The Hitchhiker's Guide to Successful Wireless Sensor Network Deployments

Guillermo Barrenetxea, François Ingelrest, Gunnar Schaefer, and Martin Vetterli
LCAV, EPFL, Switzerland
firstname.lastname@epfl.ch

ABSTRACT

The successful deployment of a wireless sensor network is a difficult task, littered with traps and pitfalls. Even a functional network does not guarantee gathering meaningful data. In SensorScope, with its multiple campaigns in various environments (*e.g.*, urban, high-mountain), we have acquired much knowledge in planning, conducting, and managing real-world sensor network deployments. In this paper, we share our experience by stepping through the entire process, from the preparatory hard- and software development to the actual field deployment. Illustrated by numerous real-life examples, excerpted from our own experience, we point out many potential problems along this way and their possible solutions. We also indicate the importance of a close interaction with the end-user community in planning and running the network, and finally exploiting the data.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Distributed Networks.

General Terms

Design, Experimentation.

Keywords

Architecture, Deployment, Environmental Monitoring, Implementation, Wireless Sensor Network.

1. INTRODUCTION

Although most theoretical aspects of wireless sensor networks (WSNs) have been well studied over the past few years (*e.g.*, synchronization [12], routing [19]), real-world deployments still remain a challenging task. All too often, good WSN systems fail to provide expected results once deployed in the real world. Such failures may be either due to a completely non-working system or an inability to meaningfully exploit gathered data. While certain issues may be anticipated, experience is still the key asset to ensure a successful deployment. A few groups have already shared part of their experience in this field, ranging from habitat monitoring [15] to precision agriculture [7], to the study of tree canopy climate [17], to environmental monitoring [14, 18].

Three main areas exist, in which expertise is needed to access the "Holy Grail" of successful deployments:

- *Development* — The first step, during which local conditions, such as the expected weather in case of outdoor deployments, must be carefully studied and considered. Hardware must be well suited for the targeted site, while embedded software must be designed in a way that eases debugging later on.
- *Testing* — Ensuring that the system is ready to be deployed before going on site is mandatory, and setting up a testbed is often the best solution. Designing a good one, however, is not so easy.
- *Deployment* — Last but not least, the deployment is most often the time to face unexpected problems due to unanticipated or—even worse—underestimated issues.

Over the past three years, we have worked on SensorScope[1], an environmental monitoring system based on a WSN. We have engineered a complete framework, including electronic circuit boards, a solar energy system, an embedded communication stack, based on TinyOS [9], as well as server-side software. We have run six deployments, ranging in size from half a dozen to a hundred stations, from EPFL's campus to high-mountain sites. Throughout these campaigns, we have gathered experience in preparing, conducting, and managing deployments. As a case in point, we have deployed our system on a rock glacier located at 2 500 m (8 200 ft), on top of the Génépi, a mountain in the Swiss Alps. Although beautiful, this environment is rough and the deployment took place under very harsh conditions. It was, nevertheless, successful and led environmental scientists to the modeling of a microclimate, which has been causing dangerous mud streams [1].

In this paper, we share our experience and go through the aforementioned steps on the way to successful WSN deployments, detailing the problems we faced (*e.g.*, weather conditions, traceability) and the solutions we found. Although

[1] `http://sensorscope.epfl.ch`

SenSys'08, November 5–7, 2008, Raleigh, North Carolina, USA.

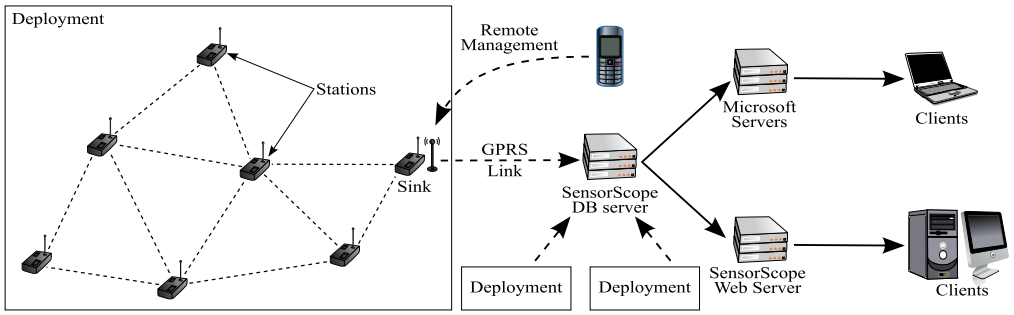


Figure 1: Overall SensorScope architecture. Deployments feature a GPRS-enabled sink that sends gathered data to a network socket on our database server, which, in turn, makes it available to other servers. Remote management of the sink is possible via GSM text messages.

SensorScope is aimed at outdoor deployments, many of the issues we describe in this paper are common to all kinds of deployments. We believe that our experience will be of interest to the community, and that our story will help other groups in anticipating many of the problems they are likely to face. In fact, this paper is written as a guide for readers aiming to deploy a WSN. It contains much advice, illustrated with many examples, all taken from our own experience.

In the next section, we bring up the difficulty of deploying a WSN, while in Sec. 3, we provide an overview of SensorScope and its various deployments. Sec. 4 is the guide to successful deployments, and Sec. 5 concludes this paper.

2. DEPLOYING WIRELESS SENSOR NETWORKS

Deploying a WSN has always been reported as a difficult task. One of the first known deployments was performed by a group at Berkeley in 2002, on Great Duck Island, to help habitat monitoring [10, 15]. While this was pioneering work, limited to single-hop communications, it helped in understanding the difficulty of coping with hardware failures and of correctly packaging sensors to protect them, while still getting correct readings.

A few years later, the same group reported results obtained from their new sensor network, Macroscope [17], built on top of TASK [2], a set of WSN software and tools, also designed at Berkeley. Macroscope has been extensively used for microclimate monitoring of a redwood tree. Despite building on their previous experience, they faced numerous issues, such as correctly calibrating sensors or detecting outliers. They recommended to build a sensor network well-suited to the application, to get the most robust system possible, and to carefully think about the deployment methodology.

Lessons stacked up with the increasing quantity of reported deployments, resulting in more and more successful data gathering campaigns. One of the most exciting experiences has been the study of an active volcano, measuring seismic and infrasonic signals, performed by a group at Harvard [18]. Still, even more advice and cautions were provided, for instance about the low quality of crystals in most sensor motes, or about the need for the highest possible degree of remote control of the deployment.

Nevertheless, new reports about failed campaigns keep appearing, demonstrating that WSN deployments remain a non-trivial task. One of them was reported by researchers at Delft University, who deployed a large-scale sensor network in a potato field [7]. Their goal was to improve the protection of potatoes against a fungal disease, by precisely monitoring its development. Unfortunately, the deployment went awry due to unanticipated issues, such as the bad interaction between many "external" software components, not suited to the targeted application. A lack of consistency and coordination between the team members was also reported. Overall, this paper remains a good advocate for carefully preparing deployments to avoid disastrous failures.

Finally, even very recent deployments still provide new insights into the methodology. For instance, researchers from the University of Virginia have reported their work on WSNs for environmental research [14]. LUSTER, their system, has been designed mainly to gather light measures. They emphasize the need to use monitoring tools, to assess the health of the system, as well as the need for deployment-time tools to ensure the system is up and running before leaving the field. Such a tool is actually already part of TASK [2].

All these examples cover a wide range of scenarios, and clearly demonstrate how much more difficult it is to deploy a WSN in the real-world rather than in a simulator. In this paper, we have brought together many of the aforementioned issues and advice, augmented with our own experience, to help at improving current deployment methodology.

3. SensorScope

SensorScope is an environmental monitoring system, based on a time-driven WSN. The network's sensing stations regularly transmit environmental data (*e.g.*, wind speed and direction) to a sink, which, in turn, uses a gateway to relay the data to a server. Depending on the deployment scenario and the available communication resources, we use different gateways: GPRS, Wi-Fi, or Ethernet. Data is published on our real-time *Google Maps*-based web interface[2] and on Microsoft's *SensorMap* website[3]. Figure 1 illustrates this architecture. Several deployments can occur at the same time, all data being sent to the same database server.

SensorScope is developed in collaboration between two research laboratories at EPFL: LCAV (signal processing and networking) and EFLUM (environmental fluid mechanics and hydrology). Our goal is to improve current environmen-

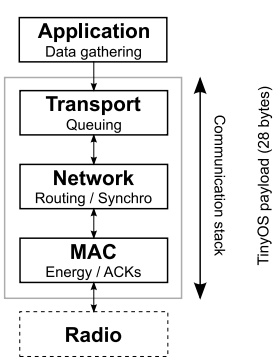
[2] `http://www.climaps.com`

[3] `http://atom.research.microsoft.com/sensormap`

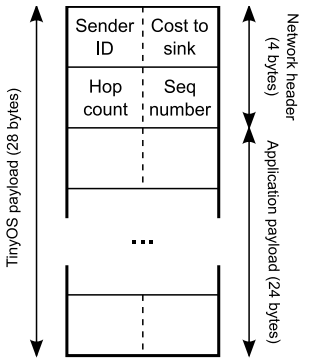
(a) Sensing station.

(b) Sensor box.

Figure 2: Design of our sensing station.



(a) The stack.



(b) Packet format.

Figure 3: SensorScope communication stack.

Table 1: TinyNode characteristics.

Component	Characteristic	Value
CPU	Type	MSP430 (16-bit)
	Frequency	8 MHz
Memory	ROM	48 KB
	RAM	10 KB
	Flash	512 KB
Radio	Type	Semtech XE1205
	Frequency	868–870 MHz
	Bit rate	76 Kbps
	Range	Up to 500 m (15 dBm)

tal data collection techniques, commonly based on a single, expensive station (± €60 000 is a common price). Furthermore, such stations use data loggers with limited capacity, requiring manual on-site downloads. Using a WSN is highly relevant to this area of research, as it allows for real-time feedback (*e.g.*, storms, pollution) as well as long-term monitoring (*e.g.*, snow level) in areas of varying size.

In this section, we survey both the hardware and the network architecture we have developed for the project. More in-depth details have been previously published [1]. We also describe the various deployments we have performed so far.

3.1 Hardware

We use a Shockfish TinyNode[4] sensor mote, whose characteristics are given in Table 1. We opted for this platform because of its long communication range and its low power consumption. A transmission power of 15 dBm allows for a communication range of up to 500 m with the on-board antenna, and up to 1 km using an external quarter-wavelength omnidirectional antenna [4].

To allow for long-term deployments, we designed a complete solar energy system. It is composed of a solar panel and two rechargeable batteries, one of them being a backup buffer. Stations are equipped with seven sensors, measuring nine environmental quantities: air temperature and humidity, surface temperature, solar radiation, wind speed and direction, soil water content and suction, and precipitation.

A sensing station, as depicted in Figure 2a, is composed of a 2 m high aluminum flagstaff, to which the solar panel and the sensors are fixed. The electronic circuitry is placed inside a hermetic box, shown in Figure 2b, which is also attached to the pole. The average price of a station is around €900. The price is kept down by using lower-end sensors. A key goal of the project is to obtain dense spatial measurements. This is achieved by deploying multiple low-cost—possibly less-accurate—sensing stations, rather than a single expensive, but very accurate one.

3.2 Network

To meet our requirements in multi-hop wireless networking, we have designed and implemented a communication stack for TinyOS 2.x, illustrated in Figure 3, that follows the OSI model and is freely available under an open-source license[5]. It stores four bytes per packet in the default 28-byte TinyOS payload, leaving 24 bytes for the application itself. We chose not to modify the TinyOS network headers for reasons of independence. There are four layers:

- *Application* — Collects data to be sent to the sink (*e.g.*, environmental measurements, battery levels).
- *Transport* — Creates, receives, and queues packets.
- *Network* — Takes all routing decisions (*i.e.*, choosing a next hop); synchronizes stations.
- *MAC* — Performs synchronous duty-cycling of the radio, similar to TASK [2], and acknowledges data packets to the previous hop; uses a random backoff mechanism to minimize packet collisions.

3.2.1 Neighborhood Management

Motes maintain a neighborhood table, in which they store the neighbors they can hear from. We chose an *overhearing* method in the spirit of MintRoute [19]: there are no dedicated neighborhood discovery packets; neighbors are discovered by listening to data traffic; the sink starts the discovery process by emitting beacons. A cost—currently the hop-distance to the sink—and a timestamp are associated with each neighbor. Each time a packet is captured, the table is updated. When a neighbor entry gets too old, it is

[4] `http://www.tinynode.com`

[5] `http://sensorscope.epfl.ch/network_code`

Table 2: All SensorScope deployments conducted since the beginning of the project. The first two deployments were limited to single-hop communications, while the others used multi-hop communications.

Place	Duration	Size	Data Points	Characteristics
Campus of EPFL	6 months (2006)	97 stations	190 000 000	Large-scale WSN
Plaine Morte	4 days (2007)	13 stations	1 300 000	Rapid deployment
Morges	1 month (2007)	6 stations	750 000	First use of multi-hop
Génépi	2 months (2007)	16 stations	5 800 000	High-mountain rock glacier
Grand St. Bernard	1.5 month (2007)	17 stations	4 300 000	High-mountain pass
Wannengrat	Ongoing (2008)	20 stations	*n/a*	High-mountain ridge
Campus of EPFL	Ongoing (2008)	10 stations	*n/a*	Outdoor testbed

removed from the table. To avoid sending packets to poorly-connected neighbors, a link quality measure is maintained for each neighbor. The measure is based on a count of the missing sequence numbers of overheard packets.

3.2.2 Synchronization

To allow for meaningful exploitation, gathered data must be time-stamped by the nodes, as part of the sensing process. Because our power management mechanism relies on duty-cycling, we opted for global synchronization of all motes. We use `SYNC_REQUEST/SYNC_REPLY` messages to propagate the local time of the sink (the *network time*), so that all nodes share its clock. When a node wants to update its clock, it sends a request to a neighbor closer to the sink than itself. This neighbor, if it knows the network time, broadcasts it back, and all receivers that are further from the sink update their clock. By doing this, the network time always propagates *away* from the sink, which acts as the global reference. Time-stamping of synchronization requests is done at the MAC layer to avoid delay errors [5]. The sink regularly sends its local time to the server, which can compute the offset between the network time and the actual time. The entire process is repeated regularly, so that all nodes stay synchronized with the sink, even if their own clocks drift. Hence, adjusting timestamps at the server always provides the correct absolute time.

3.2.3 Power Management

Even with solar energy, power management at the MAC layer is essential for long-term deployments, as the radio chip is a greedy energy consumer. Turning on the radio of a TinyNode increases its energy consumption approximately eightfold. We opted for a synchronous duty-cycling scheme [2], rather than an asynchronous low power listening approach [11]. We made this decision based on interactions with EFLUM, which allowed us to determine that overall data traffic would be low. Nodes wake up at approximately the same time thanks to the previously described synchronization mechanism, which is precise enough for this purpose. Nodes which are not yet synchronized with the rest of the network keep their radio on, until they acquire the network time. To account for clock drift, nodes wait a bit at the beginning of their active state to make sure that their neighbors are indeed awake. Because TinyOS is based on events, the division into communication cycles is transparent to other layers. When a packet has to be sent while the radio is off, it is queued until the next active state; the network layer simply waits for the TinyOS `sendDone` signal as usual.

3.2.4 Routing

To route data to the sink, we chose a randomizing solution. Each time a packet has to be routed, the forwarding node randomly selects a next hop between the neighbors closer to the sink, as opposed to MintRoute [19], where nodes are always connected to their best parent. To give priority to the better neighbors, two thresholds, based on link quality, are used: when a packet has to be routed, a random *good* neighbor is selected, if there is one; otherwise, a random *fair* neighbor is chosen [1].

3.3 Deployments

Over the past years, we have conducted six deployments (see Table 2), ranging in size from 6 to 97 stations, from the EPFL campus to high-up in the Alps. During these campaigns, we have gathered hundreds of megabytes of environmental data, freely available for download on our website[6].

We started with a large "backyard" deployment on our campus, using a simple single-hop routing protocol with several sinks and Ethernet gateways. This deployment allowed us to extensively test the hardware (*e.g.*, solar energy system, sensor board, sensors), while avoiding the added complexity of multi-hopping.

In a second step, we deployed our system on an alpine glacier. The aim of this deployment was to gather atmospheric measures during a highly stable winter period. As these events are rare and unpredictable, they require a rapid deployment, when the moment arises. Due to the small size of the deployment, we again employed our single-hop communication protocol, this time, however, with a GPRS gateway at the sink. We proved the feasibility of a rapid deployment and also tested our system under very harsh conditions (*e.g.*, extreme temperature values and variations, icing).

Once confident in our hardware and software, we concentrated on networking. The first deployment that required multi-hopping consisted of six stations located along a river bank, to monitor the air temperature and humidity, as well as the water temperature. The aim of this research project, financed by the Swiss Federal Office for the Environment, is to counter the effects of river-warming by systematically planting vegetation along river banks.

These successful deployments encouraged us to engage in more challenging ones: long-term deployments in remote and difficult-to-access high-mountain sites. Swiss authorities in charge of risk management asked us to deploy our system in two potentially dangerous places: the Génépi rock glacier and the Grand St. Bernard pass. The former is infamous

[6] `http://sensorscope.epfl.ch/environmental_data`

for being the source of dangerous mud streams that have already caused severe damage, while the latter is known as the "Combe des Morts" ("Canyon of the Dead"). Both deployments required multi-hop communications to cover the area of interest and a GPRS gateway to relay data to the server. Moreover, due to the remote location of both sites, in-situ maintenance had to be minimized (the Génépi rock glacier is reachable either by helicopter or by a strenuous five-hour hike). Each deployment enabled the development of an accurate hydrological model of the respective site, which could not have been obtained with traditional measurement methods. The use of SensorScope provided spatially and temporally dense measurements, and resulting models will help in predicting avalanches and mud streams, thus preventing accidental deaths [1].

We are currently running a new deployment in Wannengrat, a 2 500 m high environmental observatory in Switzerland. One of the projects at this observatory studies the wind field using seven permanent meteorological stations, and employs their measurements to devise new models. Sometimes however, higher resolution measurements are needed to fully understand the complex processes, which are taking place. We have thus deployed a dense network of 20 stations.

4. THE HITCHHIKER'S GUIDE

Despite careful design and concerns about possible deployment issues, we still faced many problems. In the following, we try to outline what we have learned from our mistakes, by describing many of the problems we faced, how we solved them, and how they could have been avoided.

4.1 Hardware and Software Development

Development is the first step towards the construction of a new system. During this phase, it is of prime importance to ensure that both hardware and software fit the intended application, considering not only the expected results, but also the conditions, in which deployments will take place.

Consider Local Conditions

You must carefully investigate how local environmental conditions will affect your deployments.

Because we knew that our deployments were going to be outdoors, we studied, with the help of EFLUM, how weather conditions would impact our system. However, it is not always obvious how, possibly drastic, variations in temperature and humidity will affect hardware devices in general, so that a lack of testing under real conditions may lead to serious issues. For instance, we already knew that our Li-Ion battery should not be charged when the temperature is below freezing, as it could explode. We thus disabled its charging during the Génépi deployment so that only the main battery was charged. We nevertheless faced many hardware failures during that deployment. For instance, hydrologists brought a *disdrometer*, an expensive instrument that can distinguish between different kinds of rain by analyzing the water drops. It was supposed to be used as a high-quality benchmarking tool. It turned out that it worked only during a few days, because it was too cold on top of the mountain. Of course, we noticed that only at the end of the deployment, because the device uses a data logger and no one had the chance to look at its measures until it was too late.

It is, therefore, crucial to simulate the anticipated conditions as accurately as possible. Studying the impact of weather conditions on hardware devices may be done by using a climate chamber, in which arbitrary temperature/humidity conditions can be created. However, in most cases, basic tests inside a household freezer will expose potential points of failure.

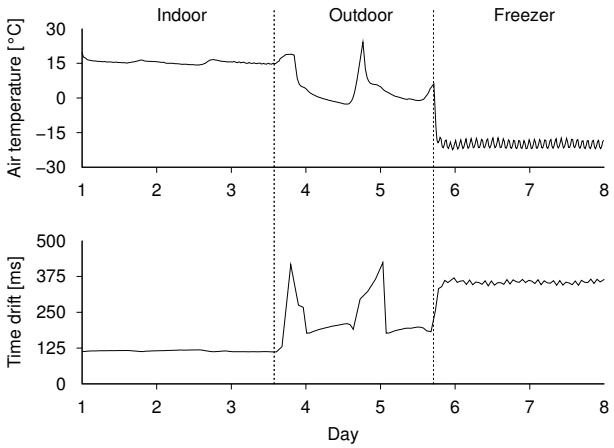


Figure 4: Time drift per hour of a TinyNode.

Time's a Drifter

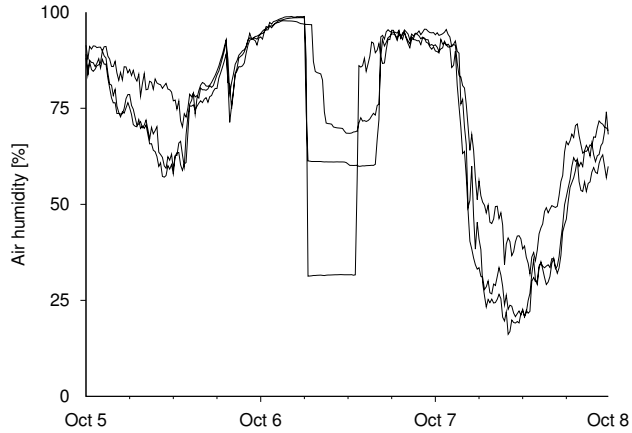
Keep in mind that the crystals used in sensor motes are imperfect and that temperature impacts their precision [18].

Figure 4 shows the time drift of a TinyNode relative to the air temperature. This experiment was conducted in three phases, during which the mote was placed successively indoors, outdoors, and inside a freezer. The drift was computed by subtracting the mote's local time from a global reference time every hour. We can see that the colder the temperature, the slower the crystal oscillates. Indoors, the drift is close to the data-sheet value (± 120 ms/h) while inside the freezer, the drift is much higher (± 375 ms/h). By looking at the outdoors part of the figure, it is clear that day/night cycles can be particularly challenging, and that rapid temperature changes greatly affect the time drift. Based on these results, we designed our protocols so that they could cope with such variations, *e.g.*, by accounting for drift when waking up nodes, rather than believing in their perfect synchronization.

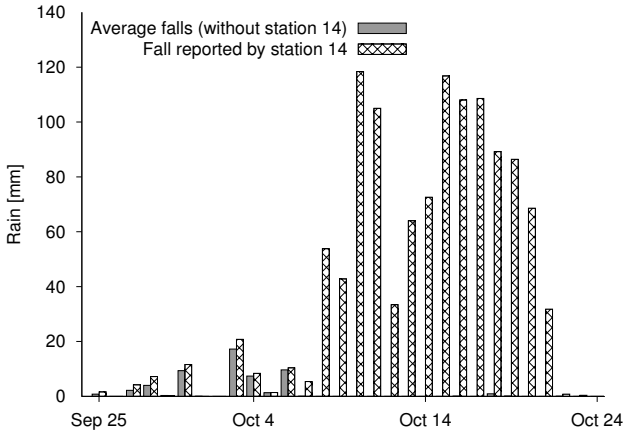
Hard Shell – Soft Core

If you target outdoor deployments, well thought-out packaging of sensors is of prime importance.

Packaging sensors for outdoor deployments is a difficult task, as it must protect electronic parts from humidity and dust while being unobtrusive at the same time [15]. International *Ingress Protection* (IP) codes are used to specify the degree of environmental protection for electrical enclosures. The required level for outdoor deployments is IP67, which provides full protection against dust and water, up to an immersion depth of one meter. With any lesser degree of protection, electronics, being susceptible to corrosion, are exposed to humidity and atmospheric contaminants, leading to irreparable damages. Corrosion may cause the malfunction of a sensor connection, consequently corrupting the data from that sensor. Even more disastrous, humidity may cause



(a) Three humidity measurements (Génépi). Due to corrosion, sensors fail when there is too much humidity



(b) Rain fall (Grand St. Bernard). A short circuit caused by previous rain falls corrupted the readings of station 14.

Figure 5: Two striking examples of bad measurements caused by the corrosion of sensors.

a short circuit in a sensor connector resulting in permanent damage and/or continuous rebooting of the affected mote.

For instance, the Sensirion SHT75 sensor we use to measure both air temperature and humidity comes unpackaged. It took us quite some time to figure out a suitable packaging, protecting from direct sunlight, while still letting the wind reach the sensor. Our solution is to use a stacked-plates structure and a small adapting circuit to embed the sensor inside. The wire connection to the sensor is additionally protected using epoxy resin. Despite all efforts, we faced many early sensor failures and node reboots during cold and humid periods, forcing us to review and improve our design several times. Figure 5a shows the humidity measures reported during the Génépi deployment by three different SHT75 sensors. Due to poor packaging, condensation settled on the connectors, corrupting measurements when the humidity was too high. This is clearly visible on Oct 6.

This particular sensor is not the only one that caused us problems, though. Another such example is illustrated in Figure 5b, which shows the average rain fall reported by all stations excluding station 14, as well as only that reported by station 14 during the Grand St. Bernard deployment. We can see that the rain meter of this station worked well during the first 12 days and reported erroneous values after that. The problem was again the corrosion of the connector to the sensor board, which caused a short circuit in the interrupt line, and, therefore, the reading of false precipitation values. Such failures are always to be expected with outdoor deployments.

There Is no Light at the End of the Tunnel

Do not forget that LEDs are big energy consumers.

On our motes, a single LED consumes about 3 mA. That makes a total of 9 mA for the typical three LEDs, while the radio chip, when on, consumes "only" 15 mA. There is thus no reason to efficiently manage the radio while carelessly using the LEDs. As they are the most useful debugging tools for WSN developers (and often the only ones), we have written a simple module implementing the TinyOS `Led` interface with empty functions. Using conditional compilation, this module is used during deployments, while the standard TinyOS module is used on our indoor testbed. Thus, the code remains the same, but calls to turn on/off the LEDs simply do nothing during deployments. With this trick, energy is conserved without error-prone manual code changes.

Keep It Small and Simple

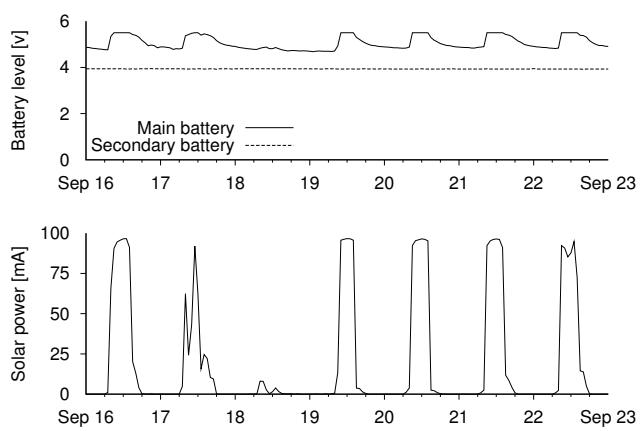
Follow Einstein's famous advice, "Everything should be made as simple as possible, but not simpler."

Both code and algorithms must be well-fitted to the intended application, in order to avoid unexpected interactions between software components as much as possible [2]. Sometimes, complexity cannot be avoided, but whenever the benefits are questionable, simple solutions should be preferred. For instance, as our stations contain energy consumers as well as batteries and solar panels, an overall positive energy balance is required and at the same time sufficient to achieve long-term autonomy. Fortunately, we were able to avoid complex, ultra low-power MAC layers, generally requiring high-precision synchronization, which may be impossible to achieve in realistic conditions, in favor of a very simple, predictable one. Furthermore, packet losses with such complex protocols are more likely to occur in harsh conditions (*e.g.*, heavy rain, intense cold), since that is when channel conditions degrade. However, such conditions are at the same time the most interesting episodes for environmental data analysis. The code of our communication stack (*i.e.*, transport, network, and MAC layers) is just over a thousand lines long, and is thus easy to read and maintain.

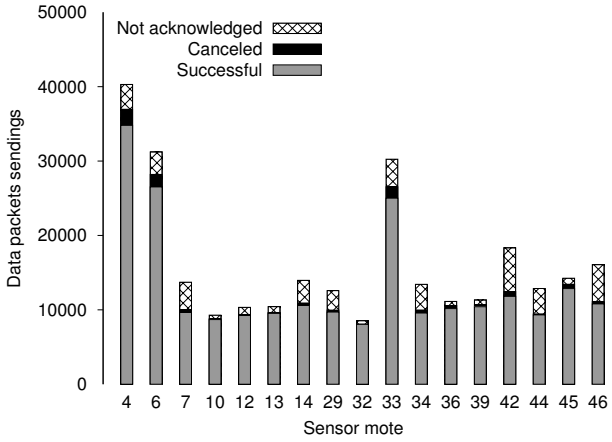
Remote Control

You should have as much remote control facilities on your deployments as possible.

If sensor motes are to be deployed in difficult-to-access places (and sometimes even in easy-to-access places), the ability to remotely control the deployment is highly desirable. When going back to the deployment site is difficult or costly, being able to adjust certain parameters remotely, such as the sampling frequency or the duty-cycle ratio, may be necessary [18]. More drastically, being able to reprogram



(a) One week of data from the solar energy system of a station (Génépi).



(b) Load distribution on our indoor testbed.

Figure 6: Two examples of how the SensorScope network can be monitored. Both plots, each showing one week of data, were generated from status packets, which are regularly emitted by the motes.

the motes of a deployment, without leaving the office, may be a desired feature.

When we developed SensorScope, we added routines to the software running on the GPRS module, which enable us to control it remotely, using simple GSM text messages, sent from a mobile phone. This allows us to query its status or to reboot either the GPRS or the sink's mote. We can also ask the GPRS to download a new version of its binary image from an FTP server, and to reboot using this new version. We still cannot, however, change parameters of the network, as this requires a mechanism to disseminate information from the sink, which is currently not implemented. We are also studying the possibility to support Deluge [6], an over-the-air programming mechanism that can be used to reprogram the motes. During the Génépi deployment, when we discovered a bug in the driver of the wind speed sensor, detailed below, such a mechanism would have been very useful.

Don't Be a Black Box

Keep in mind that programming embedded devices requires a different philosophy than traditional programming.

In traditional programming, code is easy to debug, using any kind of debugging statement or tool. It is far more difficult with sensor motes, as the simplest way for them to communicate with the outside world is by blinking their LEDs or using their serial port. These interfaces are not only limited, but also mostly unusable once a network is deployed. Moreover, embedded programs are more often subject to hardware failures than traditional software, so that their behavior can be incorrect, even if the code itself is actually fine [15].

It is thus of great importance to be able to determine what happens *inside* the network, and not only at the sink. This issue has already been pinpointed—and proved to be of prime importance—in previous work, but is unfortunately still widely underestimated in the WSN community [7, 14]. In SensorScope, besides traditional sensing packets, sensor motes generate three kinds of *status packets*:

- *Energy* — They contain the energy level of each battery, the incoming solar power, the current drawn by the system, and which battery is currently in use.
- *Network* — They contain statistics about the most recent activity of the transport layer, such as the number of data/control packets sent, the greatest size of the queues, or the quantity of non-acknowledged packets.
- *Neighborhood* — They contain a dump of the neighborhood table, including the identifier of each neighbor and its link quality. If the table does not fit into one packet, multiple packets are created.

Once the routing protocol is in place, it is easy to create such packets and to let them go to the sink—and thus to the server—just like sensing packets. Their frequency is low (four per hour), so that they do not have much impact on data traffic. Figure 6 provides two examples of how our system can be monitored. Figure 6a shows the energy level of the two batteries of a station during the Génépi deployment, as well as the incoming solar energy. This metadata allowed us to determine that the backup battery was never used, even in case of multiple, consecutive cloudy days. Due to this observation, our next-generation energy board will use only a single battery. Figure 6b provides the load distribution of the network on our indoor testbed, showing that nodes 4, 6, and 33 are the main hubs for routing packets to the sink.

Publish or Perish

You should plan your publications very early, to ensure that the required data will be gathered during deployments.

Our primary goal in SensorScope was to develop a working system and to succeed in collecting environmental data. Unfortunately, we forgot some of the issues related to publishing our results to the networking community, more or less thinking that successful deployments would be enough. This was a mistake, as our status packets have proved to be insufficient to extract some of the interesting data we would like

to publish now. For example, we have no statistics about the waiting time of individual packets at a given node, or about the time correlation of packet retransmissions. We should have actually gathered more data related to network conditions, not just environmental conditions! Once a network is deployed, it is usually too late to think about these issues. By planing publications early on, the code to gather needed data can be incorporated into the development process.

Choose Your Partner

Make sure to really know what to do with your data.

There are two components of a successful deployment: gathering the data and exploiting it. Generally, networking laboratories only care about the first component, while the second one actually plays an equal role. A WSN primarily exists to transport data from one point (*i.e.*, the targeted site) to another one (*e.g.*, a database), but there is no purpose in gathering data just for the sake of gathering it. The final objective of a WSN deployment is to gather data for an end-user. This end-user must be present in all the stages of the deployment preparation: from sensor selection, placement, and calibration, to data analysis [17].

In SensorScope, we (LCAV) work in close collaboration with EFLUM. We have frequent meetings to discuss the project, allowing us to tackle problems very pragmatically. For instance, when we learned that a sampling rate of less than two minutes was useless for environmental monitoring, we decided to omit network congestion management. Without such personal interaction, we may have ended up with solutions, potentially non-working, solving non-existing problems. EFLUM also led all decisions regarding deployments (*e.g.*, place, duration, observations), which allowed us to obtain nice results with a high scientific impact. For instance, if we had to choose ourselves, where to deploy our system, we would have never gone to the Génépi, and we would have certainly never discovered a microclimate there [1].

4.2 Testing and Deployment Preparation

To prepare for our deployments, we use two different testbeds. One is indoors, conveniently located in our building, used mostly to test our communication software. The second testbed is a pseudo real-world deployment, located on our campus, composed of actual sensing stations. This one is used to ensure that all code which is not in use on our indoor testbed (*e.g.*, sampling sensors, managing solar power) does not interfere with the rest.

Efficiency Matters

When setting up a testbed, keep in mind that it will be used to develop and to test many software components.

Because that is a long process, composed of many test-it-and-fix-it cycles, the testbed must be easily and quickly accessible. While programming motes one by one with a serial cable may be acceptable for deployments, because it is done only once, this is not the case for a testbed. Regularly replacing batteries is not a good idea either, and this will be necessary, even if some slick power-saving algorithm is used.

Our indoor testbed consists of 50 motes deployed throughout our building. As it is solely used to evaluate the network code, its motes are not wired to any external sensors. All of them are, however, plugged into AC power, allowing us to disregard any problems linked to energy management. Furthermore, all 50 motes are equipped with a Digi Connect ME[7] module which makes it possible to access and program them over a simple Ethernet connection. Each Digi module is indeed assigned an IP address which, in combination with the appropriate PC-side drivers, allows for transparent PC–mote serial communication. Such equipment is important to allow for quick testing. On our indoor testbed, it takes only between 10 and 45 seconds to flash all 50 motes (depending on the size of the image), while it takes about 50 minutes to flash the 10 motes of our outdoor testbed and to put them back onto the stations.

Labels that Stick

Carefully let other people know what you are doing.

Our indoor testbed is distributed among more than 40 offices in our building, some of them belonging to other laboratories. We frequently discover that motes have been disconnected, or have even disappeared, because people do not know what these *strange devices* are. When we installed our indoor testbed, we put stickers on the motes, stating that "this device belongs to LCAV, please contact ... for further information." Without such stickers (and sometimes even with) nothing will prevent colleagues from disconnecting a device and reclaiming that Ethernet plug. It even happens to us that people bring back a mote they found in their office, thinking it was somehow lost. Carefully explaining to co-workers what these devices are used for, and labeling them with informative, long-term messages, is not a waste of time. Trying to find out every week why some motes are no longer responding actually is a waste of time. The same, of course, applies to our outdoor testbed, as people may certainly be surprised to find a weather station on the terrace in front of their office.

Know Your Enemy

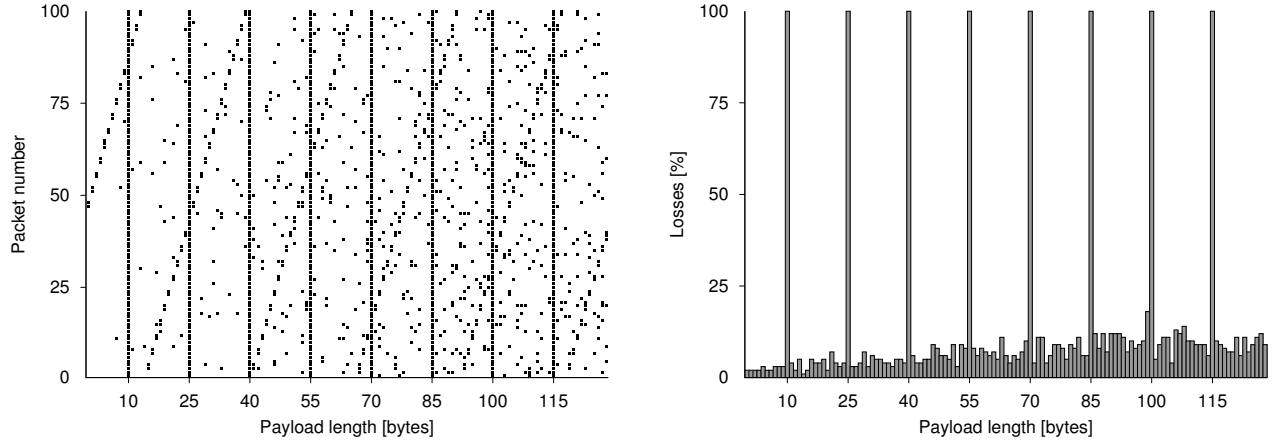
Make sure that your radio frequency is not already in use.

When setting up a deployment (especially indoors, or close to urban areas), the first order of business should be to inspect the radio spectrum used by your platform to detect possible interferences. The optimal way to do this is to use a spectrum analyzer. However, due to the analyzer size, weight, and power consumption, it is difficult to make use of it at the deployment site, and portable analyzers often do not provide the required resolution. Another solution would be to use a software-defined radio system (*e.g.*, GNU Radio[8]). Although preliminary work has already been performed in this area, the solution is not quite ready yet [13]. A simpler way to check for interferences is to run a test program to determine losses over time for the various frequencies that your selected platform can use. There are a lot of radio devices around these days that can create interference, compromising the results of test runs, and leading into thinking that your code is incorrect.

For instance, Figure 7 provides the packet losses observed during a test run on our indoor testbed. In this experiment, a run of 100 transmissions was started for each payload length, with an interval of two seconds between packets. Our primary goal was to see the receiving probability based on the payload length, but we quickly discovered that there was periodic interference at the frequency we were using. The loss pattern resulting from this interference (*i.e.*,

[7] `http://www.digi.com`

[8] `http://www.gnu.org/software/gnuradio`



(a) Temporal loss distribution. Each black square is a lost packet. Diagonal lines are losses caused by some interferences.

(b) Payload length-dependent loss distribution.

Figure 7: Packet losses on our indoor testbed over seven hours. A new run of 100 packets was started for each payload length, at a rate of one packet every two seconds. Losses when the payload length is equal to $(15x + 10)$ are caused by a bug in our radio drivers.

the diagonal lines) is clearly visible in Figure 7a. We tried to determine the source, but were finally unable to do so. We suppose that another laboratory is sometimes using the same frequency for its own purpose, as the interference is not always present.

One known source of interferences in our frequency band are cellular phones: the TinyNode operates in the 868–870 MHz band, which is close to the European GSM 900 band (890–915 MHz). We have performed some tests to determine whether cell phones would impact the performance of our testbed. It turns out that the first few seconds of an incoming call interfere with mote transmissions. In our tests, this happened when there was less than one meter between the phone and a mote, which is a frequent situation in offices. Fortunately, GSM calls are sparse enough, so that we can afford to simply ignore them in our indoor testbed results.

The Value of Simulation

Do not neglect how much simulation can help you.

In place of a testbed, or in addition to it, simulations can be used to test protocols. Many simulation tools are available, the most famous one certainly being ns-2[9]. However, such tools are very generic, and generally require the code of protocols to be specifically written for them. There are two major drawbacks: the code must be written twice, and simulation results do not provide insights into the quality of the "real" code to be deployed.

To overcome these issues, a new kind of simulation tool has recently been considered [3, 8, 16]: sensor mote *emulation*, rather than simulation. For instance, WSim loads the real target binary code and provides cycle-accurate results [3]. Thus, any sensor mote OS may be used in conjunction with it, and the code is written only once. Moreover, this code can be easily debugged and will work on the real system as long as it works with WSim. The downside is that only a few architectures have been ported so far. Such tools will, nevertheless, certainly replace simple packet-based simulators in a near future, as they fill the gap between theory and real-world deployments.

Data You Can Trust

Do not forget that your success not only depends on the amount of gathered data, but also on the quality of that data.

While most sensors are supposedly pre-calibrated, their installation may require manipulations (*e.g.*, mounting, packaging) that can affect the measurements [2]. An example of such a case is the aforementioned Sensirion SHT75, used to measure air temperature and humidity. While it is fully calibrated and should theoretically provide an accuracy of 0.3°C, inadequate packaging may skew its measurements. In SensorScope, all sensors, once packaged, are tested before deployments, first indoors, then outdoors. Readings are compared to high-precision reference stations over several days, and bad sensors (*i.e.*, correlation coefficient below 0.98) are simply discarded. During these tests, we detected sensors with an offset of more than 2°C, a significant error that would have invalidated scientific conclusions.

Calibration may also be required at the time of deployment: an example of this is the wind direction sensor, which must be precisely North-oriented to provide sensible data. We forgot this "obvious" detail in our first on-campus deployment, so that we had to return to each station to correct its orientation. Of course, wind direction data was almost unusable (*i.e.*, we would have had to document the wrong orientation of each station to correct the data later on), so that we had to drop it. Once a deployment is over, and sensors are back at the laboratory, it is important to repeat the calibration process. Doing so makes it possible to detect sensors which have been damaged during the deployment and, consequently, to flag the applicable data (see, for instance, Figure 5, showing corrupted measurements).

[9] `http://nsnam.isi.edu/nsnam/index.php/Main_Page`

Consistency Pays

It is very important that you use the same configuration during both tests and real deployments.

This may sound obvious, but at some point one may be tempted to change a few parameters, or to switch to a new version of the drivers just before a deployment, to improve a given aspect. With new versions, however, always come new bugs, and it is by far easier to detect them on a testbed than during a deployment. Another possible issue is the "last minute commit," which can kill a complete deployment [7].

For instance, Figure 7 also exposes a bug in our radio drivers: some payload lengths simply do not work, *i.e.*, packets are either not sent or not received. This occurs each time the sum of the payload length and the TinyOS header (five bytes) is a multiple of 15. During development, the payload length of some packets grew from a "valid" length to an "invalid" one, and it took us almost one full day of debugging to discover the bug. Obviously, we were glad it was discovered on our testbed rather than during an actual deployment, as it could have occurred only with the deployment drivers and not with the testbed ones, had they been different.

Practice Makes Perfect

Be aware that Murphy's law will do its best to make its way into your deployment [7].

If an ice-ax is needed to anchor the stations (as we do in high-mountain environments), chances are that none will be available. If a Phillips screwdriver is needed, chances are that only slotted ones will be at hand. Forgetting something may always happen, but the risk can be reduced by rehearsing the deployment close to home and making a checklist as you go along. This is one of the reasons why we have set up a permanent deployment on EPFL's campus. Not only does it help us in testing our code under real conditions, but it is also used as a practice deployment. To ensure that we do not forget anything, we ready all tools and hardware during the rehearsal, and put everything into bags which we take with us to the real deployments. The other advantage of having a close-to-real testbed is that it is much easier to show to visitors than a real deployment on top of a mountain!

4.3 Deployments

Once the system has been fully developed and thoroughly tested, it is ready to be deployed. Many pitfalls still remain ahead at this point, as even a working system may fail to produce the expected data and desired insights.

Consider Local Conditions – Again

Always be ready to face unforeseen bugs.

Some bugs can be hard to spot before the real deployment, simply because they do not occur under testing conditions. For instance, Figure 8 shows strange patterns in the time distribution of lost packets during the Grand St. Bernard deployment: it seems that sometimes packets from all stations were lost simultaneously. At first, we thought that the sink was at fault, and that it had rebooted for some reason, causing the loss of all queued packets. This was, however, quite strange, because packets are almost immediately forwarded to the GPRS, and a reboot would have caused the loss of at most one packet. After some investigation, we discovered that there was a bug in the GPRS drivers, preventing it from reconnecting to the cellular network upon a loss of connectivity. Due to this bug, we remotely rebooted

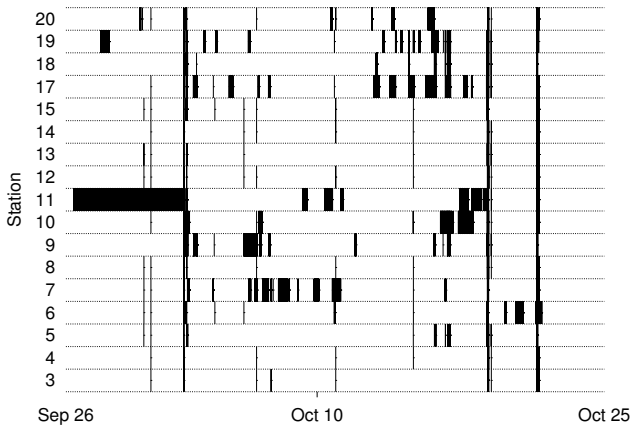


Figure 8: Time-correlated packet loss (Grand St. Bernard). Each vertical black line represents a lost packet.

the GPRS a few times during the deployment, using GSM text messages. Although GPRS reboots were already anticipated and all packets waiting to be transmitted were stored in flash memory, we still faced losses. After its reboot, the GPRS was sometimes so busy reading packets from the flash memory that it could not handle arriving messages, causing their loss. The thickness of the resulting lines in Figure 8 thus depends on the time it took us to notice the disconnection and to reboot the GPRS. We had never faced this bug during our tests, simply because cellular connectivity on our campus is very good, so that a disconnection never occurred. Needless to say that the connectivity is rather poor on top of the mountain.

The aforementioned time drift also caused some GPRS-related losses. We discovered that the crystal of the sink's mote and the crystal of the GPRS chip react differently to temperature variations. Communications between the mote and the GPRS occur over a serial bus. The drift caused by rapid temperature changes (*e.g.*, during sunrise on an alpine glacier, we observed temperature gradients of up to 10°C per hour) resulted in a loss of synchronization between the mote and the GPRS chip, and thus in lost packets. This is quite ironic, as a serial bus seems robust compared to wireless communications. At the time, however, the serial link—as opposed to the radio link—was lacking an acknowledgment mechanism, which we have now added. We could actually have detected this problem earlier by simply putting the GPRS into a freezer, as indicated earlier in this paper. *A posteriori* tests have indeed shown that after a few minutes in the freezer, most of the serial packets are lost due to a lack of synchronization.

For outdoor deployments, manipulating electronic parts in the field must be kept to a minimum. The hermetic box we use effectively prevents the corrosion of the electronic components inside, but it has a major flaw, which we had not noticed until we encountered bad weather at a deployment site: we need to open the box to replace a sensor. Indeed, while all sensors are external, they are directly plugged into the sensing board, within the box. Thus, when we detect a broken sensor, we need to open the box to replace that sensor. This is a problem in the field, especially when weather conditions are poor (*e.g.*, fog, snow). Hence, our next gen-

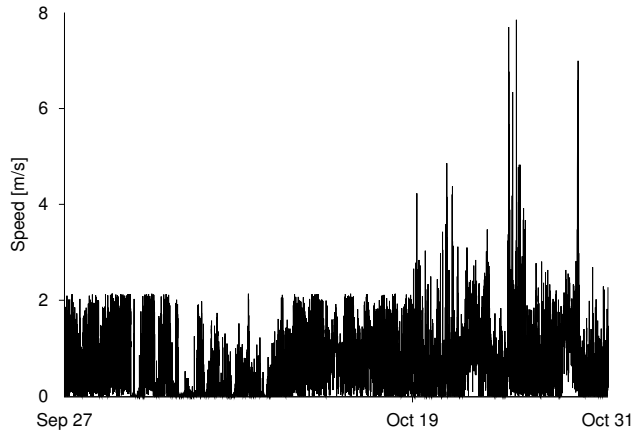


Figure 9: Wind speed measurements (Génépi). Due to a design problem in the sensor's driver, data gathered before Oct 19 is unusable.

Figure 10: Picture taken by an autonomous camera during the Génépi deployment. Weather conditions such as fog and snow are difficult to detect without visual feedback.

eration of stations will feature a better box with external, hermetic plugs, thus easing in-field sensor replacement.

The Barking Watchdog

Do not blindly believe in the quality and correctness of incoming data.

Our deployments have taught us that all data must be scrutinized as soon as it reaches the server, to promptly detect arising problems and malfunctions (*e.g.*, broken sensor, failing sink). When appropriate, the network operator should be automatically notified (*e.g.*, by emails or GSM text messages). However, while it may seem easy to detect obviously incorrect measures (see Figure 5b, showing the malfunction of a rain meter), other problems may be more subtle.

For instance, Figure 9 provides the wind speed reported by station 6 during the Génépi deployment. At first glance, the data may look correct, while it is actually not. Until Oct 19, because of a bug in the sensor's driver, all wind speed data is unusable. On Oct 19, we had detected and corrected the bug, and went back to the deployment site to update all stations. The explanation of this bug is simple: each time the anemometer completes a revolution, an interrupt is fired, which in turn increases a counter. Each time the sensor is queried, the speed is computed by multiplying the counter value by the distance represented by one revolution. To conserve memory, we used an 8-bit counter, which turned out to be too small. In the two minutes between consecutive sensor queries, the anemometer could complete more than 255 revolutions, resulting in a counter overflow and, hence, unusable data. Previously, as we had always queried the sensor every 30 seconds, this bug never occurred. While we are guilty of having broken one of our aforementioned rules (Consistency Pays), this particular problem was very difficult to detect. Correlating wind speed data among all stations would not have helped, since all of them were affected by this bug. With a remote reprogramming system, we could have avoided going back to the mountain.

Guard Your Treasure

Do not throw away even a single byte of raw data.

On the back-end server, there will be multiple programs processing data and producing statistics, aimed at visualizing certain aspects of the system. Nevertheless, all the raw data must be securely archived for future reference. There may—and will—be some statistics that were not envisioned at first. One may also, for example, discover that the equation used to transform raw data into SI units was wrong, or poorly implemented. If the original data is no longer available, and the conversion destructive, the obtained data may be worthless after all.

In SensorScope, the server creates log files, which we call *raw dumps*, besides generating the real-time statistics. These logs contain all packets (in their raw hexadecimal format) coming in from the network, together with a timestamp. Once a day, a new file is opened and the previous one is closed and compressed. This way, we have archived approximately 20 MB of compressed raw dumps (100 MB uncompressed) for the Génépi deployment alone. As these files are the most precious output from a deployment, they must be carefully backed-up. Everything else can be recovered from them. Figure 8, showing time-correlated losses during the Grand St. Bernard deployment, is an example of a plot that has been generated *a posteriori*, directly from these dumps.

Seeing Is Believing

You must make sure to be able to correctly make sense of the data you gather.

As extensively noted above, gathering data is a problem in itself, but making sense of it, is a whole nother one. Sensors provide only a partial view of the real world, which may be insufficient to correctly interpret their readings. For instance, when ambient light measurements are low, is it because of clouds or because a leaf has fallen onto the sensor? Figure 5b shows that there has been a good amount of precipitation on Oct 3 during the Grand St. Bernard deployment, but was it rain or melting snow, which had fallen during previous days? Our set of sensors provides a lot of useful information, but leaves us unable to make this seemingly simple distinction.

To better understand the data we gather, we are considering the possibility of equipping one or more stations with a camera to provide visual feedback. We have already begun

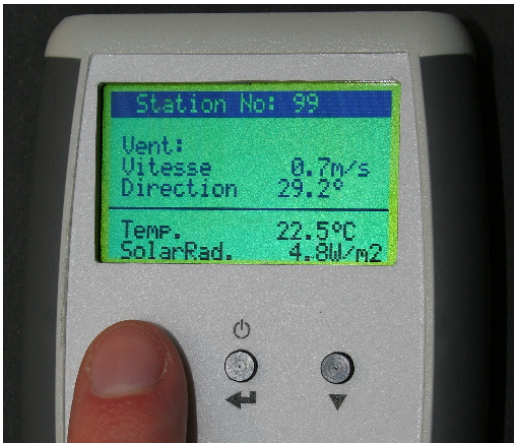


Figure 11: Our deployment-time WSN validation device.

Figure 12: Outcome of a close-by construction site.

to experiment in this domain, by installing an autonomous camera during the Génépi deployment. Figure 10 provides a sample image taken by that camera, showing a foggy and snowy day, with a thin layer of snow on the rocks. We are still a long way, however, from seamlessly integrating one or more cameras into our sensor network. A lot of problems remain unsolved, mainly regarding energy management and bandwidth usage. Our camera prototype was connected to a car battery and had its own, dedicated GPRS connection. On-site image analysis may be part of a possible solution.

Trust Is Good, Control Is Better

Perform as many on-site checks as possible.

Monitoring the status of a WSN system, while deploying it, is very important, as one certainly wants to avoid returning to the laboratory just to discover that half of the network is not functioning. Since a computer and an Internet connection may not be available at the deployment site, having a handy, battery-powered tool for deployment-time validation is highly desirable (see, for instance, TASK [2] and LUSTER [14]).

We have developed a similar device for our system. Figure 11 shows the device, as it displays a few measurements (wind speed and direction, air temperature, and solar radiation) sent by one of our stations. It has a size of 7×11 cm, and is powered by two AA batteries, making it autonomous for over one hundred hours. The system works by overhearing, *i.e.*, it captures packets sent by stations and extracts and displays their data. It can also display the various kinds of status packets, still with the aim of discovering problems as early as possible.

There Is Life after the Deployment

Keep your equipment ready for subsequent deployments.

The end of a deployment is the right time to relax before further work, namely data analysis and publication. As stated above, much work is left in these areas once the system is back in the laboratory. At the same time, however, one should not forget that more deployments are likely to be planned for the future, and that the equipment will almost certainly be used again. Starting that next deployment with fully charged batteries, certainly falls into this category.

After the Génépi deployment, we took down all stations and put the sensing boards into cardboard boxes, removing neither the motes, nor the batteries. One day, during testing, we noticed "ghost" motes in our network, and after a bit of investigation, we discovered that the motes we had brought back from the deployment were still active. They were indeed communicating with each other in their cardboard boxes, trying to send their data to a non-existing sink. Finally, they found our testbed. Of course, we had never thought about this "after-deployment" behavior. The fact that the motes had almost completely drained their batteries, no longer charged by solar panels, could have delayed our next deployment for a somewhat ridiculous reason, had we not discovered it in time. Fully charging a battery takes almost five hours, and we only have two chargers.

Flag Them All

Detect and remove outliers from your gathered data.

Having a watchdog at the server helps in detecting problems such as broken sensors, but flagging outliers often necessitates powerful offline algorithms, processing all data at once. Outliers may appear due to various reasons, some of them being quite unexpected. For instance, during one of our smaller deployments, we detected anomalous measures from a station, uncorrelated with the readings from the others. We went back to the site to inspect it, and Figure 12 shows what we found: a close-by building was under construction, and workers, not knowing what our station was, simply wrapped it in plastic to protect it. While this particular example is funny, it illustrates how easy it is to collect incorrect data with correct hardware and software.

In SensorScope, flagging outliers is currently mostly done by hand at EFLUM, which is a very time consuming task. Developing validity checks and QA/QC algorithms is complicated, since it requires a perfect understanding of the monitored parameters, and is still work in progress. However, this represents an important part in deploying WSNs, as the considered sensors are generally not of very high quality, and may often lead to such outliers.

Trace Your Steps

Do not forget that traceability is a key point in managing deployments.

As the software on both the server and the motes is likely to evolve over time, it is important to know exactly which version was used for a given deployment. Here, a version

Table 3: Our most important advice.

Dos	*Don'ts*
Add remote monitoring and control mechanisms	Make your system a black box
Always think about your upcoming publications	Make your system more complex than needed
Package your sensors well, but unobtrusively	Mistake your laboratory or testbed for the real world
Be consistent between tests and deployments	Implicitly trust your gathered data
Diligently track and flag your data	Throw away even a single byte of gathered data

control system, such as Subversion[10], is not enough. A good solution, which, unfortunately, we did not have at first, is to tag data packets with a version byte. As mentioned above, due to the wind speed bug during the Génépi deployment, we had to go back to the site to reprogram the motes. Now, the raw dumps coming from this deployment contain packets generated by two different versions of the code. The obvious problem is to distinguish between packets containing correct and incorrect wind speed measurements. Moreover, the order of the data fields inside the packets had changed. Since the packets in their hexadecimal raw format show no indication of this, we had difficulties to automatically extract meaningful data from them. If we had tagged the packets with a simple version-control byte from the beginning, the reordered fields would not have posed a problem.

The traceability of individual measures and devices is also very important. For instance, when a bad sensor is detected, which probably broke during a deployment (see for instance Figure 5b showing a broken rain meter), it is common practice to simply exchange it. However, as the sensors are not tightly associated with the ID of the station they are connected to (these are given by the motes placed onto the stations), it is virtually impossible to determine which values from previous deployments should be double-checked for integrity. In SensorScope, we are currently in the process of tagging all our motes and sensors with RFIDs. With the corresponding reader, it becomes very easy to scan stations during deployments to associate sensors and station IDs. Storing this information in a database will allow us to retrace the exact history of all devices and measures, so that flagging potentially incorrect data will be made easier.

5. CONCLUSION

Our multiple field campaigns have shed light on various problems linked to the actual deployment, to early development errors, and to system testing. In this paper, we have shared the experience we have acquired in dealing with all these problems. In spite of not addressing all possible challenges, following our advice should help other groups in appreciating and anticipating many issues related to real-world WSN deployments. Table 3 summarizes this paper, by recapitulating the most critical issues.

We are now in the refinement process of SensorScope. We believe that our most important remaining issue is data inspection and assessment. Even if a system does not manage to collect every single desired datum, still, much data is usually gathered. But what is the point of gathering data at all, if it is incorrect or cannot be properly interpreted? Therefore, in interdisciplinary collaboration, we are investigating the real-time detection of outliers to ensure data quality. Lastly, our camera project presents many challenging problems, but will also be of great benefit to data analysis once fully integrated.

[10] `http://subversion.tigris.org`

6. ACKNOWLEDGMENTS

This work was partially financed by the Swiss National Center of Competence in Research for Mobile Information and Communication Systems (NCCR MICS) and the European Commission under the FP6 project WASP. We would like to thank Samuel Madden, our shepherd for this paper, for helping us to improve it.

7. REFERENCES

[1] G. Barrenetxea, F. Ingelrest, G. Schaefer, M. Vetterli, O. Couach, and M. Parlange. Sensorscope: Out-of-the-box environmental monitoring. In *Proceedings of the ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, Apr. 2008.

[2] P. Buonadonna, D. Gay, J. M. Hellerstein, W. Hong, and S. Madden. TASK: Sensor network in a box. In *Proceedings of the IEEE European Workshop on Wireless Sensor Networks and Applications (EWSN)*, Jan. 2005.

[3] G. Chelius, A. Fraboulet, and E. Fleury. Worldsens: Development and prototyping tools for application specific wireless sensors networks. In *Proceedings of the ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, Apr. 2007.

[4] H. Dubois-Ferrière, R. Meier, L. Fabre, and P. Metrailler. Tinynode: A comprehensive platform for wireless sensor network applications. In *Proceedings of the ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, Apr. 2006.

[5] S. Ganeriwal, R. Kumar, and M. Srivastava. Timing-sync protocol for sensor networks. In *Proceedings of the ACM International Conference on Embedded Networked Sensor Systems (SenSys)*, Nov. 2003.

[6] J. Hui and D. Culler. The dynamic behavior of a data dissemination protocol for network programming at a scale. In *Proceedings of the ACM International Conference on Embedded Networked Sensor Systems (SenSys)*, Nov. 2004.

[7] K. Langendoen, A. Baggio, and O. Visser. Murphy loves potatoes: Experiences from a pilot sensor network deployment in precision agriculture. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Apr. 2006.

[8] P. Levis, N. Lee, M. Welsh, and D. Culler. TOSSIM: Accurate and scalable simulation of entire TinyOS applications. In *Proceedings of the ACM International Conference on Embedded Networked Sensor Systems (SenSys)*, Nov. 2003.

[9] P. Levis, S. Madden, D. Gay, J. Polastre, R. Szewczyk, K. Whitehouse, J. Hill, M. Welsh, E. Brewer, D. Culler, and A. Woo. *Ambient Intelligence*, chapter TinyOS: An Operating System for Sensor Networks. Springer, 2005.

[10] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of the ACM International Workshop on Wireless Sensor Networks and Applications*, Sept. 2002.

[11] J. Polastre, J. Hill, and D. Culler. Versatile low power media access for wireless sensor networks. In *Proceedings of the ACM International Conference on Embedded Networked Sensor Systems (SenSys)*, Nov. 2004.

[12] K. Römer, P. Blum, and L. Meier. *Handbook of Sensor Networks: Algorithms and Architectures*, chapter Time Synchronization and Calibration in Wireless Sensor Networks. John Wiley and Sons, 2005.

[13] T. Schmid, O. Sekkat, and M. Srivastava. An experimental study of network performance impact of increased latency in software defined radios. In *Proceedings of the ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization (WINTECH)*, Sept. 2007.

[14] 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 *Proceedings of the ACM International Conference on Embedded Networked Sensor Systems (SenSys)*, Nov. 2007.

[15] R. Szewcszyk, A. Mainwaring, J. Polastre, J. Anderson, and D. Culler. Lessons from a sensor network expedition. In *Proceedings of the IEEE European Workshop on Wireless Sensor Networks and Applications (EWSN)*, Jan. 2004.

[16] B. Titzer, D. Lee, and J. Palsberg. Avrora: Scalable sensor network simulation with precise timing. In *Proceedings of the ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, Apr. 2005.

[17] 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 *Proceedings of the ACM International Conference on Embedded Networked Sensor Systems (SenSys)*, Nov. 2005.

[18] 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.

[19] A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *Proceedings of the ACM International Conference on Embedded Networked Sensor Systems (SenSys)*, Nov. 2003.