Real-World Deployments of Sensor Networks: Practical Lessons for Researchers

Marcin Brzozowski(✉), Max Frohberg(✉), and Peter Langendoerfer(✉)

IHP, Im Technologiepark 25, 15236 Frankfurt (Oder), Germany
{brzozowski,frohberg,langendoerfer}@ihp-microelectronics.com

Abstract. To prevent other research groups from making the same errors as we did in real-world deployments of sensor networks, we share our experience from previous installations and lessons learned from them.

First of all, we are finally convinced to apply the KISS principle (Keep it Simple Stupid) also to real-world deployments, especially that as researchers we tend to prefer more complex and sophisticated ideas.

Second, researchers underestimate practical issues in outdoor installations, and it may lead to unexpected and time-consuming problems. For instance, we did not notice that the selected voltage regulator works reliably only in temperatures above zero degree Celsius, leading to packet losses and a long debugging process.

Third, apart from extensive software tests before deployment, our protocols and applications include also self-healing instructions. They detect software bugs on run-time and restart motes if needed.

With all these three major steps, we were able to run real-world sensor networks for several years without major problems, even when we do not have resources for efficient testing and debugging, the problem that most research groups have.

1 Introduction

There are several ways to ensure that hard- and software is working as expected, such as various system-tests, different behavioral simulations, and also testbeds. However, researchers usually cannot carry out such extensive tests with a limited budget of research projects. Nonetheless, WSN applications should work maintenance-free and reliable for a long time in real-world scenarios. But often they failed quite early due to some unpredicted practical issues. We suffered similar problems and in this paper we share our experience on real-world WSN deployments.

The major difference between scientific and commercial sensor networks is their maturity. Commercial WSN are mostly based on thoroughly tested hardware and software, sometimes not changed significantly throughout several years.

K. R. Chowdhury et al. (Eds.): WWIC 2018, LNCS 10866, pp. 42–53, 2018.
https://doi.org/10.1007/978-3-030-02931-9_4

On the contrary, scientific WSN include novel solutions and they resemble prototypes rather than products, due to limited testing and debugging resources. Nonetheless, even such WSN prototypes should also work reliably for a long time.

In this paper we present lessons learned from our deployments: how to install and run reliably WSN regardless of limited testing resources. We hope these lessons will help other research groups to avoid the errors we did and in this way make their scientific efforts more productive and efficient.

The main lesson we learned in our WSN deployments is: keep it simple. Simple solutions tend to have fewer errors, work more reliably than complex ones, and also are easier to maintain. Therefore, we replaced our complex medium access control protocol [4], which included advanced features like clock drift prediction, with a simpler solution based on preamble sampling [5].

Another problem with testing in that some bugs are hard to spot before the real deployment, because they do not occur under testing conditions [1]. For example, our motes deployed next to a highway collected weather/traffic conditions and forwarded it to the sink. Before that, we tested the complete setup in long-time simulations and then with our testbed. It worked without problems so we decided to deploy it along the highway. However, due to some minor bugs in the radio module and the routing protocol, not spotted during in-lab tests, our motes lost plenty of frames, leading to failures. To attack such problems, we include self-healing mechanisms in our software, which detect software bugs on run-time and restart the affected mote. These mechanisms include mainly watchdogs, assertions, periodic reset, introduced later in this work.

This papers is organized as follows. Section 2 lists similar works that provide practical information about real-world deployments of sensor networks. In Sects. 3, 4, and 5 we introduce major lessons we learned during our WSN deployments, for low power radio communication, location and hardware, and for software issues respectively. Finally, we conclude this work in Sect. 6.

2 Related Work

In our previous work [3] we presented techniques to deal with software bugs in long-living sensor networks. It includes two major steps: offline bug fixing, to fix software errors before deployment, and self-healing to detect bugs on run-time and restart motes. Section 5 shortly introduces these both techniques.

Our work [2] shows several methods to efficiently debug software for sensor networks, including drivers, protocols, and applications. In that paper we evaluated several debugging techniques, mainly their memory footprint and impact on the execution time.

Other research works showed experiences from real-world deployments of sensor networks. In Ref. [1] the authors noticed that the deployment is the time to face unexpected problems. The major observation made by the authors is KISS (*"Keep it Small and Simple"* or *"Keep it Simple Stupid"*), and we fully agree. Further, the authors explained that some bugs are hard to spot before the

real deployment, because they do not occur under testing conditions. Therefore, we add also self-healing code in our software for long-living WSN applications.

Other experiences from a real-world deployment, a sensor network monitoring a potato field, are presented in Ref. [7]. The authors confessed they neglected software testing, leading to huge problems in the runtime. They stressed the need of thorough testing of the sensor network, mainly using a testbed. One year later these authors started the same application again and presented results in Ref. [6]. This time, the authors kept in mind the KISS idea and made the system much simpler than before. For instance, the MAC protocol was much simpler and there was no routing protocol at all. This new, simple approach achieved much better results than a year before: the sink gathered about 51% of sensor readings this time, whereas it got only 2% readings a year before. After reading these works, we changed our attitude towards potential problems in the deployment and assumed the worst-case will happen. Therefore, we put even more effort on debugging and fixed most errors in the lab. The remaining bugs were handled with self-healing code.

3 Lessons Learned from Low-Power Wireless Communication

In this section, we present major problems we encountered in low-power wireless communication between motes and also highlight the lessons we learned from our mistakes.

3.1 Real-World Conditions Bring Scientific Results into Question

At the beginning of the project our partners selected several locations to install sensors for water monitoring (see Fig. 1). The distance between these locations span from about 200 m to about 23 km. We planned to cover the long distances between motes with multi-hop connections. Based on our RF measurements in this area, we would need about 4–5 extra motes, equipped with high-gain antennas, to support data transmission over 23 km.

Then, we learned from the operator they would rather prefer a simple solution, since the multi-hop setup includes hard accessible and insecure areas, such as top of the mountains, places without road infrastructure, and so on. It would lead to difficult maintenance works and a high risk of vandalism or stealing. Therefore, instead of providing a single multi-hop sensor network, we installed several single-hop networks, each network with its own cellular IP gateway (see Fig. 1).

Lesson Learned. Reasonable scientific ideas, a multi-hop network in this case, sometimes must not be applied to real-world applications due to external conditions, such as challenging topography or a risk of vandalism in our case.

It is better to discuss these issues with the local operators and technicians a long time before deployment.

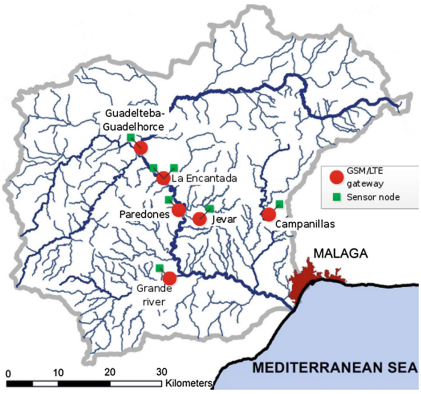

Fig. 1. We planned to deploy a multi-hop sensor network to monitor the complete area. However, due to some practical issues (a risk of vandalism, maintenance problems) we installed several 1-hop networks, each network with its own GSM modem.

Fig. 2. Avoid hard-accessible places, if possible. To achieve this platform, the technicians needed about 45-min boat drive. They installed the platform (incl. motes and a GSM modem) but had to come back here after a few days only to update software.

3.2 Assume Hardware May Break in a Way You Do Not Expect

During the project we encountered some unexpected hardware problems. For example, we assumed that the radio used by our motes cannot be partially broken: we expected either it works or not. At a certain location the mote was placed at the bridge located about 100 m away from the sink. This setup worked without major problem over several months, and then sink stopped receiving packets from the mote. On visiting this location, we noticed that the mote and the sink were still powered on so the problem was not the battery. Since the mote provided correct debug messages, we expected that the mote or the sink radio is broken. Indeed, the problem was the radio but after examining it with a spectrum analyzer we found out the mote was sending data with a too small power. After moving the mote closer to the sink, just for debug purposes, the sink started getting data. Clearly, we replaced the mote in this location and everything worked well again. This radio problem would be much harder to spot, if the sink were closer to the mote. In this case, only some packets would not reach the sink due to too weak signal.

Lesson Learned. We learned that hardware may break in a way we do not expect and to be more open-minded for potential problems in real-world deployments. Further, we also learn that a mobile spectrum analyzer speeds up fixing radio problems in outdoor installations.

3.3 Ensure You Do Not Change Anything After Final Tests

We had to install a water sensor in a hard-accessible location (see Fig. 2): the technicians needed about 45-min of boat drive. Since there was only a single sensor there, we could install a single mote connected directly to the GSM modem. However, we installed the mote and the sink (attached to the GSM modem), since this setup worked well in other places. To protect hardware from vandalism, this equipment was installed in a secure metal box. Unfortunately, we did not know the technicians used such a box, and learn about it afterwards, leading to the following problem.

On the installation day, we were checking motes remotely in our lab located about 2000 km away from the install location. After the technicians installed our motes, we informed the technicians to leave this location, as our server received data from the motes. Shortly after that, our server stopped receiving sensor data, but it still got control packets, so the GSM connection was still working.

After that, we discussed it with technicians and found out the cause of this problem: misunderstanding. Then, we learned that the sink and the mote are very close to each other, and the technicians closed the metal box just before leaving the location. As a result, the receive signal strength at the sink was too high, leading to packet losses. To solve this problem, we reduced the transmission power of the mote/sink, and the technicians updated software a few days later, on the next visit.

Lesson Learned. We learn to be very cautious in changing the setup after final tests, so that not even a single detail will be changed, including positions and installation instructions. Further, we learned again to avoid hard accessible places if possible, even when we assume that we will not have to visit this place again, for instance to update software. Moreover, wireless code update probably makes the maintenance of sensor networks much easier, provided wireless communication is still operable.

3.4 Read Operating Conditions Carefully

To enable low-power communication over long distances, we integrated off-the-she lf transceivers on a new board (Fig. 3). Then, we made some RF measurements, in the south of Spain, and this setup worked fine. Later we started another project and installed the same type of transceivers, but from another production lot, along a motorway in Germany. During first few weeks our motes worked flawlessly, but later on some problems with unexpected packet losses occurred. We suspected our new software to cause these problems, since hardware had already been tested in the previous project. We were trying to find the cause of this problem for several weeks, and finally realized that the assembled voltage regulators used on this production batch of PCB (printed circuit board) work only above 0 °C. Since the temperature dropped rapidly several weeks after installation, some motes stopped working. We did not spot this problem in the previous

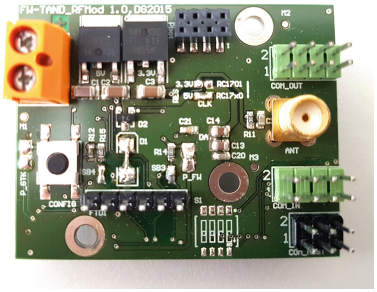

Fig. 3. Our long-range RF board worked reliably in tests in the south of Spain, but suffered from problems in the middle of Germany after several days of the deployment. Finally we found the reason: the voltage regulator did not work below 0 °C.

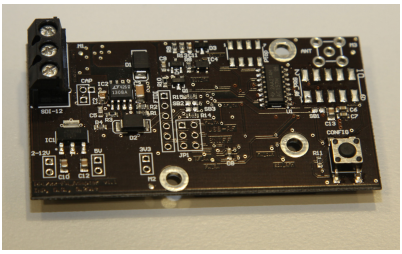

Fig. 4. We developed a new board for connecting SDI-12 sensors and were able to connect up to 10 sensors to it. Since SDI-12 sensors provide digital data, we did not encounter problems with analog-to-digital conversions again.

project, since the temperature in south of Spain was always above 0 °C. Further, we did not spot this problem while testing our motes for several hours in a freezer.

Finally, we replaced this voltage regulator with another one, with a much wider temperature range, and it fixed these problems.

Lesson Learned. We still cannot explain the reason for using a voltage regulator that operate only at temperatures above 0 °C. We probably did not specify the exact component, voltage regulator, and our PCB manufacturer might have selected another one, as we put an asterisk at the component name in the bill of materials (BOM).

We learn to double check the BOM and the operating conditions of all electronic parts used on our boards. Further, we must also check PCBs of newly assembled devices carefully and ask for the BOM of the components that really were used.

4 Low Power, Good Location for Hardware and Sensors

Apart from problems with low power wireless communication, we had also some issues with cellular IP modems and sensors. Here, we shortly introduce these problem and lessons we learned from them.

4.1 Standby GSM Router Is Not Low Power

To enable GSM communication at locations without mains power, we found a low-power GSM modem, which also included FTP and HTTP protocols, needed for our application.

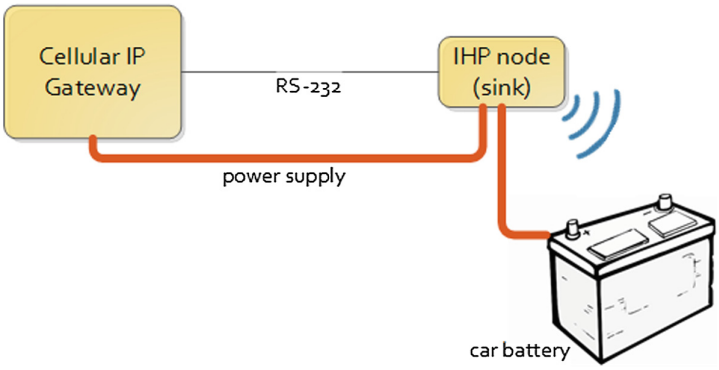

Fig. 5. The default low-power mode of the cellular IP gateway still consumes too much energy for low-power sensor application. Therefore, the sink powers down the GSM gateway completely when not used.

However, this modem still consumes too much power for our long-living application, about 21 mW, whereas our sinks needs 3000x less energy. Because of a low duty cycle of our application (about 100 kB of data transmitted every 15 min), we decided to switch off the modem completely when not used. To do so, the sink switched on the modem (see Fig. 5) only to transmit data, and switched it off afterwards. By doing so, the sink coupled with the GSM modem consumed less than 0.02 mW in standby, and achieved a lifetime of several months.

Lesson Learned. We learn that default low-power modes of various devices may still consume too much energy for long-living sensor network applications. Instead of using the device internal low power modes, switching it off completely may lead to energy savings, provided the power-down and power-up energy is not too high.

4.2 Don't Forget an Easy Accessible Placement of Hardware

At the beginning of the project, we used a GSM modem tailored for automotive application, with special antenna connectors. Since we could not find neither high-gain antennas nor cable adapters for such connectors, we installed the modem and its antenna high above the ground, on a mast. In this way, the modem benefited from a stronger GSM signal and was able to send data to the Internet. Since the modem was previously tested and used outdoors, we did not expect it to stop working, and it assured us to install the modem on the mast. Unfortunately, some problems with the modem occurred during the project, and we had to climb the mast to reach the modem. Then, we regretted we did not spend more time on looking for another antenna or connector that would allow us to install only the antenna on the mast and the modem close to the ground.

Lesson Learned. We learn that even if we are sure that nothing wrong will happen, because we tested or used our equipment previously for a long time, it may still break. Therefore, we have to keep it in mind when selecting the install location.

4.3 Periodic Reset

In our previous project, we did not have problems with limited energy and therefore the GSM modem was constantly powered on so. However, there was still a risk of power outage, up to 24 h without power, and in this time the modem should be running from an external car battery. For this scenario we selected a typical USB-dongle-based modem and connected it to the BeagleBone Black (BBB) embedded computer. In case of power shortage, this setup would work for several hours on a car battery.

Since the equipment was installed far from our office, we configured it to work autonomously. We provided several scripts that fix potential network or software problems on its own.

Again, we encountered problems that we did not foresee before. After some time, our server stopped getting data from the monitored area, although the complete setup was tested in our lab for a long time. First, it looked like the GSM connection was broken, because we could not log in remotely onto the embedded server. After examining the system on-site, we noticed that the cellular IP gateway could not be detected by the BBB, although the modem was still connected to it and nobody even touched it. We restarted the BBB computer and it detected the cellular modem without problems. We did not expect that the modem can just disappear from the system and our script could not handle this problem. Instead of adapting our script to deal also with this problem, we implemented a well-known, simple and robust solution: a watchdog[1]. In short, the hardware-based watchdog restarted the operating system of BBB once a day.

Lesson Learned. We learn that backup strategies are needed to deal with various hardware and software problems. For example, in this case a periodic reset or watchdog solves various temporary problems.

4.4 Do Not Forget About Redundancy

In the same application as in the previous paragraph, the BBB stored sensor readings on a SD card. After some time, our server was receiving some control frames from the BBB, but the sensor data was missing. We suspected the sink was broken and it did not forward sensor readings to the Internet server. After arriving to the monitored area, we examined the BBB computer and noticed that

[1] A watchdog is a counter that performs system reset on a certain value, mostly zero. However, if software clears the watchdog flag, the counters starts again. In our case, we did not clear the flag so the watchdog performed reset each time it finished counting.

SD card was not detected by the system anymore. Then, the BBB tried to write sensor data to the internal flash memory but it was full. After reinserting the SD card, the BBB detected it and saved sensor reading on the card. It showed us that a periodic reset, introduced in the previous paragraph, helps in various run-time problem. Further, we considered again adding an extra hardware drive to store data, in case the SD card breaks.

Lesson Learned. Here we learn not to forget to use redundant storage devices to safely save sensor readings. Further, we saw again the need of simple mechanism - a periodic reset - to deal with unpredictable problems.

4.5 Sensors: Do Not Reinvent the Wheel

During our first real-world deployment we considered how to design software and hardware to be reused among several different sensors. Then, we decided to use analog sensors, as they can easily be connected to a typical analog-to-digital converter (ADC). Further, since our mote already has an ADC available, we would not need to design new hardware for using such sensors.

Unfortunately, we suffered from some problems soon, because the embedded ADC precision was not good enough. Even by taking dozens of analog samples and averaging it, the error in digital values was higher than 10%, due to design imperfections. Further, in some locations we had to connect up to 12 sensors, but we could connect only a single analog sensor to our mote.

Nonetheless, we were still convinced to use analog sensors and developed a new board (Fig. 6) with connectors up to 12 sensors. Although this board provided more accurate results than the embedded ADC, we had to calibrate each board manually by defining some conversion constants in our application. Further, we were not sure if these constants need adapting later, for example when the outdoor temperature changes.

In the next project we got some water monitoring sensors from the project partner. We found out that most sensors provide only digital data and were afraid of writing several sensor drivers. However, it turned out that all these sensor support the SDI-12 protocol, which is quite popular in environmental monitoring. In the end, we needed only to implement this protocol and develop an SDI-12 adapter board (Fig. 4). After that, all SDI-12 based sensors can be connected to our mote, and a single mote can read data from 10 sensors. Further, we do not care about ADC conversion and calibration, we just get digital data. We can also add or remove sensors on runtime, which makes this setup even more flexible and useful.

Lesson Learned. Digital sensor readings are more reliable than our previous, ADC-based solution, and also simpler to maintain. So instead of struggling with several ADC issues, it is better to find and use standards that are already available.

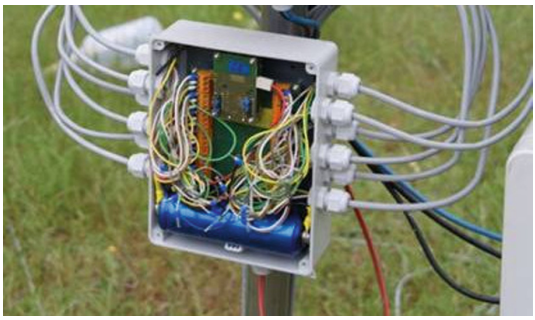

Fig. 6. Since the embedded analog-to-digital converter suffered from precision problems, and could not connect several sensors, we developed our own ADC board and connected up to 12 sensor to it (showed in this photo). However, we had to still manually calibrate the new ADC and did not know if the calibration is needed after some time, e.g. when the temperature changes. Therefore, in the next project we used sensors based on the digital SDI-12 protocol.

5 Software Development

In this section, we present our major findings for software problems running in long-term outdoor applications, introduced partially in our previous work [3].

5.1 Keep It Simple

In some of our previous projects we applied our schedule-based MAC protocol [4]. To make it work reliably in long-term applications, the protocol included quite complex mechanisms to deal with problems like beacon overlap, clock drift, and so on. Although the protocol was already running in outdoor applications, we kept finding minor problems in PC simulations. Thus, we were not convinced that the protocol has no bugs, and there was a risk that some undetected bugs affect motes while running outdoors. Further, because of high complexity it was not easy to add new features to the protocol, and to maintain it.

Because of all these problems we were looking for another, a simpler MAC protocol and decided to use a protocol based on preamble sampling [5]. It took us only a few weeks to implement the new protocol from scratch, and it was much simpler indeed. According to our calculations, this simple MAC would need a bit more energy in our application, caused by idle listening. However the extra penalty was small enough to start using this protocol and benefit from its simplicity.

Lesson Learned. Although it might be tempting to make complex, sophisticated solutions, especially in scientific communities, the real-world application need simple and robust approaches. We learned to make our design simple and benefit from easier maintenance, easier testing, and high reliability compared to the complex solutions.

5.2 Some Bugs Attack Only on Run-Time

To examine the impact of weather and traffic conditions on wireless communicatio n, we deployed our motes along a highway in Germany. Before starting our application outdoors, we performed several test operations: PC simulations, hardware tests, and finally WSN testbed. After fixing some minor problems our application worked without problems in simulations and on the testbed. Therefore, we installed motes along the highway but soon after they suffered from communication problems, leading to high packet error rates.

We examined the problem in our lab and found a bug in the radio driver. In short, this driver should filter incoming messages based on the destination address. However, sometimes this filter did not work properly and it forwarded incoming packets to the upper layers. It lead to various problems with our routing protocol, and in the end to packet losses.

This problem shows the risk already mentioned in Ref. [1]: some bugs cannot be spotted in in-lab tests, because they occur only on certain conditions. Therefore, we include in our software also self-healing instructions, which detect run-time problems and reset motes if needed. It is based mainly on hardware watchdogs, available on most microcontrollers, and also on software assertions. We provided more details about that in our previous work [3].

With all these means we run various WSN applications outdoors for more than three years without major software problems. Further, in our first deployment we were aware of some bugs in our medium access control protocol, but still could run the complete application for several weeks outdoors, thanks to the self-healing mechanisms.

Lessons Learned. Although software testing for sensor networks needs plenty of time and effort, it pays off in long-running application: fewer bugs means less maintenance effort. However, some bugs cannot be spotted before deployment, since they do not occur under testing conditions. To deal with this problem, software for sensor networks should include also self-healing instructions, which recover from software errors on run time.

6 Conclusion

In research projects we mostly concentrate on scientific work and do not have enough time and resources to deal with practical problems in real-world applications. Nonetheless, these applications should operate for a long time, preferably without maintenance. In last years we learned how to achieve this goal, that is, run long-term applications with limited testing resources. In the following we summarize the lessons we learned from our previous WSN deployments:

- you must include plenty of time for test and verification into your project schedule
- apart from simulations and testbed you have to run outdoor tests with real world conditions

- do not forget backup strategy and redundancy, even when you think your components are tested
- read the hardware errata carefully and make sure you ordered the right components, for example the same revision that you tested
- make sure your project partner knows what you do and vice versa,
- close the communication and specification gap by clear and strict documentation
- for complex installations get the specification or build it up in advance
- consider how to reuse your previous work, such as implementation or hardware designs, to gain more time for testing

We hope these lessons will let other research groups to install real-world applications more efficiently by avoiding trial-and-error approach.

Acknowledgement. The research leading to these results was partly founded from the Federal Ministry of Education and Research (Germany), from the DIAMANT project.

References

1. Barrenetxea, G., Ingelrest, F., Schaefer, G., Vetterli, M.: The hitchhiker's guide to successful wireless sensor network deployments. In: Proceedings of SenSys 2008 (2008)
2. Brzozowski, M., Langendoerfer, P.: Overview and benchmarks of pragmatic debugging techniques for wireless sensor networks. In: Proceedings of SoftCOM 2013 (2013)
3. Brzozowski, M., Langendoerfer, P.: Bug-tolerant sensor networks: experiences from real-world applications. In: Mitton, N., Kantarci, M.E., Gallais, A., Papavassiliou, S. (eds.) ADHOCNETS 2015. LNICST, vol. 155, pp. 251–262. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25067-0_20
4. Brzozowski, M., Salomon, H., Langendoerfer, P.: Completely distributed low duty cycle communication for long-living sensor networks. In: Proceedings of EUC 2009 (2009)
5. El-Hoiydi, A.: Spatial TDMA and CSMA with preamble sampling for low power ad hoc wireless sensor networks. In: Proceedings of ISCC 2002 (2002)
6. Haneveld, P.K.: Evading murphy: a sensor network deployment in precision agriculture (2007). http://www.st.ewi.tudelft.nl/koen/papers/LOFAR-agro-take2.pdf
7. Langendoen, K., Baggio, A., Visser, O.: Murphy loves potatoes. In: Proceedings of IPDPS 2006 (2006)