Experiences from Two Sensor Network Deployments — Self-Monitoring and Self-Configuration Keys to Success

Niclas Finne, Joakim Eriksson, Adam Dunkels, Thiemo Voigt

Swedish Institute of Computer Science, Box 1263, SE-164 29 Kista, Sweden
{nfi,joakime,adam,thiemo}@sics.se

Abstract. Despite sensor network protocols being self-configuring, sensor network deployments continue to fail. We report our experience from two recently deployed IP-based multi-hop sensor networks: one in-door surveillance network in a factory complex and a combined out-door and in-door surveillance network. Our experiences highlight that adaptive protocols alone are not sufficient, but that an approach to self-monitoring and self-configuration that covers more aspects than protocol adaptation is needed. Based on our experiences, we design and implement an architecture for self-monitoring of sensor nodes. We show that the self-monitoring architecture detects and prevents the problems with false alarms encountered in our deployments. The architecture also detects software bugs by monitoring actual and expected duty-cycle of key components of the sensor node. We show that the energy-monitoring architecture detects bugs that cause the radio chip to be active longer than expected.

1 Introduction

Surveillance is one of the most prominent application domains for wireless sensor networks. Wireless sensor networks enable rapidly deployed surveillance applications in urban terrain. While most wireless sensor network mechanisms are self-configuring and designed to operate in changing conditions [12, 16], the characteristics of the deployment environment often cause additional and unexpected problems [8, 9, 11]. In particular, Langendoen et al. [8] point out the difficulties posed by, e.g., hardware not working as expected.

To contribute to the understanding of the problems encountered in real-world sensor network deployments, we report on our experience from recent deployments of two surveillance applications: one in-door surveillance application in a factory complex, and one combined out-door and in-door surveillance network. Both applications covered a large area and therefore required multi-hop networking.

Our experiences highlight that adaptive protocols alone are not sufficient, but that an approach to self-monitoring and self-configuration that covers more aspects than protocol adaptation is needed. An example where we have experienced the need for self-monitoring of sensor nodes is when the components used in low-cost sensor nodes behave differently on different nodes. In many of our experiments, radio transmissions triggered the motion detector on a subset of our nodes while other nodes did not experience this problem.

Motivated by the observation that self-configuration and adaptation is not sufficient to circumvent unexpected hardware and software problems, we design and implement a self-monitoring architecture for detecting hardware and software problems. Our architecture consists of pairs of probes and activators where the activators start up an activity that is suspected to trigger problems and the probes measure if sensor components react to the activator's activity. Callback functions enable a node to self-configure its handling of a detected problem. We experimentally demonstrate that our approach solves the observed problem of packet transmissions triggering the motion detector.

To find software problems, we integrate Contiki's software-based on-line energy estimator [5] into the self-monitoring architecture. This allows us to detect problems such as the CPU not going into the correct low power mode, a problem previously encountered by Langendoen et al. [8]. With two examples we demonstrate the effectiveness of the self-monitoring architecture. Based on our deployment experiences, we believe this tool to be very valuable for both application developers and system developers.

The rest of the paper is structured as follows. The setup and measurements for the two deployments are described in Section 2. In Section 3 we present our experiences from the deployments, including unexpected behavior. Section 4 describes our architecture for self-monitoring while the following section evaluates it. Finally, we describe related work in Section 6 and our conclusions in Section 7.

2 Deployments

We have deployed two sensor network surveillance applications in two different environments. The first network was deployed indoors in a large factory complex setting with concrete floors and walls, and the second in a combined outdoor and indoor setting in an urban environment.

In both experiments, we used ESB sensor nodes [14] consisting of a MSP430 microprocessor with 2kB RAM, 60kB flash, a TR1001 868 MHz radio and several sensors. During the deployments, we used the ESB's motion detector (PIR) and vibration sensor.

We implemented the applications on top of the Contiki operating system [4] that features the uIP stack, the smallest RFC-compliant TCP/IP stack [3]. All communication uses UDP broadcast and header compression that reduces the UDP/IP header down to only six bytes: the full source IP address and UDP port, as well as a flag field that indicates whether or not the header is compressed.

We used three different types of messages: *Measurement messages* to send sensor data to the sink, *Path messages* to report forwarding paths to the sink, and *Alarm messages* that send alarms about detected activity.

We used two different protocols during the deployment. In the first experiment, we used a single-hop protocol where all nodes broadcast messages to the sink. In the second experiment, we used a multi-hop protocol where each node calculates the number of hops to the sink and transmits messages with a limit on hops to the sink. A node only forwards messages for nodes it has accepted to be relay node for. A message can take several paths to the sink and arrive multiple times. During the first deployment only a few nodes were configured to forward messages, but in the second deployment any node could configure itself to act as relay node.

After a sensor has triggered an alarm, an alarm message is sent towards the sink. Alarm messages are retransmitted up to three times unless the node hears an explicit acknowledgment message or overhears that another node forwards the message further. Only the latest alarm from each node is forwarded.

2.1 First Deployment: Factory Complex

The first deployment of the surveillance sensor network was performed in a factory complex. The main building was about 250 meters times 25 meters in size and three floors high. Both floors and most walls were made of concrete but there were sections with office-like rooms that were separated by wooden walls. Between the bottom floor and first floor there was a smaller half-height floor. The largest distance between the sink and the most distant nodes was slightly less than 100 meters.

The sensor network we deployed consisted of 25 ESB nodes running a surveillance application. All nodes were either forwarding messages to the sink or monitored their environment using the PIR sensor and the vibration detector. We made several experiments ranging from a single hop network for measuring communication quality to a multi-hop surveillance network.

Single-Hop Network Experiment We made the first experiment to understand the limitations of communication range and quality in the building. All nodes communicated directly with the sink and sent measurement packets at regular intervals.

Node	Distance (meter)	Walls	Received	Sent (expected)	Sent (actual)	Reception ratio (percent)	Signal strength (avg,max)	
2	65	1 C	92	621	639	15%	1829	2104
3	21	1 W	329	587	588	56%	1940	2314
4	55	1 C	72	501	517	14%	1774	1979
5	33	2 W	114	611	613	19%	1758	1969
6	18	1 W	212	580	590	37%	1866	2230
7	26	2 W	347	587	588	59%	2102	2568
8	15	1 W	419	584	585	71%	2131	2643
9	25	1 W	194	575	599	34%	1868	2218
10	23	2 W	219	597	599	37%	1815	2106
11	17	1 W	331	591	593	56%	2102	2582
50	27	2 W	230	587	594	39%	1945	2334

Table 1. Communication related measurements for the first experiment.

Table 1 shows the results of the measurements. The columns from left are node id, distance from the sink in meters, number of concrete and wooden walls between node and the sink, number of messages received at the sink from the node, number of messages sent by the node (calculated on sequence number), actual number of messages sent (read from a log stored in each node), percentage of successfully delivered messages, and signal strength measured at the sink. Table 1 shows that as expected the ratio of received messages decreases with increasing distance from the sink. As full sensor coverage of the factory was not possible using a single-hop network, we performed the other experiments with multi-hop networks.

Multi-Hop Network Experiments After performing some experiments to understand the performance of a multi-hop sensor network with respect to communication and surveillance coverage, we performed the final experiment in the first deployment during a MOUT (Military Operation on Urban Terrain) exercise with 15-20 soldiers moving up and down the stairs and running in the office complex at the top level of the building.

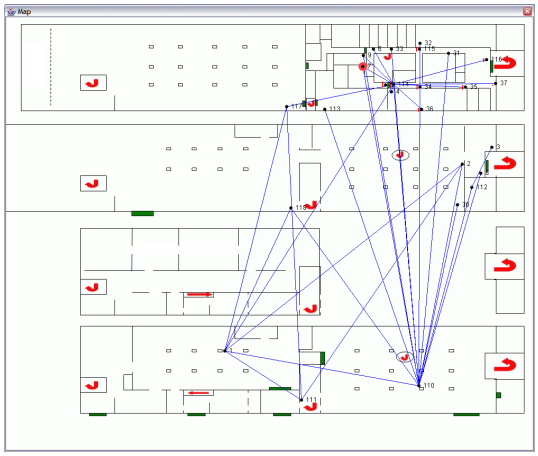

Fig. 1. Screenshot from the final experiment in the factory deployment illustrating placement of the sensor nodes during the surveillance and the paths used to transfer messages. All levels of the factory are shown with the top level at the top of the screenshot and the basement at the bottom. The office complex is on the right side at the top level in the figure.

Node 110 (see Figure 1) was the most heavily loaded forwarding node in the network. It had a direct connection to the sink and forwarded 10270 messages from other nodes during the three hour long experiment. During the experiment the sink received 604 alarms generated by node 110. 27 percent of these alarms were received several times due to retransmissions. Node 8 had seven paths to the sink, one direct connection with the sink, and six paths via different forwarding nodes. The sink received 1270 unique alarms from node 8 and 957 duplicates. Most of the other nodes' messages multi-hopped over a few alternative paths to the sink, with similar or smaller delays than those from node 110 and 8. This indicates that the network was reliable and that most of the alarms got to the server; in many cases via several paths. With the 25 sensor nodes we achieved coverage of the most important passages of the factory complex, namely doors, stairs, and corridors.

2.2 Second Deployment: Combined in-door and out-door urban terrain

The second deployment was made in an artificial town built for MOUT exercises. It consisted of a main street and a crossing with several wooden buildings on both sides

of the streets. At the end of the main street there were some concrete buildings. The distance between the sink and the nodes at the edge of the network was about 200 meters, making the network more than twice as long as in the first deployment.

The surveillance system was improved in two important ways. First, the network was more self-configuring in that there was no need for manually configuring which role each node should have (relay node or sensor node). Each node configured itself for relaying if the connectivity to sink was above a threshold. Second, alarm messages also included path information so that information of the current configuration of the network was constantly updated as messages arrived to the sink. Even with the added path information in the alarm messages, the response times for alarms in the network were similar to the response times in the first deployment despite that the distant nodes were three or four hops away from the sink rather than two or three. Using 25 nodes we achieved fairly good sensor coverage of the most important areas.

3 Deployment Experiences

When we deployed the sensor network application we did not know what to expect in terms of deployment speed, communication quality, applicability of sensors, etc. Both deployments were made in locations that were new to us. This section reports on the various experiences we made during the deployments.

Network Configuration During the first deployment the configuration needed to make a node act as a relay node was done manually. This made it very important to plan the network carefully and make measurements on connectivity at different locations in order to get an adequate number of forwarding nodes. This was one of the largest problems with the first deployment. During the second deployment the network's self-configuration capabilities made deployment a faster and easier task.

The importance of self-configuration of the network routing turned out to be higher than we expected since we suddenly needed to move together with the sink to a safer location during the second deployment, where we did not risk being fired at. This happened while the network was deployed and active.

Unforeseen Hardware Problems During radio transmissions a few of the sensor nodes triggered sensor readings which cause unwanted false alarms. Since we detected and understood this during the first deployment, we rewrote the application to turn off sensing on the nodes that had this behavior. Our long term solution is described in the next section.

Parameter Configuration In the implementation of the communication protocols and surveillance application there are a number of parameters with static values set during early testing with small networks. Many of these parameters need to be optimized for better application performance. Due to differences in the environment, this optimization can only partly be done before deployment. Examples of such parameters are retransmission timers, alarm triggering delays, radio transmission power level, and time before refreshing a communication link.

Ground Truth It is important for understanding the performance of a sensor network deployment to compare the sensed data to ground truth. In our deployments, we did not have an explicit installation of a parallel monitoring system to obtain ground truth, but in both deployments we received a limited amount of parallel feedback.

During the first deployment, the sensor networks alerted us of movements in various parts of the factory but since we did not have any information about the soldiers' current locations it was difficult to estimate the time between detection by the sensor nodes and the alarm at the sink. Sometimes the soldiers threw grenades powerful enough to trigger the vibration sensors on the nodes. This way, we could estimate the time between the grenade explosions and the arrival of the vibration alarm at the sink. During the second deployment we received a real time feed from a wireless camera and used it to compare the soldiers' path with the alarms from the sensor network.

Radio Transmission During the first deployment we placed forwarding nodes in places where we expected good radio signal strength (less walls, and floors). We expected the most used path to the sink via forwarding nodes in the stairwells. However, most messages took a path straight through two concrete floors via a node placed at the ground floor below the office rooms where the sensors were deployed.

Instant Feedback One important feature of the application during deployment was that when started, a node visualized its connection. When it connected, the node beeped and flashed all its leds before being silent. Without this feature we would have been calling the person at the sink all the time just to see if the node had connected to the network. This way, we could also estimate a node's link quality. The longer time for the node to connect to the network, the worse it was connected. We usually moved nodes that required more than 5 - 10 seconds to connect to a position with better connectivity.

4 A Self-Monitoring Architecture for Detecting Hardware and Software Problems

To ensure automatic detection of the nodes that have hardware problems, we design a self-monitoring architecture that probes potential hardware problems. Our experiences show that the nodes can be categorized into two types: those with a hardware problem and those without. The self-probing mechanism could therefore possibly be run at start-up, during deployment, or even prior to deployment.

4.1 Hardware Self-Test

Detection of hardware problems is done using a self-test at node start-up. The goal of the self-test is to make it possible to detect if a node has any hardware problems.

The self-test architecture consists of pairs of probes and activators. The activators start up an activity that is suspected to trigger problems and the probes measure if sensors or hardware components react to the activator's activity. An example of a probe/activator pair is measuring PIR interrupts when sending radio traffic. The API

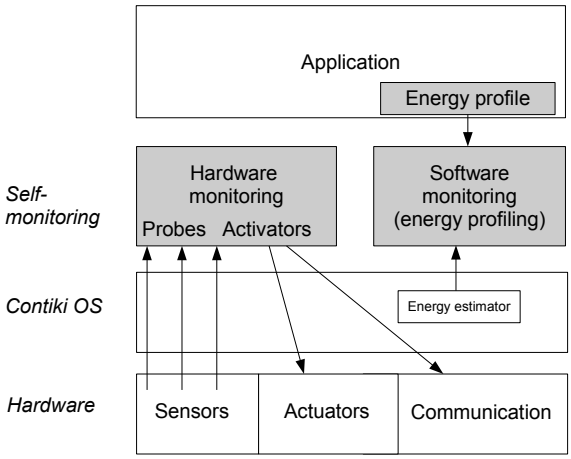

Fig. 2. Architecture for performing self-monitoring of both hardware and software. Self-tests of hardware are performed at startup or while running the sensor network application. Monitoring of the running software is done continuously using the built-in energy estimator in Contiki.

for the callbacks from the self tester for probing for problems, executing activators and handling the results are shown in Figure 3.

```
int probe();
void execute_activator();
void report(int activator, int probe, int percentage);
```

Fig. 3. The hardware self-test API.

With a few defined probes and activators, it is possible to call a self-test function that will run all probe/activator pairs. If a probe returns anything else than zero, this is an indication that a sensor or hardware component has reacted to the activity caused by the activator. The code in Figure 4 shows the basic algorithm for the self-test. The code assumes that the activator takes the time it needs for triggering potential problems, and the probes just read the data from the activators. This causes the self-test to monopolize the CPU, so the application can only call it when there is time for a self-test.

The self-test mechanism can either be built-in into the OS or a part of the application code. For the experiments we implement a self-test component in Contiki on the ESB platform.

A component on a node can break during the network's execution (we experienced complete breakdown of a node due to severe physical damage). In such case the initial self-test will not automatically detect the failure. Most sensor network applications have moments of low action, and in these cases it is possible to re-run the tests or parts of the tests to ensure that no new hardware errors have occurred.

```
/* Do a self-test for each activator */
for(i = 0; i < activator_count; i++) {
  /* Clear the probes before running the activator */
  for(p = 0; p < probe_count; p++) {
    probe[p]->probe();
    probe_data[p] = 0;
  }
  for(t = 0; t < TEST_COUNT; t++) {
    /* run the activator and probe all the probes */
    activator[i]->execute_activator();
    for(p = 0; p < probe_count; p++)
      probe_data[p] += probe[p]->probe() ? 1 : 0;
  }
  /* send a report on the results for this activator-probe pair */
  for(p = 0; p < probe_count; p++)
    report(i, p, (100 * probe_data[p]) / TEST_COUNT);
}
```

Fig. 4. Basic self-test algorithm expressed in C-code.

4.2 Software Self-Monitoring

Monitoring the hardware for failure is taking care of some of the potential problem in a sensor network node. Some bugs in the software can also cause unexpected problems, such as the inability to put the CPU into low power mode [8]. This can be monitored using Contiki's energy estimator [5] combined with energy profiles described by the application developer.

```
ENERGY_PROFILE(60 * CLOCK_SECOND,      /* Check profile every 60 seconds */
               energy_profile_warning, /* Call this function if mismatch */
               EP(CPU, 0, 20),         /* CPU 0%-20% duty cycle */
               EP(TRANSMIT, 0, 20),    /* Transmit 0%-20% duty cycle */
               EP(LISTEN, 0, 10));     /* Listen 0%-10% duty cycle */
```

Fig. 5. An energy profile for an application with a maximum CPU duty cycle of 20 percent and a listen duty cycle between 0 and 10 percent. The profile is checked every 60 seconds. Each time the system deviates from the profile, a call to the function *energy_profile_warning* is made.

4.3 Self-Configuration

Based on the information collected from the hardware and software monitoring the application and the operating system can re-configure to adapt to problems. In the case of the surveillance application described above the application can turn off the PIR sensor during radio transmissions if a PIR hardware problem is detected.

5 Evaluation

We evaluate the self-monitoring architecture by performing controlled experiments with nodes that have hardware defects and nodes without defects. We also introduce arti-

ficial bugs into our software that demonstrate the effectiveness of our software self-monitoring approach.

5.1 Detection of Hardware Problems

For the evaluation of the hardware self-testing we use one probe measuring PIR interrupts, and activators for sending data over radio, sending over RS232, blinking leds and beeping the beeper. The probes and activators are used to run the tests on ten ESB nodes of which two are having the hardware problems.

A complete but simplified set of probes, activators and report functions is shown in Figure 6. In this case, the results are only printed instead of used for deciding how the specific node should be configured.

```
/* A basic PIR sensor probe */
static int probe_pir(void) {
  static unsigned int lastpir;
  unsigned int value = lastpir;
  lastpir = (unsigned int) pir_sensor.value(0);
  return lastpir - value;
}

/* A basic activator for sending data over radio */
static void activator_send(void) {
  /* send packet */
  rimebuf_copyfrom(PACKET_DATA, sizeof(PACKET_DATA));
  abc_send(&abc);
}

/* Print out the report */
static void report(int activator, int probe, int trigged_percent) {
  printf("Activator %u Probe %u: %u%%\n", activator, probe, trigged_percent);
}
```

Fig. 6. A complete set of callback functions for a self test of radio triggered PIR sensor.

As experienced in our two deployments, the PIR sensor triggers when transmitting data over the radio during the tests on a problem node. On other nodes the PIR sensors remain untriggered. Designing efficient activators and probes is important for the self-monitoring system. Figure 7 illustrates the variations in detection ratio when varying the number of transmitted packets and packet sizes during execution of the activator. Based on these results a good activator for the radio seems to be sending three 50 bytes packets.

5.2 Detection of Software and Configuration Problems

Some of the problems encountered during development and deployment of sensor network software are related to minor software bugs and misconfigurations that decrease the lifetime of the network [8]. Bugs such as missing to power down a sensor or the radio chip when going into sleep mode, or a missed frequency divisor and therefore higher sample rate in an interrupt driven A/D based sensor can decrease the network's

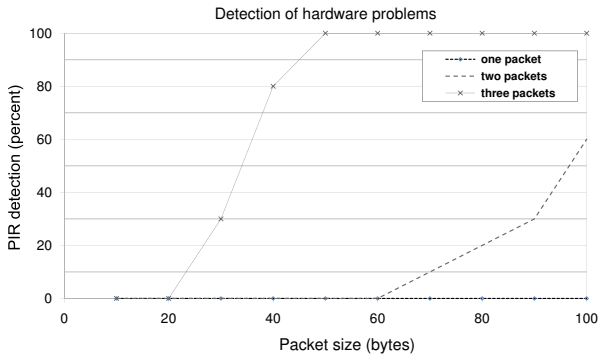

Fig. 7. Results of varying the activator for sending radio packets on a problem node. Sending only one packet does not trigger the PIR probe, while sending three packets with a packet size larger than 50 bytes always triggers the problem. On a good node no PIR triggerings occur.

expected lifetime. We explicitly create two software problems causing this type of behavior.

The first problem consists of failing to turn off the radio in some situations. Figure 8 shows the duty cycle of the radio for an application that periodically sends data. XMAC [1] is used as MAC protocol to save energy. XMAC periodically turns on the radio to listen for transmissions and when sending it sends several packet preambles to wake up listeners. Using the profile from Figure 5 a warning is issued when the radio listen duty cycle drastically increases due to the software problem being triggered.

In the second problem the sound sensor is misconfigured causing the A/D converter to run at twice the desired sample rate. The node then consumes almost 50% more CPU time than normal. This misbehavior is also detected by the software self-monitoring component.

6 Related Work

During recent years several wireless sensor networks have been deployed the most prominent being probably the one on Great Duck Island [9]. Other efforts include glacier [11] and water quality monitoring [2]. For an overview on wireless sensor network deployments, see Römer and Mattern [13].

Despite efforts to increase adaptiveness and self-configuration in wireless sensor networks [10, 12, 16], sensor network deployments still encounter severe problems: Werner-Allen et al. have deployed a sensor network to monitor a volcano in South America [15]. They encountered several bugs in TinyOS after the deployment. For example, one bug caused a three day outage of the entire network, other bugs made nodes lose time synchronization. We have already mentioned the project by Langendoen that encountered severe problems [8]. Further examples include a surveillance application called “A line in the sand” [6] where some nodes would detect false events exhausting

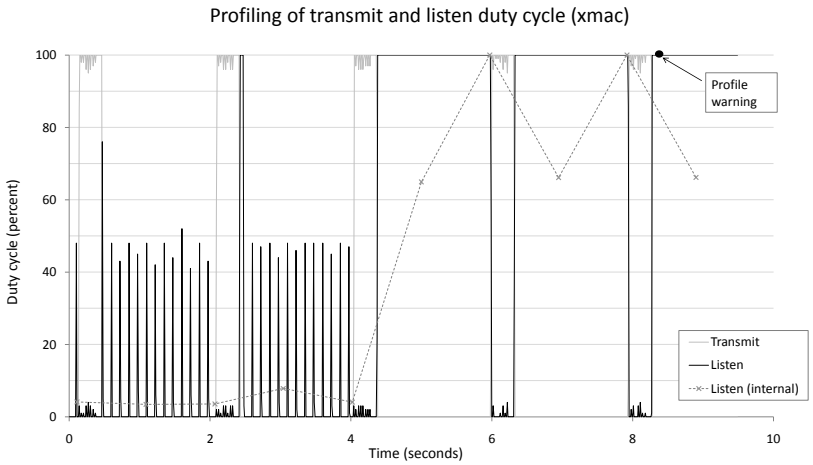

Fig. 8. Duty-cycle for listen and transmit. The left part shows the application's normal behavior with a low duty cycle on both listen and transmit. The right part shows the behavior after triggering a bug that causes the radio chip to remain active. The dashed line shows the listen duty cycle estimated using the same mechanism as the energy profiler but sampled more often. The next time the energy profile is checked the deviation is detected and a warning issued.

their batteries early and Lakshman et al.'s network that included nodes with a hardware problem that caused highly spatially correlated failures [7]. Our work has revealed additional insights such as a subset of nodes having a specific hardware problem where packet transmissions triggered the motion detector.

7 Conclusions

In this paper we have reported results and experiences from two sensor network surveillance deployments. Based on our experiences we have designed, implemented and evaluated an architecture for detecting both hardware and software problems. The evaluation demonstrates that our architecture detects and handles hardware problems we experienced and software problems experienced during deployments of other researchers.

Acknowledgments

This work was funded by the Swedish Defence Materiel Administration and the Swedish Agency for Innovation Systems, VINNOVA.

References

1. Michael Buettner, Gary V. Yee, Eric Anderson, and Richard Han. X-mac: a short preamble mac protocol for duty-cycled wireless sensor networks. In *ACM SenSys*, November 2006.

2. T.L. Dinh, W. Hu, P. Sikka, P. Corke, L. Overs, and S. Brosnan. Design and Deployment of a Remote Robust Sensor Network: Experiences from an Outdoor Water Quality Monitoring Network. *IEEE Congf. on Local Computer Networks*, pages 799–806, 2007.
3. A. Dunkels. Full TCP/IP for 8-bit architectures. In *Proceedings of The First International Conference on Mobile Systems, Applications, and Services (MOBISYS '03)*, San Francisco, California, May 2003.
4. A. Dunkels, B. Grönvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Proceedings of the First IEEE Workshop on Embedded Networked Sensors (IEEE Emnets '04)*, Tampa, Florida, USA, November 2004.
5. Adam Dunkels, Fredrik Österlind, Nicolas Tsiftes, and Zhitao He. Software-based on-line energy estimation for sensor nodes. In *EmNets '07: Proceedings of the 4th workshop on Embedded networked sensors*, pages 28–32, 2007.
6. Anish Arora et al. A line in the sand: a wireless sensor network for target detection, classification, and tracking. *Computer Networks*, 46(5):605–634, 2004.
7. L. Krishnamurthy, R. Adler, P. Buonadonna, J. Chhabra, M. Flanigan, N. Kushalnagar, L. Nachman, and M. Yarvis. Design and deployment of industrial sensor networks: experiences from a semiconductor plant in the north sea. In *ACM SenSys*, pages 64–75, 2005.
8. K.G. Langendoen, A. Baggio, and O.W. Visser. Murphy loves potatoes: Experiences from a pilot sensor network deployment in precision agriculture. In *14th Int. Workshop on Parallel and Distributed Real-Time Systems (WPDRTS)*, apr 2006.
9. A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless sensor networks for habitat monitoring. In *First ACM Workshop on Wireless Sensor Networks and Applications (WSNA 2002)*, Atlanta, GA, USA, September 2002.
10. PJ Marron, A. Lachenmann, D. Minder, J. Hahner, R. Sauter, and K. Rothermel. TinyCubus: a flexible and adaptive framework sensor networks. *EWSN 2005*.
11. P. Padhy, K. K. Martinez, A. Riddoch, H. Ong, and J. Hart. Glacial environment monitoring using sensor networks. In *Proc. of the Workshop on Real-World Wireless Sensor Networks (REALWSN'05)*, Stockholm, Sweden, June 2005.
12. I. Rhee, A. Warrier, M. Aia, and J. Min. Z-MAC: a hybrid MAC for wireless sensor networks. *ACM SenSys*, pages 90–101, 2005.
13. K. Römer and F. Mattern. The design space of wireless sensor networks. *IEEE Wireless Communications*, 11(6):54–61, December 2004.
14. J. Schiller, H. Ritter, A. Liers, and T. Voigt. Scatterweb - low power nodes and energy aware routing. In *Proceedings of Hawaii International Conference on System Sciences*, Hawaii, USA, January 2005.
15. Geoff Werner-Allen, Konrad Lorincz, Jeff Johnson, Jonathan Lees, and Matt Welsh. Fidelity and yield in a volcano monitoring sensor network. In *Symposium on Operating Systems Design and Implementation (OSDI)*, Seattle, USA, 2006.
16. A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. *ACM SenSys*, pages 14–27, 2003.