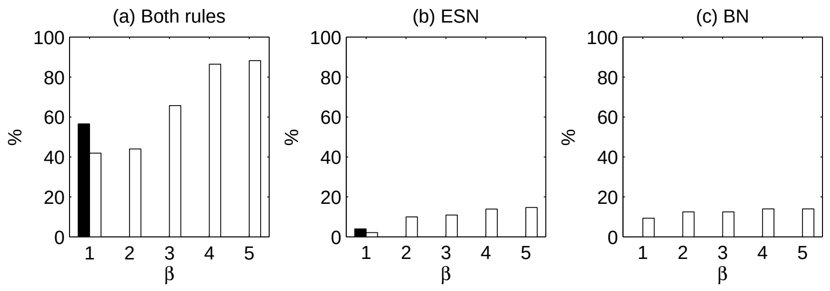


Fig. 20. Both rules, ESN detection, and BN detection applied to the temperature data set with both faults present. Black and white bars represent false negatives and false positives, respectively.

Overall, the accuracy improves significantly, with more faults being detected. Also, note that the Noise rule generates a large number of false positives, supporting the claim that the diurnal temperature patterns in the data set can be misclassified as Noise faults.

Again, when used on both faults simultaneously we see that the false positives is the biggest drawback with the hybrid detector. The BN and ESN detectors, however, do not misclassify to the same extent, exhibiting the learning algorithms' ability to distinguish between *normal* and *anomalous* data. Compared to each other, the ESN and BN perform equally well.

6.3 Discussion

We have shown, for the modalities we tested, that the BN can detect anomalies with equally high accuracy as the larger and slower ESN without any significant differences in the number of false positives and negatives generated.

We have also shown that both learning based algorithms are capable of detecting low-amplitude anomalies better more effectively than specific rule-based anomaly detectors. At the same time, they are equally effective over multiple anomaly types, due to their ability to detect a wide range of features deviating from the training set.

Nevertheless, the BN's smaller ROM footprint and faster execution time resolve two key issues with ESNs: (1) The size of the 50-node ESN approaches the limit of the TelosB mote's ROM capacity. While the ESN necessary for detecting anomalies in soil moisture and temperature is small enough to fit with the rest of the code, modalities with higher rates of dynamics will require ESNs that do not fit in memory. On the other hand, the BN provides flexibility equivalent to that of a larger ESN, as well as room to expand the data collection application. (2) The prediction time of the ESN is in the order of seconds. For environmental monitoring applications in which changes occur on the scale of minutes this performance is acceptable, but ESNs cannot be used in applications that sample their sensor multiple times a second. BN on the other hand, can do inference on the order of tens of milliseconds.

7. CONCLUSION

This paper unifies fault and event detection in sensor networks under the general framework of anomaly detection. We show that online anomaly detection is feasible on mote-class devices by implementing both an Echo State Network (ESN) and a Bayesian Network (BN) on a TelosB mote. These networks perform as well as their PC-based implementations of the same size, proving that it is feasible to employ sophisticated pattern recognition algorithms on motes. Indeed, both ESN and BN are small and fast enough to function alongside an environmental monitoring application, detecting measurement anomalies in real time. Depending on the amplitude of the injected anomalies, they provide equivalent or higher detection accuracy compared to rule-based detectors customized to specific faults. However, the most significant feature of the ESN and BN detectors are their generality since they are capable of detecting all features not present in the training set. Contrary to our initial intuition, the BN performs as well as the ESN, while being an order of magnitude faster and occupying a fraction of the memory.

In our future work we will explore mechanisms for selecting the best supervised learning method for mote-based online classification given a particular training set provided by the domain scientists.

REFERENCES

Bai, Z., Demmel, J., Dongarra, J., Ruhe, A., and van der Vorst, H. 2000. *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide.* SIAM.

Bokareva, T., Bulusu, N., and Jha, S. 2006. Learning Sensor Data Characteristics in Unknown Environments. In *IWASN.*

Chang, M., Terzis, A., and Bonnet, P. 2009. Mote-Based Online Anomaly Detection using Echo State Networks. In *DCOSS.*

Cornou, C. and Lundbye-Christensen, S. 2008. Classifying sows' activity types from acceleration patterns. *Applied Animal Behaviour Science 111,* 3-4, 262 – 273.

Dagum, P. and Luby, M. 1993. Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence 60,* 1, 141 – 153.

Gupchup, J., Sharma, A., Terzis, A., Burns, R., and Szalay, A. 2008. The Perils of Detecting Measurement Faults in Environmental Monitoring Networks. In *DCOSS.*

Herbert Jaeger. Matlab toolbox for ESNs. Available at `http://www.faculty.jacobs-university.de/hjaeger/pubs/ESNtools.zip` Last checked: 2008-08-31.

Hu, W., Tran, V. N., Bulusu, N., Chou, C. T., Jha, S., and Taylor, A. 2005. The design and evaluation of a hybrid sensor network for cane-toad monitoring. In *IPSN*.

Jaeger, H. 2001. The Echo State Approach to Analysing and Training Recurrent Neural Networks. Tech. Rep. GMD Report 148, German National Research Center for Information Technology.

Jaeger, H. 2002. Tutorial on Training Recurrent Neural Networks, Covering BPPT, RTRL, EKF and the Echo State Network Approach. Tech. Rep. GMD Report 159, German National Research Center for Information Technology. October.

Kaplantzis, S., Shilton, A., Mani, N., and Sekercioglu, A. 2007. Detecting Selective Forwarding Attacks in WSN Using Support Vector Machines. In *ISSNIP*.

Kevin Patrick Murphy et al. Bayes Net Toolbox for Matlab. Available at `http://people.cs.ubc.ca/~murphyk/Software/BNT/bnt.html` Last checked: 2009-09-12.

Mackey, M. C. and Glass, L. 1977. Oscillation and Chaos in Physiological Control Systems. *Science 197,* 287.

MANA. Monitoring remote environments with Autonomous sensor Network-based data Acquisition systems. `http://www.itu.dk/mana/`.

Marra, S., Iachino, M., and Morabito, F. 2006. Tanh-like Activation Function Implementation for High-performance Digital Neural Systems. *Research in Microelectronics and Electronics 2006, Ph. D.*, 237–240.

Moteiv Corporation. Tmote Sky. `http://www.moteiv.com/`.

Müller, B., Reinhardt, J., and Strickland, M. T. 1995. *Neural networks (2nd ed.): an introduction*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Musăloiu-E., R., Terzis, A., Szlavecz, K., Szalay, A., Cogan, J., and Gray, J. 2006. Life Under your Feet: A WSN for Soil Ecology. In *EmNets Workshop*.

Obst, O., Wang, X. R., and Prokopenko, M. 2008. Using Echo State Networks for Anomaly Detection in Underground Coal Mines. In *IPSN*.

Omitaomu, O. A., Fang, Y., and Ganguly, A. R. 2008. Anomaly Detection from Sensor Data for Real-time Decisions. In *Sensor-KDD*. Las Vegas, Nevada, USA.

Osborne, M. A., Roberts, S. J., Rogers, A., Ramchurn, S. D., and Jennings, N. R. 2008. Towards Real-Time Information Processing of Sensor Network Data Using Computationally Efficient Multi-output Gaussian Processes. In *IPSN '08: Proceedings of the 7th international conference on Information processing in sensor networks*. IEEE Computer Society, Washington, DC, USA, 109–120.

Pister, K. 2001. Tracking vehicles with a UAV-delivered sensor network. Available at http://robotics.eecs.berkeley.edu/ pister/29Palms103/.

Polastre, J., Szewczyk, R., and Culler, D. 2005. Telos: Enabling Ultra-Low Power Wireless Research. In *IPSN/SPOTS*.

Rashidi, P. and Cook, D. J. 2008. An Adaptive Sensor Mining Framework for Pervasive Computing Applications. In *Sensor-KDD*. Las Vegas, Nevada, USA.

Römer, K. 2006. Distributed Mining of Spatio-Temporal Event Patterns in Sensor Networks. In *EAWMS at DCOSS*.

Russell, S. and Norvig, P. 2003. *Artificial Intelligence A Modern Approach, 2nd Ed.* Prentice Hall.

Selavo, L., Wood, A., Cao, Q., Sookoor, T., Liu, H., Srinivasan, A., Wu, Y., Kang, W., Stankovic, J., Young, D., and Porter, J. 2007. LUSTER: Wireless Sensor Network for Environmental Research. In *ACM SenSys*.

Sharma, A., Golubchik, L., and Govindan, R. 2007. On the Prevalence of Sensor Faults in Real-World Deployments. *IEEE SECON*.

Tolle, G., Polastre, J., Szewczyk, R., Turner, N., Tu, K., Buonadonna, P., Burgess, S., Gay, D., Hong, W., Dawson, T., and Culler, D. 2005. A Macroscope in the Redwoods. In *ACM SenSys*.

Wang, L. and Fu, X. 2005. *Data Mining with Computational Intelligence (Advanced Information and Knowledge Processing)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

WANG, X. R., LIZIER, J. T., OBST, O., PROKOPENKO, M., AND WANG, P. 2008. Spatiotemporal Anomaly Detection in Gas Monitoring Sensor Networks. In *EWSN*. 90–105.

WEISSTEIN, E. W. Normal Distribution. From MathWorld–A Wolfram Web Resource at `http://mathworld.wolfram.com/NormalDistribution.html` (Sep 2009).

WERNER-ALLEN, G., LORINCZ, K., JOHNSON, J., LEES, J., AND WELSH, M. 2006. Fidelity and yield in a volcano monitoring sensor network. In *OSDI*.

WOLFRAM RESEARCH, I. Quantile-Quantile Plot. `http://mathworld.wolfram.com/Quantile-QuantilePlot.html`.

WU, E., LIU, W., AND CHAWLA, S. 2008. Spatio-Temporal Outlier Detection in Precipitation Data. In *Sensor-KDD*. Las Vegas, Nevada, USA.

Received Month Year; revised Month Year; accepted Month Year

Chapter 5

Adaptive Data Acquisition

With the resource normalization methodology from Chapter 3 and the anomaly detection framework from Chapter 4 we have established two of the three fronts in our move towards scientific sensornets.

These two cornerstones also serve as building blocks for our final front, namely an AI-based controller to mediate between the domain scientist and the sensornet. Here, the resource normalization methodology ensures that the controller can reason about whether or not a set of actions are feasible and efficient, in terms of available time and energy, and the anomaly detection framework enables the controller to adapt to changes in the environment. In other words, with this controller we are able to perform adaptive data acquisition instead of only automated.

In "Meeting Ecologist Requirements with Adaptive Data Acquisition" we present *ADAE*, our system to bridge the gap between ecologists and sensornets. We reject the notion of yield as benchmark, and let the ecologist control the sensornet by stating requirements based on sparseness (e.g., least acceptable amount of measurements pr. unit time or space). Our system continuously retasks a sensornet based on detected anomalies in the measured data and external sources, thereby adapting to the changing environment as a scientist in the field would do. We base our work on the *Planning and Scheduling* [5] architecture known from the Artificial Intelligence community. This paper has been submitted to the *ACM/IEEE International Conference on Information Processing in Sensor Networks, 2010* and is currently under review. The work was done in collaboration with Peter Stæhr and Philippe Bonnet.

Our work is closely related to the soil monitoring sensornet *Suelo* by Ramanathan et al. [4] which is also driven by online detection algorithms. However, while their main objective is to detect faults with the purpose of alerting a human technician, our system adapts to detected anomalies based on predefined requirements from the domain scientist and thus avoids human intervention as long as possible.

These predefined requirements are associated with different collection modes, each reflecting a different purpose and strategy. Similar to *Levels* by Lachenmann et al. [3], energy consumption and expected deployment time play a key role in choosing between collection modes, however, instead of choosing between mutually exclusive modes based on energy consumption and lifetime expectations, our system combines valid collection modes in order to maximize a utility function while still fulfilling the energy and time constraints.

This utility function is similar to the one used in *Lance* by Werner-Allen et al. [6], where a scoring function assigns utility to measurements based on their estimated scientific value, and the measurements with the highest score are forwarded through the sensornet first. Our scoring function works orthogonally to this, and is used to estimate the outcome of different collection strategies before measurements are even sampled with the purpose of choosing collection modes with the highest combined utility.

Changing data collection strategies based on the available measurements is similar to the adaptive sampling used in *NIMS* by Borgstrom et al. [1], where measurements are chosen based on a model in order to increase resolution and decrease uncertainty. We use adaptive sampling to maintain consistency and to increase utility in a changing environment.

The *Planning and Scheduling* architecture is closely related to *Control Theory* [2] known from engineering and mathematics, which employs a controller to ensure that the actual performance of the system is as close to the desired performance as possible. This is done through a feedback loop that continuously changes the system's parameters in order to match performances. We see these two approaches as orthogonal, where the former is controlling *what* the performance should be, while the latter is controlling *how* to achieve it.

Bibliography

[1] P. Borgstrom, A. Singh, B. Jordan, G. Sukhatme, M. Batalin, and W. Kaiser. Energy Based Path Planning for a Novel Cabled Robotic System. In *IEEE IROS*, 2008.

[2] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury. *Feedback Control of Computing Systems.* John Wiley & Sons, 2004.

[3] A. Lachenmann, P. J. Marrón, D. Minder, and K. Rothermel. Meeting lifetime goals with energy levels. In *SenSys '07: Proceedings of the 5th international conference on Embedded networked sensor systems*, pages 131–144, New York, NY, USA, 2007. ACM.

[4] N. Ramanathan, T. Schoellhammer, E. Kohler, K. Whitehouse, T. Harmon, and D. Estrin. Suelo: Human-assisted Sensing for Exploratory Soil Monitoring Studies. In *SenSys '09: Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, pages 197–210, New York, NY, USA, 2009. ACM.

[5] S. Russell and P. Norvig. *Artificial Intelligence A Modern Approach, 2nd Ed.* Prentice Hall, 2003.

[6] G. Werner-Allen, S. Dawson-Haggerty, and M. Welsh. Lance: optimizing high-resolution signal collection in wireless sensor networks. In *ACM SenSys*, pages 169–182, New York, NY, USA, 2008. ACM.

Meeting Ecologist Requirements with Adaptive Data Acquisition

Marcus Chang
Department of Computer Science
University of Copenhagen
marcus@diku.dk

Philippe Bonnet
IT University of Copenhagen
phbo@itu.dk

Abstract

Ecologists instrument ecosystems with in-situ sensing to collect measurements. Sensor networks promise to improve on existing data acquisition systems by interconnecting standalone measurement systems into virtual instruments. Such ecological sensor networks, however, will only fulfill their potential if they meet the scientists requirements. In an ideal world, an ecologist expresses requirements in terms of a target dataset, which the sensor network then actually collects and stores. In fact, failures occur and interesting events happen making uniform, systematic ecosystem sampling neither possible nor desirable. Today, these anomalous situations are handled as exceptions treated by technicians that receive an alert at deployment time. In this paper, we detail how ecological sensor networks can adapt to anomalies and maximize the utility of the collected datasets. More specifically, we present the design of a controller that continuously maintains its state based on the data obtained from the sensor network (as well as external systems), and configures motes with parameters that satisfy a constraint optimization problem derived from the current state. We describe our implementation, discuss its scalability, and discuss its performance in the context of a case study.

Categories and Subject Descriptors

I.2.8 [**Computing Methodologies**]: Artificial Intelligence—*Problem Solving, Control Methods, and Search*
; C.2.4 [**Computer Systems Organization**]: Computer-Communication Networks—*Distributed Systems*

General Terms

Design

Keywords

Autonomous System, Constraint Optimization Problem, Planning, Scientific Data, Wireless Sensor Networks

1 Introduction

For years, ecologists have deployed in-situ sensing infrastructures to observe and monitor the biotic and abiotic factors in a given ecosystem. They primarily rely on fixed data loggers to collect and store data from a wide variety of sensors. They have been promised that low power wireless sensor networks would be able to provide them with sampling at unprecedented scale and resolution [10]. However, the MEMS revolution has not yet delivered a radical change of the optical, biological and chemical sensors that are pervasive in ecological monitoring, and scientists cannot afford high density deployment of the current generation of sensors, which are still bulky, energy hungry and expensive. Still, low power wireless networks can have a tremendous impact on ecological monitoring by transforming stand-alone devices into a networked system that is monitored and controlled to meet the scientists requirements. In this paper, we study how ecological sensor networks can be steered to improve the utility of the collected datasets.

Ecologists rely on in-situ sensing to collect datasets of the form $(t,x,y,z) \rightarrow (v_1,v_2,...,v_n)$, where the independent variables represent time (t) and space (x,y,x), and the dependent variables correspond to the modalities of the sensors deployed. These raw measurements are the foundation of the scientific workflow. They are tagged with metadata, and transformed into derived data products via calibration, verification, or extrapolation processes. The derived data products are then used for modeling purposes. The derivation processes and the models are applied in the lab, as a post-processing phase, based on the primary data collected in the field. If an offline verification process exposes a sensor failure then the collected data is useless. If a model gives evidence of interesting events, then the collected data might not be dense enough (in space, time or modality) to allow a deep analysis of the phenomenon. In this paper, we propose to move portions of the existing offline scientific processes online, within the ecological sensor networks in order to improve the quality of the collected data. Specifically, we propose that anomalous situations should be recognized and handled online, while data is collected, so that the sensor network can adapt and maintain high utility.

Consider a scientist that monitors a lake. She is interested in measuring conductivity and temperature at five different depths, at a sampling rate of one measurement per hour, for a month. This is her initial requirement based on the dataset she wishes to collect. However, if we go further and consider potential anomalous situations, we obtain a much more complete picture:

- *Failures*: She can tolerate that measurements are taken up to once every six hours; however below that threshold, measurements are useless. Also, she requires that both conductivity and temperature are measured together; if either measurement is missing the other is useless. She indicates a valid range for conductivity and temperature measurements; measurements outside these ranges should be considered errors. Conductivity errors might be compensated by either repeating a measurement within a few seconds, and if that fails resetting the conductivity sensor. Temperature errors might be compensated by looking up the temperature at an adjacent depth. In addition, a sensor should not be considered damaged if its measurements drift in time; regular manual samples are taken periodically to compensate for such errors.
- *Interesting events*: The scientist indicates that she is interested in thermoclines (rapid changes in the temperature within a limited depth range) - so if possible, measurements should be taken at additional depths if a thermocline is detected (given a simple temperature variation threshold for detecting thermoclines). Also, in case of a storm (signaled by the RSS feed of a close-by weather station), the sampling rate should be increased to twelve measurement per hour for the two depths that are closest to the surface of the lake. The scientists notes, however, that if energy is limited the baseline measurements should have priority.

The core of the problem is that ecological data acquisition has been based on the premise of systematic ecosystem sampling: it is assumed that the ecosystem is sampled with given modalities at predefined intervals in time and space. This is neither possible (because of failures), nor desirable (because interesting events might not be captured by the baseline settings). In contrast, we propose that ecological sensor networks should rely on adaptive ecosystem sampling, where the procedure for selecting samples may depend on the values of observed variables [26]. More specifically, we propose an ecological sensor network controller that checks the measurements it collects and adapts how the next measurements should be obtained (in time, space and modality) to maximize their utility for the scientists [25]. Now, the questions are: (1) How can scientists represent the utility of measurements? (2) How can such a controller operate to maintain high utility at reasonable cost in a changing environment? We address these questions in this paper. Our contribution is the following:

1. We capture the scientist requirements in terms of data collection modes. For each data collection mode, the scientist describes a range of acceptable parameters. Utility is represented as a ranked preference of these data collection modes.
2. We describe a controller that continuously maintains its state based on the data obtained from the sensor network (as well as external sources), and configures motes with parameters that satisfy a constraint optimization problem derived from the current state.
3. We adopt a three-tier architecture, developed for autonomous systems, in the context of a sensor network controller.
4. We detail the implementation of the ADAE system based on this design, and discuss how it scales.
5. We describe a case study based on an actual deployment for lake monitoring.

2 Related Work

In this section, we look back on the evolution of data acquisition based on sensor networks, we discuss the previous use of adaptive sampling in sensor networks, and we review existing work on autonomous systems controllers.

2.1 Data Acquisition with Sensor Networks

Cougar [3] and TinyDB [19] introduced a distributed query processing paradigm for sensor network data acquisition. The goal was to ensure a flexible tasking of motes via a relational query interface. The assumption was that (a) the relational model was appropriate to capture sensor data, (b) that users would submit queries to task motes, and (c) that in-network processing was necessary in the context of a sensor network. The relational query interface is not a good abstraction for ecologists that do not wish to query the sensors but aim at systematically collecting primary data sets for their scientific processes (see [20] for a thorough discussion of the limitations of these approaches).

BBQ [16] and MauveDB [8] introduced the notion of *model-based querying* as an abstraction for data acquisition. The system maintains a statistical model of the data, and instead of blindly collecting time series, only collects the data that are needed to improve the precision of the model. For instance, correlations across modality are leveraged to reduce the cost of data collection as expensive measurements are replaced by cheaper ones. Users obtain probabilistic, approximate answers to their queries. Such approaches are not relevant for ecologists since they are the ones discovering new models and thus need primary datasets as a foundation for their scientific processes.

PRESTO [18] further develops the idea of model-based querying. The PRESTO gateway constructs a seasonal ARIMA model of the time series collected at a given sensor. In order to maintain these models, PRESTO combines the pull approach from MauveDB (data is collect as needed to improve precision), with a push approach, where each sensors use the model parameters defined by the gateway to predict future values and sends data to gateway in case an anomaly is detected (i.e., there is a significant difference between a predication and the actual measurement). The gateway refines the sensor model as it receives new measurements to reflect changes in the sensed data. We share with PRESTO this focus on anomaly detection and on adaptation to a changing environment. PRESTO returns approximate answers that match the confidence interval specified by users. The rationale behind the design of PRESTO is to improve energy efficiency, not to maximize utility for users.

Lance [27] introduced utility-based controllers in the context of sensor network data acquisition. This system focuses on the collection of high-bandwith signals, where not all the data acquired by the motes can be transmitted to the base station. Lance controls bandwidth usage by splitting the data acquired at each mote into a sequence of data packets, and making sure that only the most relevant data packets are transferred back to the base station. The selection is performed by the base station based on summaries sent by motes and on a trade-off between cost and utility provided by the user. We share with Lance a focus on optimizing the utility of the collected data. The fundamental difference is that in Lance the optimization problem concerns the deletion of data collected in a predefined way, while in ADAE the optimization problem concerns the selection of the data collection parameters (e.g., sampling rate, sensor placement, modality). Those two problems are orthogonal. In terms of architecture, Lance focuses on flexible policy modules, while ADAE relies on the three-tier architecture – both aspects are complementary. In the rest of the paper, we assume that all data collected at the motes can be transmitted to the gateway.

2.2 Adaptive Sampling in Sensor Networks

In statistics, adaptive sampling designs are those in which the selection procedure may depend sequentially on observed values of the variable of interest [26]. In the context of sensor networks, adaptive sampling has been introduced to (a) maintain high resolution while covering large regions of space using mobile sensors, e.g., light sampling with Networked Infomechanical Systems (NIMS) [4], or to (b) reduce approximation errors with additional samples taken by mobile sensors, e.g., weather forecasting with autonomous UAVs [7]. Compared to these approaches, we do not seek to improve resolution with a reduced number of sensors, but to maintain utility of measurements in a changing environment. Our challenge is to take a decision on when, where or how to sample whenever the environment changes, rather than gradually improve the resolution of a given model.

2.3 Autonomous Systems

Autonomous systems constitute a popular research topic in the areas of AI and robotics. The most interesting developments have been achieved in the area of autonomous controllers, with contributions ranging from the seminal work on Deep Space 1 [9] to the Mars Rover [1]. An architecture for autonomous systems has emerged [2] based on the following three tier architecture: the bottom tier is the functional layer that is the interface with sensors and actuators, the middle tier executes the planned actions and check their effects, and the top tier implements the planning and scheduling functionalities. As we discuss in Section 4.3, we adopt a similar architecture for the ADAE system. Note that NASA has now made publicly available the platforms they developed for their autonomous systems, e.g., Apex [12] or Europa [11]. We did not use these systems because they did not support the type of solver we envisaged for our planner, and because implementation constraints did not allow us to deploy these systems on our target gateway[1]

[1] Apex relies on multi-threaded Lisp, which was not available on the Linux-based platform we used for our deployment.

3 The Ecologist Requirements

We aim at designing a system that autonomously adapts data acquisition to meet the ecologist's requirements. In this Section, we detail those requirements. Note that our goal is not to define a rigid template for software engineering purpose, but to put a stick in the ground regarding the scientists expectation of an ecological sensor network.

3.1 Data Collection Modes

Ecologists rely on in-situ sensing to collect primary datasets. In the case of manual sampling, they define a protocol that ensure the relevance, quality and consistency of the collected data. In case of automatic sampling, they have to express requirements to the monitoring system. These requirements are based on the description of the target datasets, $(t,x,y,z) \rightarrow (v_1,v_2,...,v_n)$, described in the Introduction, where the time domain defines the sampling rate as well as the lifetime of the deployment, the space domain defines sensor placement, and the dependent variables define sensor modalities and accuracy.

The traditional requirement is that given a dataset description, all data must be stored, i.e., the whole dataset must be collected [21]. The problem with this requirement is twofold. First, it defines an ideal goal. In case of failure, the monitoring system will not be able to deliver the target data set. A consequence is that system designers tend to assume that yield (what percentage of the target dataset is actually collected) is an appropriate metric for system performance. For ecologists however, the relevance of a dataset is not proportional to its yield. In our experience, they identify portions of the collected data set that they can use for modeling purposes, and portions that are useless - typically because the dataset is locally too sparse (in time, space or modality). Second, the requirement of uniform, systematic dataset collection does not account for interesting events. Such events are arguably the most interesting elements of a dataset. Their analysis might require denser sampling in time, space or modality for a limited period of time.

For example, consider the soil monitoring project, "Life Under Your Feet" [21]. This sensor network consists of more than hundred TelosB motes, each equipped with two temperature and moisture sensor probes. These four probes are dug into the soil at specific depths and sampled every ten minutes. This is fast enough to monitor moisture evaporation but not precipitation, which changes within seconds and not minutes. Obviously, continuously sampling at a rate capable of capturing rain events would significantly strain the power supply. However, using external sources such as local weather forecasts and only increase the sampling rate when the chance of a rain event is significant would be far less expensive power wise.

To overcome these limitations, our goal is to (a) capture an envelope of datasets relevant for the ecologists in the context of a given deployment, and (b) a means of representing the scientists preferences within that envelope.

We propose to capture the ecologist's requirements as a ranked list of *data collection modes* (e.g., baseline, degraded, failure, event detection). Some of the modes are exclusive (e.g., baseline and degraded), while others can be active simultaneously (e.g., baseline and failure or event detection). For each collection mode, the ecologists define:

1. A description of the conditions that must be satisfied to activate or deactivate these modes. A condition is specified using a rule (e.g., humidity inside a mote is greater than 50%), a model (e.g., Echo State Network for anomaly detection with a training set specified by the scientist [6]), or a timing constraint (e.g., within five minutes or for five hours).
2. A target dataset, i.e., its time component (lifetime and sampling rate), its space component (sensor location), and its dependent variables (modality and accuracy). Note that, for data collection modes associated to failures, the target data set specifies relevant redundancy in time, space or modality.
3. A sparseness threshold for each modality, i.e., the number (or distribution) of usable measurements per chunk of time and space.

The ranking of the collection modes defines an ordinal utility function. Despite our insistence, none of the scientists we are collaborating with could find a non-trivial cardinal utility in the context of their activity. In addition to these data collection modes, the ecologists define a target *lifetime* for data collection.

For example, we derive from [21], the following requirements for ”Life Under Your Feet”:

- We define the following **data collection modes**: *baseline*, *precipitation*, and *fault* modes (which should be defined with the ecologist).
- The *baseline* is always present, while the **condition** for the *precipitation* event is when the weather forecast predicts rain and the *condition* for the *fault* are humidity in the mote greater than a given threshold, out of bounds measurement.
- The **target dataset** for the *baseline* is the four probes sampled every 10 minutes, for the *precipitation* it is every minute instead. For the fault modes, we would need to identify redundancy in time, space or modality.
- We set the **sparseness threshold** to six set of samples every hour in each data collection mode.

We derived this form of requirements from our collaboration with ecologist. When asked about their requirements all scientists initially defined a single ideal target dataset. When faced with the fact that failures might occur, they came up with a form of sparseness threshold, and the definition of one or several degraded modes. They expressed interesting events characterized by simple conditions (external events or simple thresholds on the sensed data).

3.2 The Case for Autonomous Data Acquisition

The solution promoted in commercial data acquisition systems to tackle failures and anomalous situations consists in involving human supervision. Let us go back to the lake monitoring example from the Introduction. A buoy is deployed equipped with a data logger that stores the data it collects at a predefined sampling rate from the CTD sensors (conductivity, temperature, depth) deployed at five different depths. The data logger is equipped with long-range wireless communication and it acts as a server for telemetry and tele-command, possibly alerting a technician in case of problems and accepting commands and configuration operations. This design, which is the state-of-the-art in ecological data acquisition, is however flawed in several respects:

1. Contingency planning is weak. In case the data logger detects an anomalous situation, it raises an alert and it is up to the technician to handle it. This is a best effort approach, where response to anomalies is unspecified and variable. In our experience, the resources available for monitoring purposes do not allow 24/7 supervision. Because, long-range communication and technician supervision are expensive, the data logger is programmed to send alerts in limited cases. The system is not configured to compensate for errors or to react to interesting events.
2. No graceful degradation. When energy supplies are low, data acquisition continues at the predefined sampling rate at the risk of thrashing. More generally, the assumption is that the system has a single regime, and that human intervention is needed to keep this regime operational in case of failure.
3. The system is stand-alone. Co-located data loggers are not interconnected, thus possibly missing opportunities for increased redundancy, and detection of interesting events.

Our goal with this work is to limit human intervention to the initial requirements, and let an autonomous data acquisition system handle anomalies with a controller that continuously adapts to changes in the environment, and tasks motes to maximize the utility of the measurements[2]. We detail the design and implementation of such a controller in the next Section.

4 The ADAE System

ADAE is an autonomous gateway-based controller that tasks motes to keep on maximizing utility in a changing environment. Before we describe its design, architecture and implementation, we address the following question: Which actions can ADAE take in order to control the sensor network?

4.1 Sensor Network Model

We model an ecological sensor network as a cluster of motes connected to a gateway. We adopt a classical two-tier model [18, 27], where motes are slaves, tasked by the gateway-based controller to sample, store and transmit data. We do not consider any form of in-network aggregation or storage (beyond local computation or storage on the mote that produces data). We further assume a best effort delivery between mote and gateway (e.g., CTP [13]) that allows the gateway to collect routing statistics. Finally, we assume that each mote is appropriately duty cycled (based on the sam-

[2]Obviously, when possible and affordable, human intervention should be used to maintain the optimal regime. Our point, here, is that the system should maintain high utility and degrade gracefully when the optimal regime is no longer sustainable.

pling rate and offload rate), and that it is accessible (using a form of low-power listening [24]).

We also assume that the sample, store and transmit tasks are accomplished by a program deployed on all motes, and that this program can be configured with parameters to modify the sampling or transmission policy. We make this assumption because it allows for a straightforward integration of legacy systems (including the current generation of commercial motes). Leveraging rich mote APIs or mote reprogramming (via tasklet distribution [14] or full image reprogramming [15]) is an issue for future work.

We introduce virtual sensors to abstract the details of the actual motes[3]. Each virtual sensor represents a modality of a given mote (we describe virtual sensors in more detail below). Virtual sensors export a single API function, that defines the space of possible controller actions (note that such actions must be mapped to the API exported by the actual motes):

- $configure(VS, SR, TR)$ to configure the sampling rate (SR) and transmission rate (TR) on a given virtual sensor (VS).

4.2 Controller Design

The key questions that we need to address are: (a) What is an appropriate abstraction of the sensor network?, (b) How to represent user requirements, i.e., data collection modes and utility?, (c) How to define cost?, and (d) How to plan a sequence of actions given the controller state and the user requirements.

4.2.1 Virtual Sensors

Conceptually, the following relations can be used to organize the state variables representing a sensor network:

```
VirtualSensor(VS_id, Modality, Mote_id,
       X, Y, Z, SampleTime, SampleEnergy,
       TransmitTime, TransmitEnergy)
VirtualSensorState(VS_id,
       SamplingRate, TransmitRate)
Mote(Mote_id, BatteryCapacity, VB_id,
       Forward, Overhear)
Topology(ParentMote_id, ChildMote_id)
VirtualBattery(VB_id, LifeTime, Percentage,
       NbInstallments, Credit, MaxBurnRate)
VirtualBatteryState(VB_id, Balance,
       InstallmentsDone, MaxOverDraft)
```

Each virtual sensor is identified by a VS_{id} and represents the $Modality$ of a physical sensor attached to mote $Mote_{id}$, at a fixed location X, Y, Z[4]. A calibration phase defines the time and energy it takes to make a measurement with the given modality (*SampleTime*, *SampleEnergy*)[5], as well as the time and energy required to transmit one measurement (*TransmitEnergy*, *TransmitTime*). Note that such normalized time and energy attributes correspond to the notion of platform vector introduced in [17]. The *VirtualSensor* relation is configured at deployment time and remains unchanged thereafter.

Virtual sensors are configured with two data acquisition parameters: the sampling rate (*SamplingRate*), and the transmission rate (*TransmissionRate*)[6].

For each mote, we store the capacity of the battery it contains (*BatteryCapacity*) as well as the reference of a virtual battery VB_{id}, which is the abstraction [5] that we rely on to reason about energy allocation. A calibration phase allows to define for each mote, the cost of forwarding or overhearing a measurement. We use a simple representation of the collection routing tree using the topology relation. Note that our assumption here is that the topology observed at a given time is a good predictor of the topology in the subsequent epoch. Obviously, we do not capture network dynamics with this model, but this is not our goal. Our goal is to represent topology and transmission costs to estimate the cost of a data acquisition plan (see the discussion of our cost model below).

Each virtual battery is characterized by the $Percentage$ of the total capacity associated to sampling, forwarding or overhearing (with a given mote). Virtual batteries are further specified with a $LifeTime$ requirement (given by the user), a total number of installments ($NbInstallments$), a $Credit$ rate that allows to specify an energy allocation policy ($Credit$ is a real number in the interval [0, 1], e.g., 0 corresponds to a conservative policy that only grant energy installments when there is an energy surplus, while a positive credit rate corresponds to a policy that allows energy deficit up to $Credit$), and a maximum allowed energy burn rate $MaxBurnRate$. The virtual battery state relation is used to maintain the actual energy $Balance$, the number of $Installments$ already received and the maximum overdraft allowed ($MaxOverDraft$ which is a negative number).

4.2.2 Physical Limitations

The controller maintains a set of constraints over virtual sensors that reflect the actual limitations of the physical system. Those constraints concern the range of possible values for the location parameters (X, Y, Z), the sampling rate (*SamplingRate*), and the transmission rate (*TransmissionRate*). In addition, a physical mote is represented as several virtual sensors, one for each modality. We capture the serial or parallel constraints that exist between co-located virtual sensors, in terms of location and in terms of timing of the measurements. In order to represent the timing constraints, we do not need the whole power of the event calculus [25], we just need to represent constraints on serial or parallel executions. We thus introduce variables $t_{i,j}$ that represents the time required before VS_j can take a measurement after VS_i and use integer constraints to represent these timing constraints. To sum up, we represent the physical limitations of the sensor network using two types of constraints: (1) domain restrictions, and (2) integer constraints based on virtual sensors variables.

[3]Our notion of virtual sensor is inspired by Franklin et al. [20]

[4]For static sensors, (X, Y, Z) are given at deployment time, while for mobile sensor a valid range and possibly constraints are given for these variables.

[5]Note that we rely on constraints to indicate the dependencies that may exist between modalities on a same physical sensor.

[6]Note that we force transmissions to be triggered by a time-based condition (the transmission rate) instead of a more general form of condition (e.g., transmit when mote storage is half full) and thus sacrifice flexibility for predictability.

4.2.3 *Data Collection Modes*

To represent data collection modes, the controller maintains:

- A set of predicates $\mathcal{P}$ to represent the conditions that activate and deactivate the given data collection modes. The controller implements the rule-based, model-based or time-based methods specified by the users to evaluate these predicates.
- Specific constraints imposed by the sparseness threshold for the given data collection modes. These constraints are expressed as restrictions of the state variables domains $\mathcal{D}$ (e.g., $X \in [1..100]$).

4.2.4 *Utility Model*

We base our controller on the principle of maximum expected utility [25]. Each action taken by the controller configures motes in conformance with one of the data collection modes described by the ecologist. We associate a cardinal utility to each action, based on the ranked preference among the resulting data collection modes. This utility function is a simple scoring function with uniform spacing (for N modes, the scoring function is such that the top ranked mode gets a score of N, and the bottom ranked mode gets a score of 1). Using the binomial distribution, we model the probability of success for a configuration as the probability of collecting a number of samples higher than the sparseness threshold: $1 - \frac{1}{\binom{SamplingRate}{SparsenessThreshold}}$ (where both $SamplingRate$ and $SparsenessThreshold$ are defined in numbers of samples for a given epoch Δ). The sparseness threshold might be defined for several modalities (i.e., several virtual sensors) within a given data collection mode, so we select the lower probability and multiply it by the rank to obtain the expected utility for that data collection mode.

4.2.5 *Cost Model*

The controller associates a cost to each virtual sensor configuration, based on the energy used to sample, transmit and overhear measurements. We adopt a variation of the cost model introduced in Lance [27], and represent the cost δ_j of a virtual sensor configuration VS_i per virtual battery VB_j. The source mote to which VS_i is associated incurs a sampling and transmit cost, while motes on the communication path incur a forwarding cost, and motes one hop away from the communication path incur a overhearing cost. For a given period of time T,

- Sampling and transmission cost on the source virtual sensor is estimated as $(SampleEnergy_i + TransmitEnergy_i) * (SamplingRate_i * T)$, i.e., the product of the energy cost of obtaining a measurement with the number of measurements in the period.
- Forwarding cost is estimated as $Forwarding_j * (SamplingRate_i * T)$.
- Overhearing cost is estimated as $Overhear_j * (SamplingRate_i * T)$. It is associated to the transmission virtual batteries of all physical motes in the neighborhood of the forwarding motes.

We introduce integer constraints derived from the virtual battery energy allocation model: for a given time period T, (1) the balance is greater than the maximum overdraft ($Balance - \delta_j > MaxOverDraft$), and (2) the amount of energy spent by VS_i is bound by the maximum burn rate ($\delta_j \leq MaxBurnRate_i * T$).

4.2.6 *Planning Problem*

Now, the question is: How does the controller pick appropriate actions given its current state? Because the controller operates in a changing environment, it needs to proceed online, i.e., select some actions at one point in time and evaluate their impact regularly, possibly selecting new actions to react to a change in the environment. We call epoch, noted Δ, the period of time after which a given action is reevaluated (note that our cost model and utility function are defined for limited time frames). A default epoch size is given as a system parameter. Note that an epoch is shorter than the default, in case a data collection mode predicate requires it (e.g., the actions following a failure might be valid/relevant only for a short period of time). We impose a constraint that the period corresponding to the transmit rate is lower than (or equal to) the epoch Δ.

For each epoch, virtual sensors have a fixed configuration (i.e., fixed location, fixed SR and TR). The actions generated, for a given epoch, are thus a collection of at most one API call per virtual sensor. The planning problem is thus reduced to a constraint optimization problem, where the controller must find values of the state variables that satisfy all the constraints, maximize expected utility and minimize cost: ($\mathcal{V}$, $\mathcal{R}$, $\mathcal{C}$, $\mathcal{U}$), where $\mathcal{V}$ represent variables (i.e., all the attributes from the virtual sensor relations), $\mathcal{R}$ are the restrictions on these variables (either given by the system model, the cost model or the user requirements), $\mathcal{C}$ are the constraints (i.e., physical limitations, or virtual battery constraints) and $\mathcal{U}$ is the expected utility. The size of the search space grows exponentially with the number of virtual sensors $O(sr\_range \cdot 2^N)$, where N is the number of virtual sensors and sr_range is the average size of the $SamplingRate$ domains

4.3 System Architecture

Our controller needs to address three sub-problems:

1. How to update the controller state?
2. How to generate the appropriate constraint optimization problems when appropriate?
3. How to solve the given constraint optimization problems?

In order to tackle these problems, we structure our controller using the classical three-layer architecture developed for AI planning [2]:

- Functional Layer, which provides abstractions for the motes, the sensor tasks, the storage subsystem, and the detection modules. Its interface is generic, but its implementation is deployment-specific.
- Executive Layer, which checks the collected data, call the decision layer if a new plan is needed, and transmits the plans from the decision layer to the functional layer. Both its interface and implementation are generic.
- Decision Layer, which produces a new plan based on the data it gets from the executive layer. The deci-

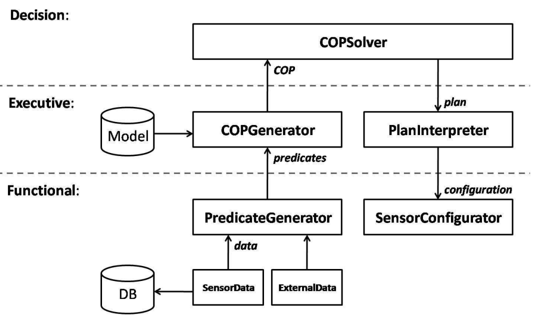


Figure 1. Architectural overview of ADAE.

sion layer is composed of a generic solver and of a deployment-specific model.

The flow of information in the individual components in this three-layer architecture is illustrated in Figure 1. Data generated from the sensor network (both measurements and network status) is collected by **SensorData**. This data is stored in a local database, and passed along to the upper layers of the controller. We use virtual sensors to present a uniform abstraction to the upper layers of the controller. One issue, though, is to map the data received from actual motes into data associated to virtual sensors. This mapping is straightforward for stationary sensors since there is a direct one-to-one mapping between virtual sensors and the modality of a mote at a given location. Mobile sensorsOn the other hand, have a one-to-many mapping, were each distinct location of a mobile sensor corresponds to a different virtual sensor. The data associated to virtual sensors is then passed on to **PredicateGenerator**. Information from external sources, such as weather forecasts and time and date specific events are collected by **ExternalData**. This data is passed on to **PredicateGenerator**.

In **PredicateGenerator**, detection and mapping algorithms are used to transform the time series, network status, and external data into predicates. In terms of architecture, one or several detection modules are attached to each virtual sensor. For example, the range of each measurement value can be checked and if some are found to be out-of-bounds the *OutOfBounds* predicate is set to true. The conditions described by the ecologist are also checked at this point with each condition generating its own predicate.

These predicates are passed on to **COPGenerator** where they are used to represent the current state of the system. The role of this component is twofolds. First, it maintains the state of the virtual sensor and virtual batteries. Second, it constructs a COP that reflects this state, and incorporates the constraints as well as utility function from the set of data collection modes activated by the predicates that evaluate to true. Note that we generate a single COP for the entire network in order to account for forwarding and overhearing costs.

This COP is then passed on to **COPSolver** which tries to find a sensor configuration that satisfies all the constraints of the COP and at the same optimizes the expected utility and minimize cost. In ADAE, we model our COP using the MiniZinc [22] constraint programming language which allows us to define our COPs at a high level of abstraction. This gives us the flexibility to switch between different engines depending on performance and platform availability. The solving of the COP is accomplished in two-steps. First, the COP formulated in MiniZinc is translated to the FlatZinc language, a lower level constraint programming language. Second, a generic solver with a FlatZinc parser is used to solve the COP. The downside of using a generic language such as MiniZinc is the added overhead from the intermediate step and the missed opportunity to leverage solver specific performance enhancements, i.e., specific API calls.

The resulting plan is passed on to **PlanInterpreter** where a configuration is generated for each mote and using **SensorConfigurator** each mote in the network is reconfigured. Similar to the mapping process in **SensorData**, the configurations for the virtual sensor abstraction are transformed into commands and configurations specific to the physical sensor network. Virtual sensors corresponding to sensors with fixed locations are mapped directly, while for virtual sensors representing mobile sensors, the robot carrying the sensor is instructed to follow a path connecting the virtual sensors. A cache of all the current configurations are kept and motes are only reconfigured if there are any changes. Whenever possible, **SensorConfigurator** requires that motes piggyback their energy status on the data they transmit. Such energy status are identified by **SensorData** and used by **PredicateGenerator** to update the virtual battery state.

5 Evaluation

The key question from a performance point of view is whether our approach based on generating and solving Constraint Optimization Problems is viable, specially in a multi-hop setting with a cluster of 40-50 motes. This is the question we address in this Section. Our implementation of ADAE is publicly available[7] and based on the standard C++ library to ensure portability. We only use the MiniZinc-to-FlatZinc translator provided by [22] and not the corresponding FlatZinc solver. Instead we use the Gecode solver with

[7] http://code.google.com/p/adae/

the FlatZinc interface[8], since it has better performance, supports are wider range of platforms, and allows a more controlled search process. All benchmarks are run on an Intel Core 2 T7600 2.33 GHz processsor.

5.1 Sanity Check

5.1.1 Constraints

Because we are considering Constraint Optimization Problems, we expect the resource constraints (i.e. time and energy) to have a significant dual impact on the search space. On one hand, tight resource constraints will limit the search space by rendering certain states inaccessible, and thus reduce the runtime. On the other hand, loose resource constraints will make even the high utility states accessible, giving the full benefit of the optimization directed search. We thus expect the search space to be largest when the resource constraints are neither restrictive enough to render a significant portion of the state space inaccessible nor loose enough to make the states with the highest utility available.

Of course, this only holds if the cost/benefit relation between time/energy and utility is positive, i.e., states with higher utility requires more resources than states with lower utility. With a negative relation, tight resource constraints would lead to the benefits of both a small state space and a optimization directed search, while a loose constraint would have neither. For the remainder of this Section we choose a positive relation since this seems most applicable, i.e., higher cost yields higher utility.

In Figure 2 we show the runtime for three different COPs, with varying energy constraints. Because the time constraint are modeled completely analog we only consider the energy. The number of motes and available sampling rates are all fixed at one for all three problems.

The energy constraints are set as a fraction of the maximum energy required for the most demanding state, since this depends on the number of virtual sensors. As expected, there is a significant difference in runtime when the constraints are varied. Specifically, there is a difference of three orders of magnitude between the COPs with no energy constraints (100%) and the ones with exactly half available (50%). This confirms our initial analysis that the search space is largest when neither the constraints nor the optimizations can be used to minimize the search space significantly.

5.1.2 Virtual Sensors

We know from Section 4.2.6 that the size of the state space grows exponentially with the number of virtual sensors and linearly with the number of available sampling rates. On the other hand, the number of motes only effects the shape of the state space. We thus expect the former to have a significant impact on runtime while the contribution from the latter will mostly be overhead from book keeping.

We explore the scalability with regards to the number of virtual sensors in Figure 3. As before, we keep the number of motes and sampling rates fixed at one. With the new information above, we set the energy constraint to 50% of the highest energy state in order to explore the largest search space. As expected, the runtime grows exponentially with the number of virtual sensors (note the logarithmic scale).

5.1.3 Sampling Rates

Next we explore the impact of the size of the sampling rate domain. Again, we keep the number of motes fixed at one and set the energy constraint to 50% of the highest energy state. We expect from Section 4.2.6 that the runtime grows linearly with the sampling rate domain size, which is also the case as can be seen in Figure 4.

5.1.4 Motes

In Figure 5 we show the run time as a function of increased number of motes.

Not surprisingly, increasing the number of motes does not have a significant impact on runtime, with only a small linear addition when adding motes. The reason why motes add a small amount of overhead lies in the way cost and utility are calculated in the model: there is an intermediate calculation step for each mote.

5.2 Constraining Runtime

In the previous experiments, the runtimes we measured have all been for exhaustive searches. However, with an exponential state space we have no guarantees that the search space will be traversed in a timely manner.

Although our goal is the optimal solution, any assignment that satisfies our COP will of course also satisfy the ecologist's requirements. Hence, any solution will be tolerable although one with higher utility is obviously preferred.

Thus, we explore the quality of the intermediate solutions (if any) that the solver discovers during each search when subject to a hard upper bound on the runtime. We turn our attention back to the problems from Section 5.1.2 in order to compare both the constrained runtime with the exhaustive runtime and the intermediate utility with that of the optimal one. However, even with the lowest runtime we could enforce on the search, 15 ms, the solver was able to find a solution with a utility identical to the optimal one. Even when we increased the number of virtual sensors to 40, we still obtained the optimal solution as one of the first solutions where an exhaustive search would have taken more than 20 days.

The reason for this surprising result is that although the problems we seek to solve have an exponentially large search space, the solutions themselves are rather simple; and the reason why the solver requires an exhaustive search to find the optimal solution is because we use a generic one that can only be given general search directions and not a problem specific solver with detailed knowledge of the system's dynamics.

In our case, the general search directions we use are to first establish the optimal sampling rate, by searching in ascending order, and next determine which virtual sensors to include, starting with them all. This strategy favors the localization of *a* solution involving as many sensors as possible, which quite often is exactly the lowest tolerable operation mode. We leave the creation of a constraint programming solver tailored for environmental models open as future work.

Instead we turn our attention to a more complex and realistic COP, and consider the soil monitoring sensor network de-

[8]Generic Constraint Development Environment. http://www.gecode.com/

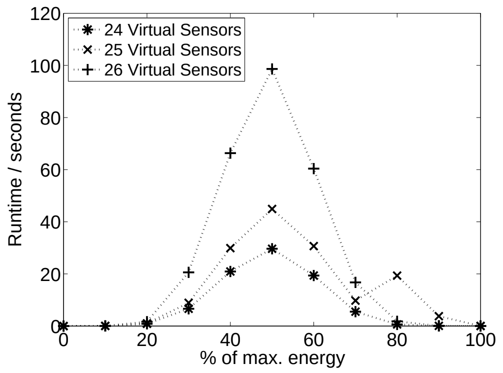


Figure 2. The effect constraining resources has on the search space.

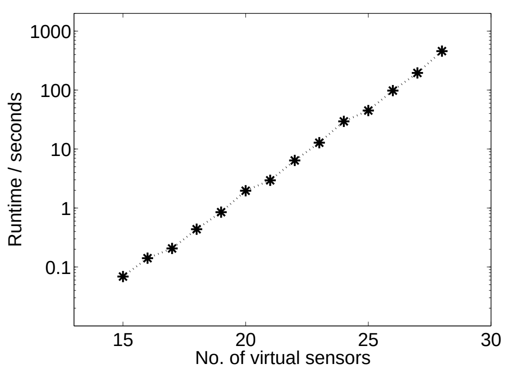


Figure 3. The influence of virtual sensors on the search space. Note the logarithmic scale.

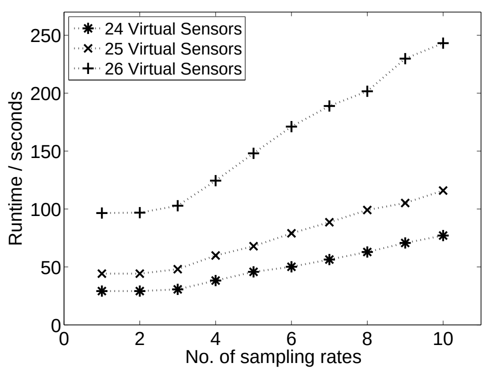


Figure 4. The influence of the number of sampling rates on the search space.

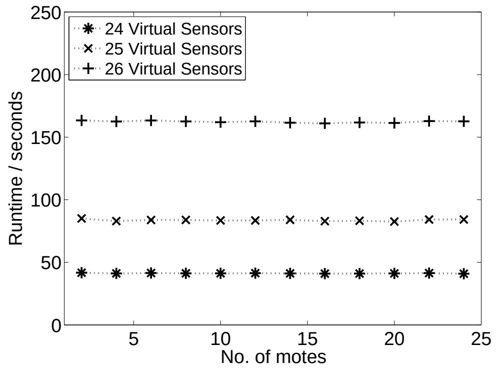


Figure 5. The influence of motes on the search space.

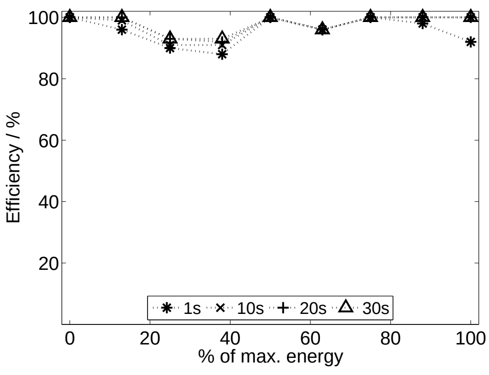


Figure 6. Restricting runtime for the ”Life Under Your Feet” COP with 30 motes and 120 virtual sensors.

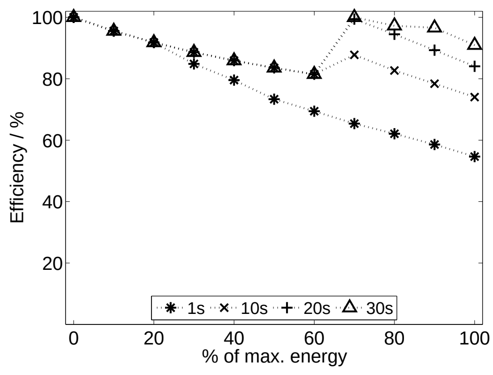


Figure 7. Restricting runtime for the ”Life Under Your Feet” COP with 100 motes and 400 virtual sensors.

scribed in Section 3.1. Besides the increased virtual sensors needed, this COP also considers each mote's communication cost caused by multi-hop routing. For this evaluation, we choose a simple 4-hop binary tree topology, with the excess motes evenly spread among the leaves.

First we consider a system of 120 virtual sensors, spread evenly among 30 motes, each containing two datasets and two possible sampling rates. We plot the relation between energy constraint and optimal solution for four cut-off runtimes in Figure 6. The energy is varied between the lowest to highest state and the efficiency is measured as the percentage of the optimal solution. Surprisingly, the efficiency is above 88 % for even the shortest runtime of 1 s while almost half of all the solutions found are the optimal one.

We then increase the state space by considering 400 virtual sensors attached to 100 motes. The results can be seen in Figure 7. Overall the increased state space decreases the efficiency for all cut-off times and not surprisingly the 1 s cut-off suffers the most. Looking closer, although the state space has increased by a factor of 2^{70} the 20-30 s cut-off times are still able to achieve 80-100 % efficiency.

This result shows that for environmental monitoring where changes happen on the order of minutes, our controller is efficient enough to instrument the sensor network in a timely manner. Especially, since *any* solution that satisfies the COP also satisfies the needs of the ecologist, regardless of the achieved utility.

5.3 Discussion

In the previous sections we explored the scalability of our model by measuring the runtime over a broad range of parameters. We discovered that even with a generic solver and an exponential search space, the first solutions found are quite often close to the optimal solutions. This last result is encouraging since it suggests that we can plan for even large networks by enforcing an upper bound on the search and still be confident that the result will be close to optimal.

At the same time, it enables several contingency strategies. For instance, if no solution can be found within the given time limit, another search can be initiated but with a higher time limit, or relaxation techniques can be used to simplify the problem by removing datasets from the model one at the time, stopping with the baseline dataset. These simpler problems will thus have a higher chance of success.

6 Case Study: Lake Monitoring

In order to illustrate the usefulness of ADAE we present a case study of a buoy equipped with a mobile water monitor [23]. The virtual sensor abstraction completely shields from the controller that the physical sensors are in fact mobile and not stationary. Unlike typical wireless sensor networks built from low power motes with inexpensive sensors attached, this water monitoring system consists of a $20,000 high-quality data logger which has been network enabled. Being able to control legacy systems is interesting because these are instruments the ecologists know they can trust.

This system consists of a single buoy equipped with a water monitor capable of measuring conductivity, temperature, dissolved oxygen, pH, and fluorescence. These properties are important in estimating the primary production and respiration in the lake ecosystem.

The water monitor is attached to the buoy with a 10 m long cable. An electric motor is used to adjust the vertical location of the water monitor thus enabling measurements at different depths. The buoy is powered by solar panel and is equipped with battery for night time operation. A Real-Time Control Unit (RTCU) instruments the motor and water monitor. Collected data is directly transmitted from the buoy to a back-end database over the Internet through a GSM modem. This connection is also used to transmit new configurations to the buoy, which is used by the RTCU to control the depth and sampling rate of the water monitor. In case of network outage, the RTCU reverts to being a data logger and stores all measurements locally until network service has been restored, at which point the data is offloaded.

This system has been deployed continuously for an entire season. Twice an hour the water monitor is lowered to ten predefined depths and at each depth all five sensors are sampled.

6.1 Problem Statement

Because of surface heating in the summer and the lake's dynamics, two distinct temperature regions, characterized by a sharp boundary, can be formed at the top and bottom. This stratification is interesting for the ecologist because the biological activity is particularly high at this boundary.

Understanding this layering with measurements clustered around the boundary would be of significant scientific value. However, since the formation of this stratification and its location is neither predictable nor static over time it is not possible to specify the exact measurement depths *a priori*. Hence, the previous season's measurements have all been done at ten fixed depth, spread out evenly down to 9 m.

As it turns out, the buoy's power supply has been over dimensioned with a solar panel and battery capacity exceeding the maximum power consumption of the system, even when running continuously. On the other hand, because the water monitor has to physically move, the system can at most measure 15 samples every half-an-hour, due to the actual movement and the following stabilization of the water.

Thus, in this case study the purpose of our adaptive data acquisition is not to conserve energy or meet lifetime requirements, but rather to increase the quality of collected data by adapting the sampling strategy.

In other words, the problems we seek to solve is (1) to detect and track the location of this temperature boundary and (2) instruct the buoy to collect extra samples in this region, besides the ten fixed samples.

Using the formalism presented in Section 3.1 we state the ecologist's requirements as follows:

- We define three **data collection modes**: *baseline*, *stratification*, and *correction*. We did not define failure modes in this first deployment. We will leverage some of the lessons we learnt to define failure modes in the next deployment (based on anomalous patterns indicating a problem with a probe as suggested in the Introduction).
- The *baseline* is always present, while the **condition** for the *stratification* event is the formation of a temperature gradient greater than $0.5°C/m$. Finally, the **condition** for the *correction* is when the *sparseness threshold* is reached.
- The **target dataset** for the *baseline* is ten samples at fixed depth. For the *stratification*, it is five samples clustered around the highest temperature gradient. The *correction* is to re-collect missing samples.

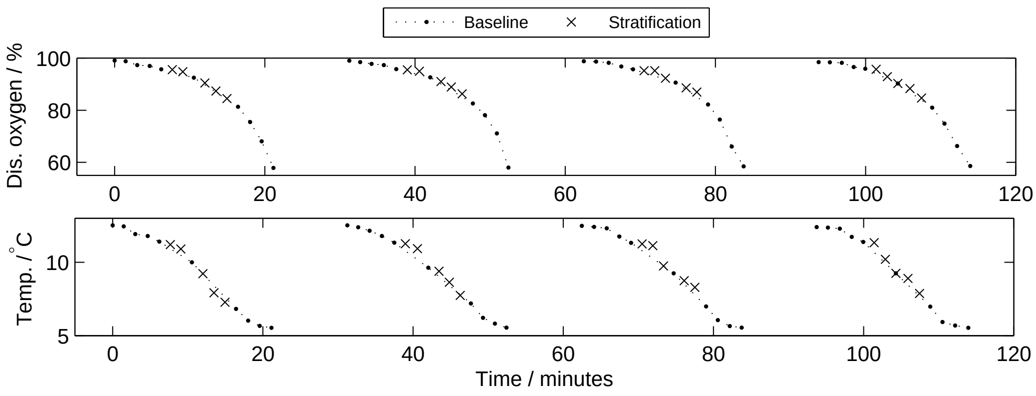


Figure 8. Four vertical profiles of temperature (buttom) and dissolved oxygen (top) measurements. Measurements obtained with the *baseline* dataset has been overlaid with measurements from the *stratification* dataset.

- Only the *baseline* is given a **sparseness threshold** of ten samples every half-an-hour.

6.2 COP Modeling

Given the requirements and the physical constraints of the system, we create three datasets. The *baseline* contains five virtual sensors (one for each modality) fixed at each of the ten depths, while the *stratification* is constructed from two parameters: the epicenter and the range of the stratification. Computing the stratification **condition** is straightforward, and the highest value is used as the center and all values exceeding the $0.5°C/m$ threshold constitute the range. Last, the *correction* is constructed from the missing samples.
The reason behind the 30 minutes sparseness threshold is that physical changes in the lake happens on this timescale. Thus, combining a baseline dataset with its associated stratification dataset in one sequence is not an option because the *stratification* set will by definition be shifted with the transmit rate. In other words, the two datasets are mutually exclusive and must be applied sequentially. We constraint the transmit rate and set the *epoch* to 20 min. for the *baseline* and 10 min. for the *stratification*.
The ordering of the **collection modes** are then: *baseline*, *correction*, and *stratification*.
Because energy is not an issue in this case we do not define a constraint for this resource. However, there is still the temporal constraint on the system, where the water monitor's round trip time combined with the sampling time must not exceed the time frame dictated by the sampling rate. Because the water monitor and control unit operates in parallel we do not consider the transmission time since this is significantly smaller than the measurement time.
The water monitor requires 90 seconds to settle at each new location and sample all the sensors. We assign this time cost to each virtual sensor but add in the model that virtual sensors at the same depth can be sampled in parallel. In the application specific part of the model we define the systems temporal cost function as the number of virtual sensors times the cost of each individual sensor combined with the round-trip-time of the water monitor. We model the round-trip-time as twice the distance of the deepest placed virtual sensor divided by the speed of the electric motor.

6.3 Results

We implemented the model above in ADAE and used it to control the deployed buoy remotely through the back-end server. We used a low-power ARM based single board computer to serve as our controller.
In Figure 8 we show four series of temperature (bottom) and dissolved oxygen (top) measurements. The *baseline* dataset has been overlaid with the *stratification* dataset in order to illustrate the benefit of adaptive sampling. In each series, the measurements to the left are closer to the surface than the measurements to the right.
For the temperature, we see a sharp boundary in the 7-11$°C$ region where only one baseline measurement is present in each series and the rest of the measurements are clustered either above or below this region. When we look at the stratification measurements we see that there indeed are five in each series and that they are filling the gap left by the baseline measurements. For the dissolved oxygen we see a similar trend, with the extra measurements filling out the gaps left by the baseline measurements. Interestingly, the *stratification* measurements do not lie on a straight line between the *baseline* measurements, meaning knowledge has been gained by adding the extra measurements.

7 Conclusion

Sensor networks promise to radically improve the data acquisition systems that ecologists can deploy for in-situ instrumentation. There is however a risk of mismatch between the assumption by ecologists that the sensor network delivers exactly the time series that has been specified, and on the assumption by computer scientists that the goal is to collect as much data as possible (using yield as a performance metric). We argue that it is necessary to take failures and interesting events into account when specifying the ecologist requirements. We proposed data collection modes as a means to represent an envelope of target datasets (e.g., higher sampling rate, or higher accuracy, or different combination of modalities for a given period).

Based on the insight that uniform and systematic ecosystem sampling is neither possible, nor desirable, we proposed ADAE, a utility-based controller, that adaptively configure motes in a changing environment. ADAE is based on the assumption that motes export a simple configuration API in order to easily interface with legacy systems. We described a three-tier architecture to organize the complexity of communicating with motes, representing the sensor network state and the user requirements, generating constraint optimization problems (COP) to determine the configuration parameters, and solving these COP. We showed that a COP solver scales to realistic multihop sensor networks, and we illustrated the benefits of ADAE in a lake monitoring monitoring system deployed this summer.
We are in the process of preparing new deployments. These are needed to explore the limitations of data collection modes, as well as ADAE. In particular, we are investigating how to handle a very dynamic environment, and how to handle failures in a long-term autonomous deployment.

8 References

[1] M. Ai-Chang, J. Bresina, and L. e. a. Charest. MAPGEN: mixed-initiative planning and scheduling for the Mars Exploration Rover mission. *Intelligent Systems, IEEE*, 2004.

[2] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand. An Architecture for Autonomy. *International Journal of Robotics Research*, 1998.

[3] P. Bonnet, P.Seshadri, and J. Gehrke. Towards Sensor Database Systems. In *Mobile Data Management*, 2001.

[4] P. Borgstrom, A. Singh, B. Jordan, G. Sukhatme, M. Batalin, and W. Kaiser. Energy based path planning for a novel cabled robotic system. In *IEEE IROS*, 2008.

[5] Q. Cao, D. Fesehaye, N. Pham, Y. Sarwar, and T. Abdelzaher. Virtual Battery: An Energy Reserve Abstraction for Embedded Sensor Networks. In *IEEE RTSS*, 2008.

[6] M. Chang, A. Terzis, and P. Bonnet. Mote-Based Online Anomaly Detection Using Echo State Networks. In *DCOSS*, 2009.

[7] H.-L. Choi and J. P. How. A Multi-UAV Targeting Algorithm for Ensemble Forecast Improvement. *AIAA Guidance, Navigation and Control Conference and Exhibit*, 2007.

[8] A. Deshpande and S. Madden. MauveDB: supporting model-based user views in database systems. In *ACM SIGMOD*, 2006.

[9] R. Doyle, D. Bernard, E. Riedel, N. Rouquette, J. Wyatt, M. Lowry, and P. Nayak. Autonomy and Software Technology on NASA's Deep Space One. *IEEE Intelligent Systems*, 1999.

[10] D. Estrin, L. Girod, G. Pottie, and M. Srivastava. Instrumenting the World with Wireless Sensor Networks. In *International Conference on Acoustics, Speech, and Signal Processing*.

[11] J. Frank and A. Jónsson. Constraint-Based Attribute and Interval Planning. *Constraints*, 2003.

[12] M. Freed and R. Remington. Managing decision resources in plan execution. In *IJCAI'97: Proceedings of the 15th international joint conference on Artifical intelligence*, 1997.

[13] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection Tree Protocol. In *ACM SenSys*, 2009.

[14] O. Gnawali, B. Greenstein, K.-Y. Jang, A. Joki, J. Paek, M. Vieira, D. Estrin, R. Govindan, and E. Kohler. The TENET Architecture for Tiered Sensor Networks. In *ACM SenSys 2006*.

[15] J. W. Hui and D. Culler. The Dynamic Behavior of a Data Dissemination Protocol for Network Programming at Scale. In *ACM SenSys*, 2004.

[16] G. H. K. Lam, H. Va Leong, and S. C. F. Chan. BBQ: group-based querying in a ubiquitous environment. In *ACM SAC '06*, 2006.

[17] M. Leopold, M. Chang, and P. Bonnet. Characterizing Mote Performance: A Vector-Based Methodology. In *EWSN*, 2008.

[18] M. Li, D. Ganesan, and P. Shenoy. PRESTO: feedback-driven data management in sensor networks. *IEEE/ACM Trans. Netw.*, 2009.

[19] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TinyDB: an acquisitional query processing system for sensor networks. 2005.

[20] S. M.Franklin, J.Hellerstein. Thinking Big About Tiny Databases. *Data Engineering Bulletin*, 2007.

[21] R. Musăloiu-E., A. Terzis, K. Szlavecz, A. Szalay, J. Cogan, and J. Gray. Life Under your Feet: A WSN for Soil Ecology. In *EmNets Workshop*, May 2006.

[22] N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, and G. Tack. MiniZinc: Towards a Standard CP Modelling Language. In *13th International Conference on Principles and Practice of Constraint Programming*, 2007.

[23] Peter A. Staehr and Rikke M. Closter. Measurement of whole system metabolism using automatic profiling sensors. In *GLEON 6*, 2008.

[24] J. Polastre, J. Hill, and D. Culler. Versatile Low Power Media Access for Wireless Sensor Networks. In *ACM SenSys*, 2004.

[25] S. Russell and P. Norvig. *Artificial Intelligence A Modern Approach, 2nd Ed.* Prentice Hall, 2003.

[26] G. S. S. Thompson. *Adaptive Sampling*. Wiley Interscience, 1996.

[27] G. Werner-Allen, S. Dawson-Haggerty, and M. Welsh. Lance: optimizing high-resolution signal collection in wireless sensor networks. In *ACM SenSys*, 2008.

Chapter 6

Conclusion

Although sensornets used for habitat and wildlife monitoring have been around for a decade, domain scientists have yet to embrace sensornets as a scientific instrument. We postulated in the introduction that the reason for this discrepancy lies in the complexity of successfully deploying a sensornet, a fact backed up by the large presence of computer scientists in every sensornet deployment. We argued that in order for sensornets to be used by domain scientists, the same way they would any other instrument in their toolbox, this gap had to be bridged and the complexity reduced. In summary, we identified three key issues with using sensornets in a scientific capacity:

- Deployment. Successfully designing and deploying a sensornet is not trivial. A set of tools is needed in order for domain scientists to be able to design and deploy their own sensornets.
- Quality. For the collected data to be usable as experimental evidence the quality must be high enough, meaning faulty and missing measurements must be kept at a minimum.
- Flexibility. Sensornets deployed for scientific monitoring have lacked the flexibility a scientist in the field would have.

These are the problems we addressed in this dissertation, which lead us to our thesis that **sensornets should become instruments that domain scientists can pick, deploy, and program to efficiently collect high quality datasets.**

We used automated data acquisition in the Hogthrob project, where we monitored the activity of household sows using accelerometry. Although successful in detecting oestrus in sows, we encountered the same problems presented above. First, during the design phase it was not obvious which mote would suit our deployment best. We needed a way to evaluate the performance of different platforms without actually deploying them. Second, after the experiment, we realized that many of the measurements where either faulty or missing, making it impossible to describe the sows activity during those periods, based on the acceleration data alone. Being able to detect the faulty measurements immediately would have enabled us to resample the sensors and perhaps recover some usable measurements. Third, with a fixed sampling rate all activities were measured with the same resolution. Since storage became a problem due to network outage, it would have increased the quality of the data if the sampling rate had been adapted to the actual activity of the sow, e.g., reduce the sampling rate when the sow was sleeping.

Following the Scientific Method, we returned to revise our problem statement and hypothesis, which lead us to our three front approach: First, to help choose the right set of motes and applications for a deployment we developed a benchmarking methodology. Second, to increase the quality of the collected data we developed an anomaly detection framework. And third, to make it easier for the domain scientist to control the sensornet, and for the sensornet to be able to react to changes in the collected data, we developed a controller to mediate between the scientist and the sensornet.

With our benchmarking methodology, we presented a systematic way to predict the energy and time consumption of one application on different platforms, and different applications on the same platform. We accomplished this by devising a method to divide the resource consumption into two independent contributions, one specific to the application and one specific to the platform, using linear algebra. This cross-platform prediction is essential in preparation for a deployment, since it enables the possibility to consider a wide range of motes without actually having to deploy them.

In the anomaly detection framework presented, we use machine learning algorithms to differentiate between normal and anomalous data, by defining the latter to be all measurements that deviate significantly from a training set. This unified fault and event detection framework, based on Neural Networks and Bayesian Networks, proved to be superior to the heuristics deployed by previous sensornets, both in terms of detection accuracy and sensitivity. A watchdog based on this anomaly detection framework, could both be used to detect faulty measurements, with the purpose of resampling, and to adapt the sampling rate based on the detected anomaly.

Finally, we presented an adaptive data acquisition controller, based on the AI *Planning and Scheduling* architecture, to act as a proxy for the domain scientist. This controller adaptively changes the sampling strategy by retasking the sensornet, based on the state of the environment and the scientists' requirements. These requirements are based on temporal and spatial sparseness of the expected measurements instead of the more traditional yield. We showed that the controller had successfully directed the sampling in a lake monitoring deployment, both detecting changes in the environment and correctly adapting the sampling strategy accordingly. More importantly, the measurements captured with the adaptive sampling proved to contain features that would not have been discovered otherwise.

In summary, with the three major contributions in this dissertation:

- A vector-based methodology to characterize mote performance, enabling cross-platform prediction of an application's energy and time consumption without actual execution.
- An anomaly detection framework, based on machine learning algorithms and lightweight enough to run online on a mote in conjunction with a data acquisition application.
- A sensornet controller based on artificial intelligence, that given the state of the environment and available resources chooses a set of actions that match the scientist's requirements most.

we conclude our thesis that **sensornets should become instruments that domain scientists can pick, deploy, and program to efficiently collect high quality datasets.**

Future Work

Normalizing Resource Consumption

In our benchmarking framework, CPU performance is measured using a resource intensive application. This simplified model is adequate as a first order approximation, however, for more precise predictions it is necessary to explore better alternatives to characterize CPU performance.

One direction could be to use either a set of applications and then average the performance, or use a synthesized application specifically tailored to mimic a broad range of applications. The challenge lies in the different CPU architectures used on different motes, where one application might suit one architecture more than another. Of course, this is closely related to the choice of compiler, which would also effect the CPU performance.

Due to the amount of motes we had available at the time, the number of platforms we benchmarked is very limited. Exploring more platforms, different types of applications, and operating systems is essential in understanding under which conditions our benchmarking framework is applicable. To facilitate the collection of traces a more recent tracing method such as *Quanto* by Fonseca et al. [2] should be used. With this increased knowledge, we will also be able to verify that our linearity hypothesis is indeed valid, by applying the multilinearity tests [1].

Finally, our method is based on collecting application traces in situ, thereby incorporating non-deterministic contributions, and subsequently divide the resource consumptions into platform and application specific vectors. However, by doing this we actually add the non-deterministic contribution to the application vector. In theory, it should be possible to deconstruct the application vector into a *deployment vector*, characterizing the non-deterministic contribution collected in the trace, and a pure application vector, which only depends on the application's source code. This would give us a three-way prediction methodology, where we would be able to predict the combined effect of multiple platforms and applications in different environments.

Anomaly Detection

With a framework based on machine learning, the most important factor in determining the accuracy of the anomaly detection lies in the quality of the training sets.

Understanding the difference between artificial training sets constructed from models and sets originating from actual measurements, would lead to more efficient anomaly detection. One could imagine, that depending on the anomaly one wish to detect and the quality of the measurements available, a tradeoff exists between choosing one over the other.

In the presence of adaptive sampling, it is also necessary to explore how to adapt the anomaly detection to match the active sampling rate. Depending on the algorithm in question, different parameters could be stored for different sampling rates. Alternatively, the training set could also be based on the highest sampling rate available and when a lower sampling rate is used, one could interpolate the missing data points before applying the detection algorithm.

Another interesting area for future work is to explore the two-dimensional space consisting of machine learning algorithms and sensor modalities. We have shown that Neural Networks and Bayesian Networks are capable of tracking moisture and temperature data, however, this might not be the case for other modalities. Understanding the effectiveness of different algorithms is key in choosing the right algorithm for the right modality.

Finally, building a watchdog based on this anomaly detection framework would be interesting. Especially in combination with an adaptive sensornet controller such as ADAE. Obviously, using a watchdog locally on the mote to react to changes or have a controller to globally retask the sensornet are both beneficial. However, exploring the effect of combining both in the same sensornet is intriguing, especially the boundary between delegating responsibility to the watchdog or to the controller.

Adaptive Data Acquisition

Although we did have an actual deployment with our controller, the sensornet instrumented was severely restricted in terms of nodes and feasible data collection strategies. With time being a limited resource, due to the actual movement of the sensor, and the over-sized power supply removing the energy constraint, the configuration space was atypical for a sensornet. In order to fully explore the benefits and limitations of our sensornet controller, more deployments with larger networks and configuration space are necessary.

In the MANA[1] project, we deploy sensornets in remote arctic regions for environmental monitoring. The conditions in these regions are significantly different from those typically encountered in sensornet deployments, with the two most important factors being the harsh weather conditions and complete lack of infrastructure. The low temperatures and strong wind conditions make equipment failures unavoidable, and the remote location means access is limited to at most a handful of visits each year, putting an extreme toll on the logistics. In these conditions, sensornets are without supervision for months, power is completely dependent on the varying sun and wind conditions, and component failure is unavoidable.

We plan to deploy ADAE to control the sensornets in this project, thereby handling failures more efficiently, and utilizing the fluctuating power supply

[1]`http://www.itu.dk/mana/`

more consistently, with the help of adaptive sampling. Another interesting way to increase the sensornet's robustness, especially in harsh environments where failures are frequent, is to incorporate *Control Theory* on each mote to ensure that the tasks given by ADAE are carried out to the best of the mote's ability.

Traditionally, after data has been collected in the field, measurements have been processed using models and heuristics as part of the analysis process. New and improved models are subsequently build from these derived data products. In ADAE, we have in effect moved part of this modeling to the data acquisition process, in order to make it adaptive. Obviously, applying too much of the model during the data acquisition process will both effect performance and bias the measurements, making it difficult to construct new models. Finding the optimal tradeoff between performance and adaptability is an interesting issue for future research.

Bibliography

[1] A. Bogdanov. The Multilinearity Test. `http://itcs.tsinghua.edu.cn/courses/complexity/notes/lec14.pdf`.

[2] R. Fonseca, P. Dutta, P. Levis, and I. Stoica. Quanto: Tracking energy in networked embedded systems. In R. Draves and R. van Renesse, editors, *OSDI*, pages 323–338. USENIX Association, 2008.

Bibliography

[1] G. Barrenetxea, F. Ingelrest, G. Schaefer, and M. Vetterli. The Hitchhiker's Guide to Successful Wireless Sensor Network Deployments. In *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 43–56, New York, NY, USA, 2008. ACM.

[2] A. Bogdanov. The Multilinearity Test. `http://itcs.tsinghua.edu.cn/courses/complexity/notes/lec14.pdf`.

[3] P. Borgstrom, A. Singh, B. Jordan, G. Sukhatme, M. Batalin, and W. Kaiser. Energy Based Path Planning for a Novel Cabled Robotic System. In *IEEE IROS*, 2008.

[4] R. Burns, A. Terzis, and M. Franklin. Software tools for sensor-based science. In *Proceedings of the Third Workshop on Embedded Networked Sensors (EmNets 2006)*, Feb. 2006.

[5] K.-W. Fan, Z. Zheng, and P. Sinha. Steady and Fair Rate Allocation for Rechargeable Sensors in Perpetual Sensor Networks. In *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 239–252, New York, NY, USA, 2008. ACM.

[6] R. Fonseca, P. Dutta, P. Levis, and I. Stoica. Quanto: Tracking energy in networked embedded systems. In R. Draves and R. van Renesse, editors, *OSDI*, pages 323–338. USENIX Association, 2008.

[7] O. Gnawali, B. Greenstein, K.-Y. Jang, A. Joki, J. Paek, M. Vieira, D. Estrin, R. Govindan, and E. Kohler. The TENET Architecture for Tiered Sensor Networks. In *ACM SenSys 2006*, Nov. 2006.

[8] B. Greenstein, C. Mar, A. Pesterev, S. Farshchi, E. Kohler, J. Judy, and D. Estrin. Capturing high-frequency phenomena using a bandwidth-limited sensor network. In *SenSys '06: Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 279–292, New York, NY, USA, 2006. ACM.

[9] J. Gupchup, A. Sharma, A. Terzis, R. Burns, and A. Szalay. The Perils of Detecting Measurement Faults in Environmental Monitoring Networks. In *DCOSS*, 2008.

[10] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury. *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.

[11] P. Huang, H. Oki, Y. Wang, M. Martonosi, L. Peh, and D. Rubenstein. Energy-efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet. In *Proceedings of the Tenth Internationla Conferece on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X)*, Oct. 2002.

[12] A. Lachenmann, P. J. Marrón, D. Minder, and K. Rothermel. Meeting lifetime goals with energy levels. In *SenSys '07: Proceedings of the 5th international conference on Embedded networked sensor systems*, pages 131–144, New York, NY, USA, 2007. ACM.

[13] O. Landsiedel, K. Wehrle, and S. Gotz. Accurate Prediction of Power Consumption in Sensor Networks. In *EmNets '05: Proceedings of the 2nd IEEE workshop on Embedded Networked Sensors*, pages 37–44, Washington, DC, USA, 2005. IEEE Computer Society.

[14] K. Lorincz, B.-r. Chen, J. Waterman, G. Werner-Allen, and M. Welsh. Resource Aware Programming in the Pixie OS. In *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 211–224, New York, NY, USA, 2008. ACM.

[15] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 88–97, New York, NY, USA, 2002. ACM.

[16] I. McCauley, B. Matthews, L. Nugent, A. Mather, and J. Simons. Wired Pigs: Ad-Hoc Wireless Sensor Networks in Studies of Animal Welfare. In *Proceedins of the Second IEEE Workshop on Embedded Networked Sensors (EmNetS-II)*, May 2005.

[17] R. Musăloiu-E., A. Terzis, K. Szlavecz, A. Szalay, J. Cogan, and J. Gray. Life Under your Feet: A Wireless Soil Ecology Sensor Network. In *EmNets Workshop*, May 2006.

[18] S. Nath, J. Liu, and F. Zhao. Sensormap for wide-area sensor webs. *Computer*, 40(7):90–93, 2007.

[19] K. Ni, N. Ramanathan, M. N. H. Chehade, L. Balzano, S. Nair, S. Zahedi, E. Kohler, G. Pottie, M. Hansen, and M. Srivastava. Sensor Network Data Fault Types. *ACM Trans. Sen. Netw.*, 5(3):1–29, 2009.

[20] N. Ramanathan, T. Schoellhammer, E. Kohler, K. Whitehouse, T. Harmon, and D. Estrin. Suelo: Human-assisted Sensing for Exploratory Soil

Monitoring Studies. In *SenSys '09: Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, pages 197–210, New York, NY, USA, 2009. ACM.

[21] S. Russell and P. Norvig. *Artificial Intelligence A Modern Approach, 2nd Ed.* Prentice Hall, 2003.

[22] L. Selavo, A. Wood, Q. Cao, T. Sookoor, H. Liu, A. Srinivasan, Y. Wu, W. Kang, J. Stankovic, D. Young, and J. Porter. LUSTER: Wireless Sensor Network for Environmental Research. In *SenSys '07: Proceedings of the 5th international conference on Embedded networked sensor systems*, pages 103–116, New York, NY, USA, 2007. ACM.

[23] M. Seltzer, D. Krinsky, K. Smith, and X. Zhang. The Case for Application-Specific Benchmarking. In *Hot Topics in Operating Systems, 1999. Proceedings of the Seventh Workshop on*, pages 102–107, 1999.

[24] A. Sharma, L. Golubchik, and R. Govindan. On the Prevalence of Sensor Faults in Real-World Deployments. In *IEEE SECON*, pages 213–222, 2007.

[25] V. Shnayder, M. Hempstead, B. rong Chen, G. Werner-Allen, and M. Welsh. Simulating the Power Consumption of Large-Scale Sensor Network Applications. In *Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys'04)*, Nov. 2004.

[26] K. Szlavecz, A. Terzis, R. Musaloiu-E., C.-J. Liang, J. Cogan, A. Szalay, J. Gupchup, J. Klofas, L. Xia, C. Swarth, and S. Matthews. Turtle Nest Monitoring with Wireless Sensor Networks. In *Proceedings of the American Geophysical Union*, 2007.

[27] G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, and W. Hong. A Macroscope in the Redwoods. In *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 51–63, New York, NY, USA, 2005. ACM.

[28] G. Werner-Allen, S. Dawson-Haggerty, and M. Welsh. Lance: optimizing high-resolution signal collection in wireless sensor networks. In *ACM SenSys*, pages 169–182, New York, NY, USA, 2008. ACM.

[29] G. Werner-Allen, K. Lorincz, M. Welsh, O. Marcillo, J. Johnson, M. Ruiz, and J. Lees. Deploying a Wireless Sensor Network on an Active Volcano. *IEEE Internet Computing*, 10(2):18–25, 2006.