SensorScope: Application-Specific Sensor Network for Environmental Monitoring

FRANÇOIS INGELREST, GUILLERMO BARRENETXEA, GUNNAR SCHAEFER, MARTIN VETTERLI, OLIVIER COUACH, and MARC PARLANGE
EPFL—WSL/SLF, Switzerland

SensorScope is a turnkey solution for environmental monitoring systems, based on a wireless sensor network and resulting from a collaboration between environmental and network researchers. Given the interest in climate change, environmental monitoring is a domain where sensor networks will have great impact by providing high resolution spatio-temporal data for long periods of time. SensorScope is such a system, which has already been successfully deployed multiple times in various environments (e.g., mountainous, urban). Here, we describe the overall hardware and software architectures and especially focus on the sensor network itself. We also describe one of our most prominent deployments, on top of a rock glacier in Switzerland, which resulted in the description of a micro-climate phenomenon leading to cold air release from a rock-covered glacier in a region of high alpine risks. Another focus of this paper is the description of what happened behind the scenes to turn SensorScope from a laboratory experiment into successful outdoor deployments in harsh environments. Illustrated by various examples, we point out many lessons learned while working on the project. We indicate the importance of simple code, well suited to the application, as well as the value of close interaction with end-users 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: Algorithms, Design, Experimentation, Performance

Additional Key Words and Phrases: Architecture, deployment, environmental monitoring, implementation, wireless sensor network

ACM Reference Format:

Ingelrest, F., Barrenetxea, G., Schaefer, G., Vetterli, M., Couach, O., and Parlange, M. 2010. SensorScope: Application-specific sensor network for environmental monitoring. ACM Trans. Sensor Netw. 6, 2, Article 17 (February 2010), 32 pages.
DOI = 10.1145/1689239.1689247 http://doi.acm.org/10.1145/1689239.1689247

This work was partially financed by the Swiss NCCR MICS and the European FP6 WASP project. Part of this work was published in ACM/IEEE IPSN 2008 and in ACM SenSys 2008.
Authors' addresses: F. Ingelrest, G. Barrenetxea, G. Schaefer, and M. Vetterli: LCAV, I&C School, EPFL. Station 14, EPFL, CH-1015 Lausanne, Switzerland; email: {francois.ingelrest,guillermo.barrenetxea,gunnar.schaefer,martin.vetterli}@epfl.ch; O. Couach, M. Parlange: EFLUM, ENAC School, EPFL. Station 2, EPFL, CH-1015 Lausanne, Switzerland; email: {olivier.couach,marc.parlange}@epfl.ch.

DOI 10.1145/1689239.1689247 http://doi.acm.org/10.1145/1689239.1689247

1. INTRODUCTION

Self-organization and multihop routing make wireless sensor networks (WSNs) highly versatile, favoring their use to replace older, less user friendly technologies. From this perspective, their potential contribution to various domains has been widely studied, ranging from habitat monitoring [Mainwaring et al. 2002; Wang et al. 2005] to the study of tree canopy climate [Tolle et al. 2005], to precision agriculture [Langendoen et al. 2006], and to environmental monitoring [Werner-Allen et al. 2006; Selavo et al. 2007].

In domains such as hydrology and micrometeorology, one quickly faces the challenges of spatial heterogeneity, as well as the strong diurnal changes due to solar forcing. Most models attempt to capture the distributed nature of the landscape, but the availability of spatio-temporal data as input (initial and boundary conditions) has generally been lacking. In fact, the typical situation in hydrology is that there is at best a single expensive sensing station (see Figure 1(a)) in a large watershed, so that engineers are confronted with the problem of making predictions without spatial information. An easy-to-deploy-and-configure WSN can greatly help in collecting the required data: this is where SensorScope comes into play with its flexibly networked stations (see Figure 1(b)).

Our objective is to provide a low-cost, reliable WSN-based system for environmental monitoring, to improve data collection techniques. The data rate in this domain is low (e.g., one data packet every two minutes), while existing communication stacks are generally designed for higher rates. We have thus developed our own stack featuring a multihop data gathering protocol and a synchronized duty-cycling MAC layer that helps in reducing the overall energy consumption, assuming a low data rate. Another key feature of our stack is the simple interface it presents to higher layers, abstracting network details, and allowing for great ease when writing applications. This is important, since SensorScope aims at being adopted by a community with no knowledge in networking. We also point out that our communication stack is freely available on our Website under an open-source license.[1]

We have already deployed several WSNs for typical environmental applications at various places in Switzerland, ranging from the border of a water stream to high mountains, as well as on the EPFL campus to study an urban environment. Therefore, the concept, architecture, hardware, and software we present here have had an immediate impact on real-world environmental monitoring applications. In this paper, we especially focus on how data is gathered. We hence describe both our hardware and software architectures, the motivations for the design of our communication stack, and how we implemented it. We provide results gathered during deployments, about the network, the sensing stations, and, of course, the environment. We also share the experience we have acquired and go through the lessons we learned, detailing the problems we faced (e.g., weather conditions, software bugs). Although SensorScope is aimed at outdoor deployments, most of the issues we describe are common to

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

(a) Traditional sensing station. (b) Lined up SensorScope stations.

Fig. 1. Traditional sensing stations (a) are big and expensive, and are unable to compete with a WSN (b) on numerous points (e.g., flexibility, real-time data access, energy consumption).

other kinds of deployments. We believe that our experience will be of interest to the community, and that it will help other groups in anticipating many of the problems they are likely to face.

In the next section, we present an overview of previous prominent outdoor WSN deployments. We then present the architecture of SensorScope in Section 3, describing both the hardware we have designed and the communication stack we have developed. In Section 4, we detail the key features of our network architecture, while in Section 5, we provide and discuss the results obtained on our indoor testbed as well as those of our aforementioned outdoor deployments. Section 6 is dedicated to the sharing of our experience and to our advice on the successful deployment of a WSN. We finally conclude in Section 7.

2. RELATED WORK

Deploying a WSN has always been reported as a difficult task. One of the first documented deployments was performed by a group at Berkeley in 2002, on Great Duck Island, to help habitat monitoring [Mainwaring et al. 2002; Szewczyk et al. 2004]. 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. This deployment also opened the path to research on WSNs for habitat monitoring: One such example is the work done later on by CENS at UCLA on using WSNs for woodpecker localization [Wang et al. 2005]. More generally, WSNs are often considered as a solution for many problems related to the environment.

A few years later, the same group at Berkeley reported results obtained from their new sensor network, Macroscope [Tolle et al. 2005], built on top of TASK [Buonadonna et al. 2005], a set of WSN software and tools, also designed

at Berkeley. Macroscope has been extensively used for micro-climate monitoring of a redwood tree. Despite building on their previous experience, the authors faced numerous issues, such as correctly calibrating sensors or detecting outliers.

Lessons have stacked up with the increasing quantity of reported deployments, and have resulted 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, by a group at Harvard [Werner-Allen et al. 2006].

Nevertheless, new reports about failed campaigns keep appearing, demonstrating that WSN deployments remain a nontrivial task. One of them was reported by researchers at Delft University, who deployed a large-scale sensor network in a potato field [Langendoen et al. 2006]. 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" and complex software components, not suited to the targeted application. A lack of consistency and coordination between the team members was also reported. Overall, this article remains a good advocate for carefully preparing deployments to avoid disastrous failures.

Many recent outdoor WSN deployments still target environmental research. For instance, researchers from the University of Virginia have reported their work on LUSTER, which has been designed mainly to gather light measurements [Selavo et al. 2007]. Nevertheless, new application domains for WSNs keep appearing, such as monitoring personal use of water resources [Kim et al. 2008]. There is no doubt that such new domains will bring new issues in deployment methodologies.

Many of our decisions during the development of SensorScope were based on all this past experience (e.g., packaging of sensors [Szewczyk et al. 2004], taking care of time drift [Werner-Allen et al. 2006]). A key decision was to follow TASK's principle by promoting the *keep it simple* philosophy [Buonadonna et al. 2005], to maximize our chances of conducting successful deployments. As we show in the following, this resulted in a working system, leading to new and original field observations. We believe that the strength of our contribution lies in the global view of the deployment process we provide, from the complete design of the system to the final analysis of the results by the end-user community. We also learned many lessons, that we share with the WSN community, providing an interesting and different view of the deployment process.

3. SENSORSCOPE OVERVIEW

Traditionally, environmental monitoring is done using only a small number of very expensive sensing stations ($\pm$ €60,000 is a common price), thus restricting spatial coverage (see Figure 1(a)). Furthermore, these stations generally use data loggers: this storing technique not only suffers from limited capacity, but also prevents users from obtaining immediate feedback, since it requires

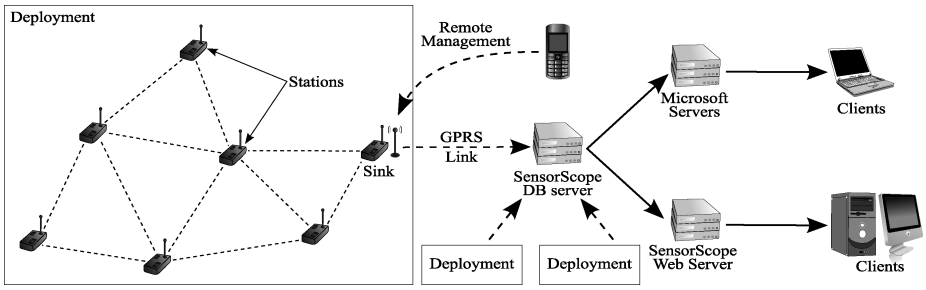


Fig. 2. Overall architecture. Most deployments feature a GPRS-enabled sink that sends gathered data to our database server, which in turn makes it available to other servers. Remote management of the sink is possible via GSM text messages.

manual downloading of the data from each station. Thus, WSNs are highly relevant to this research domain, as they allow for both real-time (e.g., storms) and long-term (e.g., ice melting) monitoring of natural events in areas of varying size.

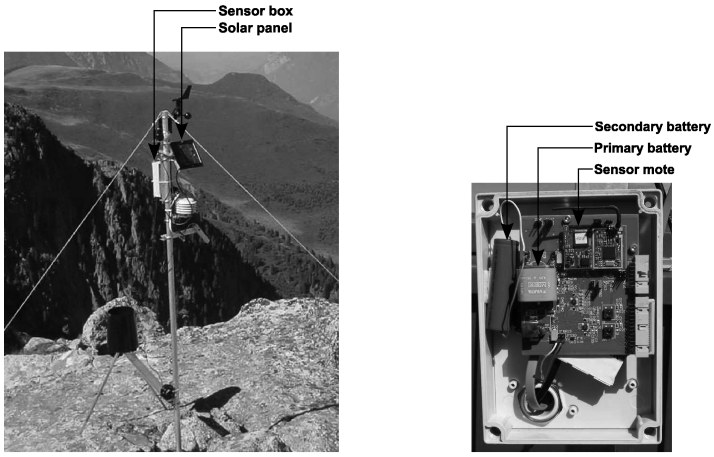
SensorScope is such an environmental monitoring system based on a time-driven WSN, developed in collaboration between two laboratories at EPFL: LCAV (signal processing and networking) and EFLUM (hydrology and environmental fluid mechanics). Our aim is to help environmental engineers to address long term monitoring questions in challenging environments. Our stations regularly transmit data (e.g., wind speed and direction) to a sink, which in turn uses a gateway to relay the data to our server. Depending on the deployment scenario and the available communication resources, we use different kinds of gateways (e.g., GPRS, Ethernet). Data is published on our real-time Google Maps–based Web interface[2] and on Microsoft's *SensorMap* Website.[3] Figure 2 illustrates this architecture. Because our objective is to replace the aforementioned traditional expensive stations, the baseline requirements that we have followed are *low cost* and *full autonomy*, while maintaining sufficient accuracy. In this section, we describe the design of our hardware and software architecture to address these requirements.

3.1 Hardware Design

When the project was started, there were no sensing stations that could be used off-the-shelf. Therefore, we had to design and build suitable ones.

3.1.1 *Sensor Mote.* We chose a Shockfish TinyNode.[4] It is composed of a Texas Instruments MSP430 16-bit microcontroller, running at 8 MHz, and a Semtech XE1205 radio transceiver, operating in the 868 MHz band, with a transmission rate of 76 Kbps. The mote has 48 KB ROM, 10 KB RAM, and 512 KB flash memory. We opted for this platform mainly for the good ratio

[2]http://www.climaps.com
[3]http://atom.research.microsoft.com/sensormap/
[4]http://www.tinynode.com



(a) A sensing station. (b) A sensor box.

Fig. 3. Design of a sensing station.

it offers between communication range and power consumption (up to 1200 meters outdoors for 60 mA) [Dubois-Ferrière et al. 2006].

3.1.2 *Stations.* A sensing station, depicted in Figure 3(a), is composed of a two-meter-high flagstaff, to which the solar panel and the sensors are fixed. The electronic circuitry is placed inside a hermetic box (see Figure 3(b)), which is also attached to the pole. The average price of a station is around €1,500. A key goal of the project is to obtain dense spatial measures. We achieve this by deploying multiple low-cost—possibly less accurate—sensing stations, rather than a single expensive, but very accurate one.

3.1.3 *Power Source.* In the spirit of Heliomote [Raghunathan et al. 2005], we have designed a solar energy system to achieve autonomy during deployments. It is composed of three modules:

(1) *Solar panel*. A 162 × 140 mm MSX-01F polycrystalline module that provides a nominal power output of 1 W in direct sunlight, with an expected lifetime of 20 years. Our power control driver is similar to Prometheus [Jiang et al. 2005]: two batteries are alternately used.
(2) *Primary battery*. A 150 mAh NiMH rechargeable battery (see Figure 3(b)). We chose a NiMH battery over a supercapacitor due to its higher capacity and its lower price. It allows for up to five days of solar blackout (considering a networking *duty-cycle* of 10%).
(3) *Secondary battery*. A Li-Ion battery with a capacity of 2 200 mAh. It is the cylinder-shaped battery located on the left in Figure 3(b). This buffer is used as a backup source of energy during long periods of low solar radiation. It is charged via the primary battery, thus undergoing fewer charging cycles.

Table I. Environmental Quantities Measured by Our Stations

Measure	Sensor	Range	Precision
Air humidity	Sensirion SHT75	0–100 %	± 2 %
Air temperature	Sensirion SHT75	-20–60°C	± 0.3°C
Precipitation	Davis Rain Collector	0–∞ mm	± 1 mm
Soil moisture	Decagon EC-5	0–100 %	± 0.1%
Solar radiation	Davis Solar Radiation	0–1800 W/m^2	± 90 W/m^2
Surface temperature	Zytemp TN901	-33–220°C	± 0.6°C
Water content	Irrometer Watermark	-200–0 kPa	unknown
Wind direction	Davis Anemometer	0–360°	± 7°
Wind speed	Davis Anemometer	1.5–79 m/s	± 1.5 m/s

This system, in conjunction with the power conserving algorithms implemented at the network level, theoretically makes the batteries' recharge cycle-count the only limiting factor for long-term deployments. Considering how our batteries are used, we currently expect them to last around three years.

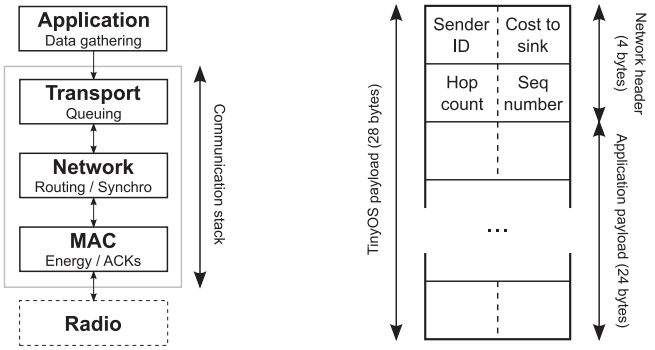
3.1.4 *Sensing Modalities.* The stations can accommodate up to seven different external sensors, some of them measuring multiple quantities, as shown in Table I. With our choice of sensors, the stations can measure up to nine environmental quantities required for environmental monitoring campaigns, in particular for hydrology. To ensure the quality of the measurements, we test sensors before deployment by comparing their readings to reference sensors over several days.

3.2 Network Design

The first outdoor deployment of SensorScope was in July 2006 on EPFL's campus, and it mainly aimed at validating the hardware design of the stations. Accordingly, the embedded software was simple, and was not built on top of a communication stack. This implied multiple limitations, especially in terms of range and reliability.

Gathering data in difficult-to-access places requires a robust system, and we have found the assertion of the TASK authors to be true [Buonadonna et al. 2005]: simple and application-specific approaches provide the most robust solutions for real-world use. Moreover, gluing existing components takes a lot of time and effort for an in-depth understanding of their interactions. Since our data rate is low, and most existing communication stacks are not specifically suited to this aspect, we chose to design and implement from scratch our own stack for TinyOS [Levis et al. 2005]. Figure 4(a) shows its architecture, which is inspired by the OSI model [Zimmermann 1980]. The arrows between the layers indicate that no data is forwarded to the application. The multi-hop mechanism is indeed automatically managed, and there is no need for the application to care about packets. This may change in the future, for instance, if *in-network processing* is considered.

Our stack stores only four bytes of information per packet. We chose to put them into the payload, leaving 24 bytes for the application layer out of the 28 available in TinyOS, as illustrated in Figure 4(b). We could have added our own header to the standard network header, to leave the TinyOS payload



(a) The stack. (b) Packet format.

Fig. 4. SensorScope communication stack and packet format.

unchanged, but this would have implied to maintain the code after each new release of the radio drivers. Moreover, these files are radio-specific. By storing these bytes in the TinyOS payload, our stack is independent of the underlying radio drivers. In the following, we describe the different layers.

3.2.1 *Application Layer.* It collects the data to be sent to the sink. In SensorScope, it periodically queries both the sensors and the batteries, whose readings are used to monitor the energy level of the stations at the server.

3.2.2 *Transport Layer.* It provides a very simple interface to end users, composed of only two commands for sending data, that completely abstracts networking details. Each of them creates a different kind of packet:

(1) *Data packets*. They contain data to be routed to the sink, examples of such data being the sensors' or the batteries' readings.
(2) *Control packets*. They are intended for a specific neighbor of the node or for all of them—in case of a *local* broadcast—and they are thus not forwarded once received. Examples of such messages are beacons or synchronization packets. There are no network-wide control packets to keep the network decentralized, and also because they did not prove to be needed in our design.

This layer is responsible for creating packets out of the data received via the two aforementioned commands, and for storing them in the corresponding queue. Whenever the queues are not empty, this layer tries to send the next packet, by passing it to the network layer. Priority is given to control packets, that is, if there are both data and control packets waiting, then the latter are sent first. We made this choice because control packets are important for the network operation, and thus have higher timeliness requirements than data messages. Since we assume the overall network traffic to be low, there are no congestion avoidance mechanisms.

The transport layer fills two fields in the header (see Figure 4(b)). The first one is the hop count, which is set to zero for new packets, and incremented for

other ones. This information is not mandatory and is used only for statistical purposes. The second field is the sequence number, filled with an internal data or control counter. These counters are incremented only when messages are correctly sent: if the sending fails (e.g., no acknowledgment), the packet is resent with the same sequence number. This field is used for link quality evaluation (see Section 4.1).

3.2.3 *Network Layer*. It decides whether packets should be routed to the sink based on their type, and if so, how this should be done. At the sink, it forwards data packets to the serial port, while control packets are passed to the MAC layer. At the motes, it passes both kinds of packets to the MAC layer. Since control messages already have a recipient, no further action is required. For data packets, this layer first chooses a next hop toward the sink. How this is done is protocol-specific and is detailed in the next section. Implementing a new routing protocol simply requires a new network layer, with the rest of the stack unchanged.

The network layer fills the two remaining header fields: the sender identifier and the cost to the sink. How this information is obtained is detailed in the next section.

3.2.4 *MAC Layer*. It manages the radio, namely switching it on/off and sending/receiving messages. In case of a data message, an acknowledgment (ACK) is sent back to the sender.

When we prepared for our deployments, the radio drivers of the TinyNode were still lacking a *carrier sense*, and we could thus not add a busy-channel detection. Therefore, we used a backoff mechanism, whose maximum delay is exponentially increased each time a data packet is not acknowledged. Upon a successful transmission, the maximum delay reverts to the minimum value. Upon a missing ACK, the failure is signaled to the network layer with the appropriate flag. Since our goal is not to control the throughput of nodes (as TCP does) but the amount of collisions, this simple mechanism is sufficient for our purpose. Moreover, it does not lead to an increased energy consumption as the duration during which the radio is turned on is independent of this mechanism. More on this is elaborated in Section 4.3.

4. NETWORKING FEATURES

In this section, we describe the key features of our communication stack that make the system auto-organized and energy-efficient, and how they are currently implemented. In the following, *broadcast* designates a local broadcast (i.e., a packet sent to all neighbors), not a network-wide one. The distance always designates the *hop-distance* to the sink, not the Euclidean distance.

4.1 Neighborhood Management

Nodes manage a *neighborhood table* in which they store the nodes they can hear from (literally, their *neighbors*). We chose to let nodes discover their neighborhood by overhearing their neighbors' packets, in the spirit of MintRoute [Woo et al. 2003]. Only the sink sends beacons to initiate the process. Each time a

node updates its table, it also updates its cost to the sink. We currently use the hop-distance to the sink as the cost metric. Because neighborhood information is mainly needed for routing data to the sink, the table is managed by the network layer. To account for *dead neighbors* (e.g., hardware failure), a timer is used to remove old entries.

The acknowledgment mechanism at the MAC layer could be used to detect asymmetric neighbors (i.e., neighbors that can only be heard). However, currently, when an ACK is not received, the message is simply sent again (potentially to the same next hop). A possible improvement could be to manage a *blacklist* of asymmetric neighbors, to avoid using them as next hops.

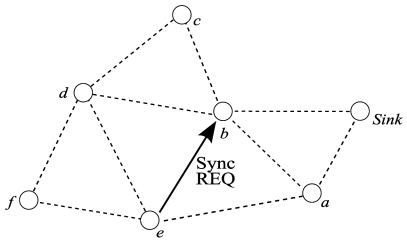
Due to the randomness of the radio channel, the nodes need to estimate the quality of links (QoS), so that poor-quality neighbors may be considered separately. To achieve this, the neighborhood table stores the sequence numbers of the latest 16 packets received, and the QoS is estimated by counting missing numbers (denoted x): the quality is then equal to $16/(16 + x)$, which varies with the quantity of missing sequence numbers. When less than 16 messages were received from a given neighbor, we set its quality to zero. An alternative solution could have been to use the *received signal strength indicator* (RSSI), but we found this method not to be precise enough. The RSSI is indeed influenced by a lot of parameters (e.g., antenna matching, location of nodes), and the measured value for a given neighbor may greatly vary at each reception.

Since the overall goal is to route data to the sink, the QoS of a neighbor must reflect its capacity to forward messages to the sink. A problematic situation may be caused, for instance, by a very good neighbor, in terms of packet delivery, with a poor capacity to route messages to the sink. To avoid this, only the sequence numbers of data messages are used to estimate the QoS. Indeed, when a neighbor is unable to successfully send such a message to a next hop, it resends it with the same sequence number, thus decreasing the QoS of that neighbor (a duplicated sequence number is counted just like a missing one). This mechanism ensures that the quality of a neighbor is based on how well it can be heard, as well as how good it is at "communicating" with the sink.

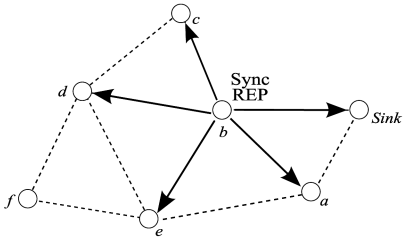
4.2 Synchronization

Connectivity problems may block packets at a node for some time, resulting in routing delays. This rules out that the server time-stamps the data, and implies that nodes must put a timestamp in their reports to allow for meaningful interpretation of the gathered data. As our power management approach, detailed in the next section, relies on synchronous duty-cycling, we chose a global synchronization mechanism.

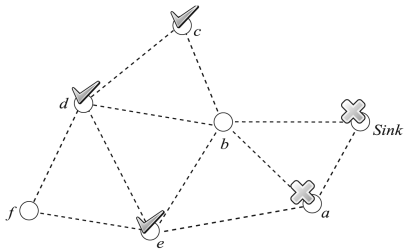
4.2.1 *At the Nodes.* We use `Sync REQ`/`Sync REP` messages, as illustrated in Figure 5, the goal being to propagate the current time into the network from the sink. When a node wants to update its clock, it sends a request to a neighbor, *closer than itself* to the sink. If it knows the current time, that neighbor then broadcasts a reply with this value. Upon reception of such a reply, all nodes



(a) Request from e to b.



(b) Reply broadcasted by b.



(c) Nodes c, d, and e are further from the sink than b, and update their clock.

Fig. 5. Synchronization amongst nodes.

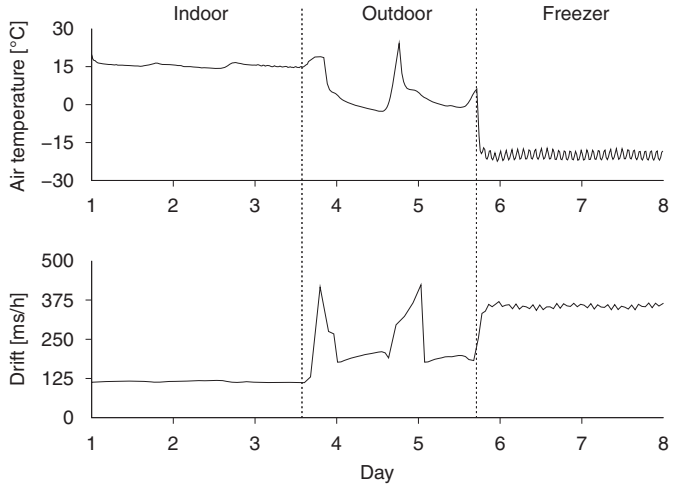


Fig. 6. Time drift per hour of a TinyNode in three different environments.

further than the sender from the sink update their clock. This synchronization mechanism is managed by the network layer, but time-stamping is performed by the MAC layer to eliminate delay errors [Ganeriwal et al. 2003].

To compensate *time drift*, we use two update modes: a *high-frequency mode*, used when nodes do not have the current time (e.g., after boot), and a *low-frequency mode* for later updates. Figure 6 shows the impact of temperature on the crystal used in TinyNodes. This experiment was conducted in three phases,

during which we placed the mote in different environments: indoors, outdoors, and inside a freezer. The drift was computed by subtracting the mote's local time from a global reference time. We can see that the cooler the temperature, the slower the crystal oscillates. Indoors, the drift is close to the datasheet value (±120 ms/h) while inside the freezer, the drift is much higher (±375 ms/h). By looking at the outdoors part, it is clear that day/night cycles can be particularly challenging.

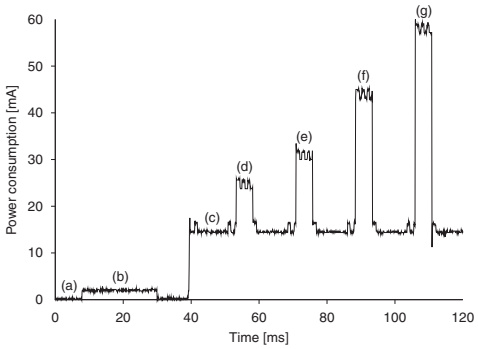
Based on these results, we consider that an update period of one hour (average drift between 120 and 375 ms) for the low-frequency mode is sufficient. The choice of the high frequency depends on the duty-cycling mechanism, and is explained in the next subsection. Although high-accuracy solutions exist, such as FTSP [Maróti et al. 2004], our approach is simple and provides sufficient precision for both time-stamping and duty-cycling. Moreover, high-accuracy solutions compensate time drift with linear regression, while the drift varies with weather conditions, making it difficult to completely avoid synchronization errors. If the application allows it, it is actually better to live with a slight drift, rather than trying to eliminate it.

4.2.2 *At the Sink.* We decided at first to regularly send the actual time from the server to the sink's mote, but we found this method to be problematic: when using a GPRS gateway, it is difficult to send data from the server to the sink's mote. We thus chose to use the local time of the sink as the *network time*, and to translate timestamps at the server. To achieve this, the sink regularly sends a message with its local time to the server, which in turn computes the offset between the network time and the actual time. To account for accidental reboots of the sink, it first tries to synchronize with other nodes, by broadcasting requests, using the high-frequency mode. In case of no reply (i.e., the network has just been started), it starts using its local time as the network time, which then propagates. This mechanism could be a problem if the base station rebooted at the same time as all 1-hop stations, since this could lead to a disconnected subnetwork. In our next-generation system, we expect to use the actual time (retrieved for instance using a GPS chip) as the network time to avoid such problems.

4.3 Power Management

Although our solar energy system is efficient, the mote's radio is a big energy consumer: keeping it on all the time leads to a negative energy balance. Figure 7 illustrates this consumption. Using an oscilloscope, we measured the power consumption of a TinyNode mote according to its activity. We can see that simply turning on the radio multiplies the consumption by more than seven. This means that keeping it off as long as possible, rather than listening constantly, greatly reduces energy consumption.

Nodes must thus organize themselves into two-state communication cycles: an *active state*, during which the radio is on for sending/receiving messages, and an *idle state*, during which the radio is off. Achieving good savings, of course, requires the idle state to be as long as possible. Two approaches exist:



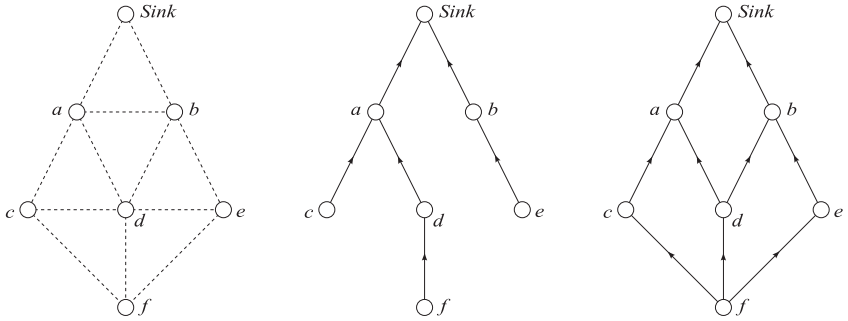
Label	Activity	Value
(a)	None	0 mA
(b)	CPU on	2 mA
(c)	Radio on	15 mA
(d)	Transmission (0 dBm)	25 mA
(e)	Transmission (5 dBm)	31 mA
(f)	Transmission (10 dBm)	44 mA
(g)	Transmission (15 dBm)	58 mA

Fig. 7. Power consumption of a TinyNode mote according to its activity.

(1) *Low-power listening* (LPL). This solution is asynchronous: nodes do not have to wake up at the same time to communicate. To achieve this, a *preamble* (i.e., a specific pattern of bits) is sent before the packet itself. If its length is longer than the idle state, all neighbors are ensured to detect it during their upcoming active state, and to wait for the incoming packet. B-MAC is a well-known MAC layer that uses this mechanism [Polastre et al. 2004].
(2) *Duty-cycling*. In contrast, this solution requires all nodes to synchronously switch their radio on. Because they are all active at the same time, there is no need for preambles and packets can be sent as usual, resulting in slightly better savings upon transmissions. TASK makes use of duty-cycling to conserve energy [Buonadonna et al. 2005].

We opted for the duty-cycling method because we found this solution to be more interesting than LPL. The latter indeed requires the preamble to be longer than the idle state, and since good energy savings require this state to be long, transmissions can themselves get very long, potentially resulting in congestion. It has also been shown that LPL may actually lead to a higher energy consumption [Buonadonna et al. 2005]. Moreover, waking up nodes at the same time is easily done, thanks to the synchronization mechanism previously described, which is precise enough for this purpose. To take care of startup, when nodes do not have the network time, they keep their radio on until being synchronized, for which the high frequency, mentioned in the previous section, must be chosen carefully. To ensure that a request will be received by a neighbor during its upcoming communication cycle, the delay used in this mode must be smaller than the length of the active state. To account for time drift, and given that during our tests the maximum drift we measured was 375 ms/h (see Section 4.2), a node first waits for 500 milliseconds, without sending messages at the beginning of its active state, to ensure that its neighbors are indeed awake.

This mechanism is managed by the MAC layer, and is transparent to the other ones. When a message must be sent while the node's radio is off, it is kept and sent only during the upcoming active state. Since upper layers wait for the TinyOS *sendDone* signal, the actual waiting time does not matter.



(a) The topology. (b) A possible backbone. (c) All routes from f.

Fig. 8. Using a static data gathering tree greatly reduces the possibilities of reaching the sink.

4.4 Routing

To route data messages, a possible solution is to maintain a backbone on the connectivity graph (see Figure 8(a)), generally a tree rooted at the sink, such as the one illustrated in Figure 8(b). This implies a maintenance cost to detect broken links, but also quite an effort to balance the load between all possible routes. Indeed, without any further provision, one could imagine a situation where all 2-hop nodes would use the same 1-hop node as their next hop. This node would then become a bottleneck, and would spend most of its energy forwarding not only the messages of 2-hop nodes, but also the messages of 3-hop nodes, 4-hop nodes, and so forth. This situation may happen when nodes are linked to their best parent (with regards to the considered metric), such as in MintRoute [Woo et al. 2003]. As illustrated by Figure 8(c), if we assume that the packets should get closer to the sink at each hop, there are actually four different routes to go from f to the sink.

To avoid these issues, we decided to let nodes choose their next hop at random each time they have to forward a packet, resulting in a simplified form of *anypath routing* [Dubois-Ferrière 2006; Schaefer et al. 2009], in which next-hop decisions are made after sending the packets. In WSNs, it is indeed not important to take care of which route is used to reach the sink, provided that it eventually gets all data messages. Figure 8 clearly illustrates that philosophy: using a backbone such as the one in Figure 8(b) constrains node f to use d as the next hop for all its messages, while there is no reason to use neither node c nor node e. Moreover, node a has to support three nodes (c, d and f), while node b supports only e, resulting in poor load balancing. Assuming equal link qualities, using a different next hop each time results in automatic load balancing, since each neighbor is used in the best possible way according to the underlying topology. This simple method does not however ensure that neighbors with more children are chosen less frequently than others. A solution could be to let nodes broadcast their workload in order to integrate this parameter in next-hop selection.

While selecting next hops at random inherently provides good load balancing, it is however of interest to favor good neighbors. To achieve this, we defined

Table II. System Parameters Used During Deployments

Layer	Parameter	Value
Application	Sampling frequency	120 seconds
	High-quality links	$0.9 \leq$ QoS ≤ 1.0
	Low-quality links	$0.7 \leq$ QoS < 0.9
Network	Neighbor timeout	480 seconds
	High synchronization interval	5 seconds
	Low synchronization interval	1 hour
MAC	Active/idle state	12/108 seconds (10%)

two thresholds: all neighbors with a QoS above the first one are considered as *high-quality neighbors*, while other ones above the second threshold are *low-quality neighbors*. When a node must forward a message, it chooses a high-quality neighbor at random. If none exist, it randomly picks a low-quality one. Neighbors under the low-quality threshold are not considered. Since the QoS is computed in a purely point-to-point fashion, this protocol can lead to not using the best overall paths to the sink, in terms of QoS. We nevertheless decided to stick to this method to favor simplicity, instead of maintaining global information at each node.

4.5 Conclusion

We chose to develop our own communication stack to make SensorScope well suited to environmental monitoring. With regards to this goal, the two major features are:

(1) *No routing backbone*. This allows to get rid of maintenance messages, resulting in a lightweight protocol, and the random selection of next hops makes the whole protocol robust to environmental changes.
(2) *Relaxed synchronization*. This is made possible by the application itself, which does not require high-precision time stamping, and by the global duty-cycling approach, resulting once again in a robust solution.

As we show in the next section, these two main features make it possible to work in harsh environmental conditions, while providing the expected results.

5. EXPERIMENTAL RESULTS

We now provide various experimental results gathered during our indoor experiments and during one of our outdoor campaigns. Table II shows the parameters we used in both cases.

5.1 Indoor Experiment

We first tested our communication stack on our indoor testbed, composed of TinyNode motes deployed in our office building. These motes are not wired to any external sensors since this testbed is used only to test the network code. For the test run presented here, we deployed our code on 17 motes and let it run for one full week. Figure 9 provides a map of them (node 29 being one floor below the other ones). The sink is symbolized by the big circle at the bottom of the

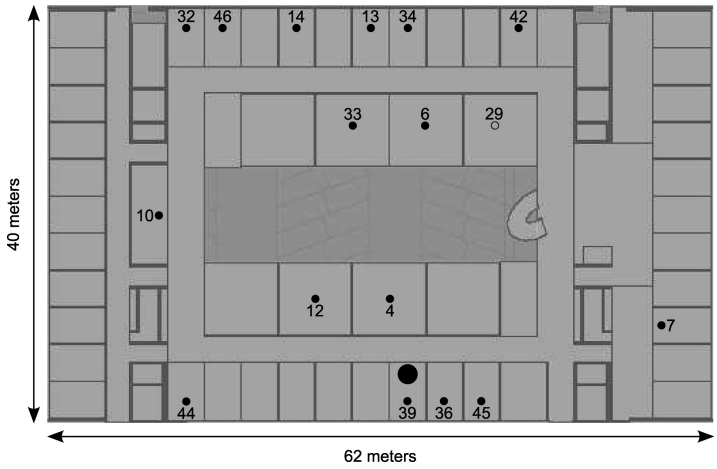


Fig. 9. The map of our indoor experiment. The sink is symbolized by the big circle at the bottom of the map. Node 29 is one floor below the other ones.

map. The center part of the building is an atrium, letting nodes communicate through it. At that point, we did not care about measuring energy consumption since some external sensors consume quite a bit, and they are not present in the testbed.

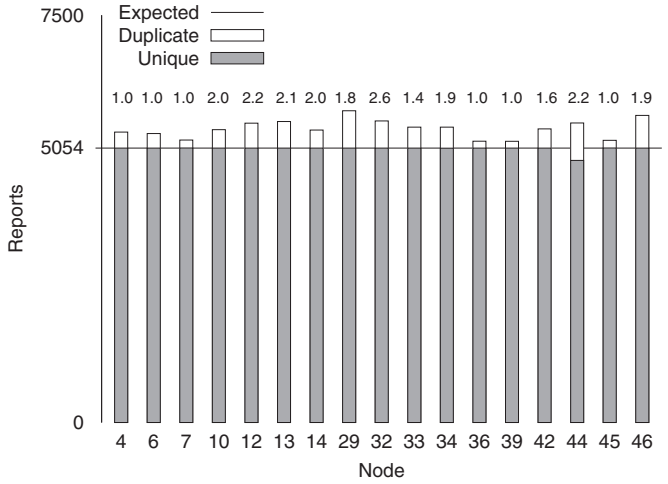
Figure 10(a) shows how many reports were received, the numbers above the bars giving the average hop count for the whole run. Thanks to the acknowledgment mechanism, we received all reports, except from node 44. It seems that this one got disconnected for some time, and its data message queue overflowed, resulting in a gross loss of 200 reports. Node 12 is spatially near to the sink, but for shielding reasons, it cannot communicate directly with it.

Duplicate packets were kept at an acceptable level of 6.5% during the run. These duplicates cannot be easily filtered out of the network because a random next hop is chosen each time, being a retransmission or not.

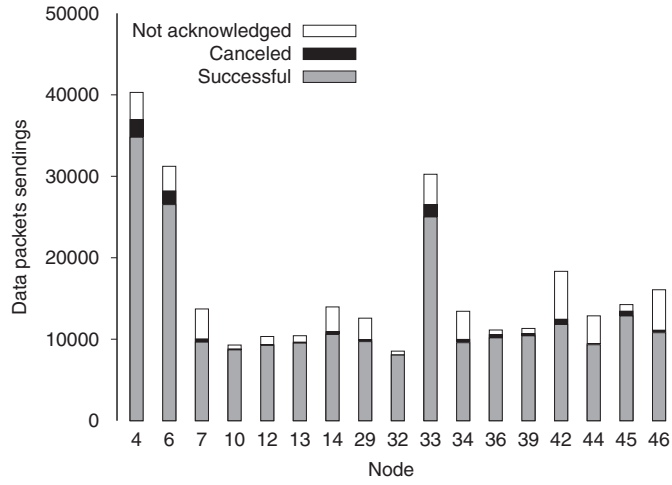
Figure 10(b) shows the quantity of data packets sent by each node, including all packets forwarded because of the multihopping mechanism. All kinds of data packets are included in this figure (e.g., reports, network statistics). Not surprisingly, nodes 4, 6, and 33 were the ones with the highest number of sent data packets, due to their central location. Canceled sendings generally occur when a data packet is received and an ACK has to be sent with high priority. Then, the current backoff (if any), is canceled to immediately send the ACK, while the data packet is kept in its queue and resent later on. A good example is node 4: because it was heavily used as a relay, a lot of its sendings had to be canceled.

5.2 Outdoor Deployments

We have so far conducted seven outdoor deployments (see Table III), ranging in size from 6 to 97 stations and from our campus to high-up in the Alps. During



(a) Data gathering reliability.



(b) Load distribution.

Fig. 10. Results of the testbed experiment over one full week of operation.

these campaigns, we have gathered hundreds of megabytes of environmental data, freely available for download on our Website.[5]

5.2.1 *Single-Hop Outdoor Deployments.* The first outdoor deployment was on the EPFL campus, using single-hop communications with multiple sinks and Ethernet gateways. We extensively tested our hardware during this deployment, while avoiding the added complexity of multihop routing. This field experiment is being used to assess the heat transfer of the urban environment

[5]`http://sensorscope.epfl.ch/index.php/Environmental_Data`

Table III.
Outdoor deployments conducted since we started the project. The first two deployments were limited to single-hop communications, while the others used multihop communications.

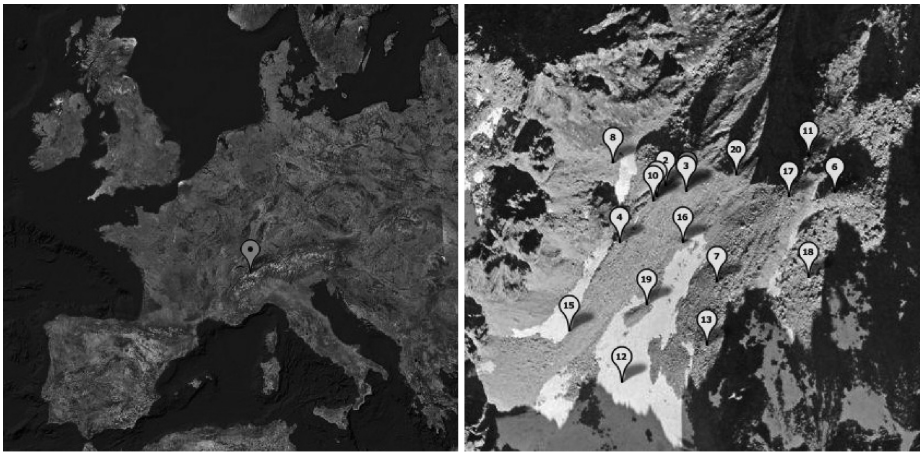
Place	Duration	Stations	Data Pts	Characteristics
Campus of EPFL	180 days (2006)	97	190 000 000	Large-scale WSN
Plaine Morte	4 days (2007)	13	1 300 000	Rapid deployment
Morges	30 days (2007)	6	750 000	First use of multihopping
Génépi	60 days (2007)	16	5 800 000	High mountain rock glacier
Grand St. Bernard	45 days (2007)	17	4 300 000	High mountain pass
Wannengrat	Ongoing (2008)	20	*n/a*	High mountain ridge
Campus of EPFL	Ongoing (2008)	10	*n/a*	Outdoor testbed

using 97 networked sensing stations [Nadeau et al. 2009]. We then deployed SensorScope on an alpine glacier to test the application at high altitude (3000 m) and tough environment (-20°C), and to gather atmospheric measures during a highly stable winter period. As these events are rare and unpredictable, they require a rapid deployment. We used once again single-hop communications, this time with a GPRS gateway. We thus proved the feasibility of a rapid deployment and also tested our system under very harsh conditions (e.g., extreme temperature values and variations, icing).

5.2.2 *Multihop Outdoor Deployments.* The first deployment requiring multihopping consisted of six stations located along a stream bank. Our goal, in collaboration with a federal project, was to gather measurements highlighting river-warming caused by systematically removing vegetation along the banks. Since our multihop communication protocol proved to be robust enough, we went on to more challenging deployments in high mountain environments.

5.2.3 *The Génépi Deployment.* Here, we focus on our most important high mountain deployment, which occurred on a rock glacier located at 2,500 m on the Génépi, in Switzerland. This site was chosen because it is the source of dangerous mud streams during intense rains, causing accidents by flooding the adjacent road. Furthermore, there are major concerns on the warming of the permafrost and potentially devastating release of rocks currently resting on a glacier and frozen soils. The authorities in charge did not have any measurements at that site and asked us to deploy SensorScope to correlate rain measurements with wind and temperature, based on the shape of the landscape. They provided us all the technical help, including a helicopter to deploy the stations during the last days of August 2007. They were taken down again two months later, in late October.

We deployed 16 stations on a 500×500 m area (see Figure 11), with special care on placement, to retrieve meaningful measurements. For instance, station 20 was specifically put at the dislocation border of the glacier, and station 11 in the soil slope. To transmit the packets to the server, the sink, placed close to station 3, was equipped with a GPRS module. Although GPRS connectivity was poor at this site, it was sufficient for the deployment to be successful.



(a) Global view. (b) Local view.

Fig. 11. The map of the Génépi deployment.

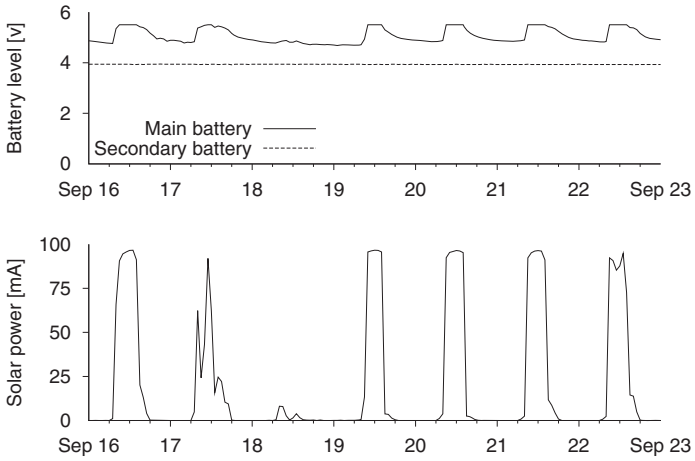
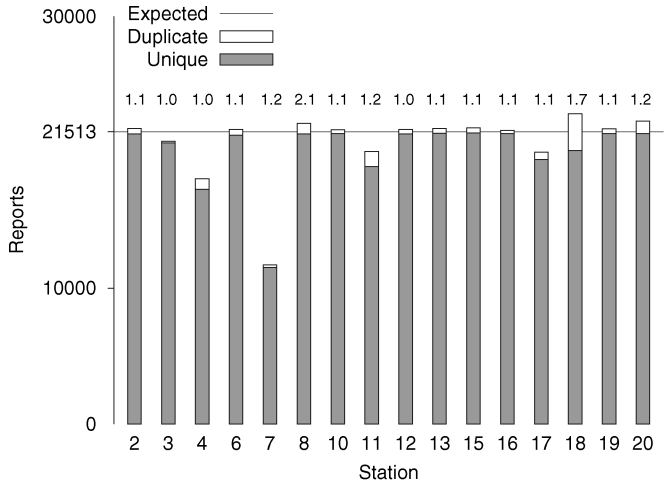


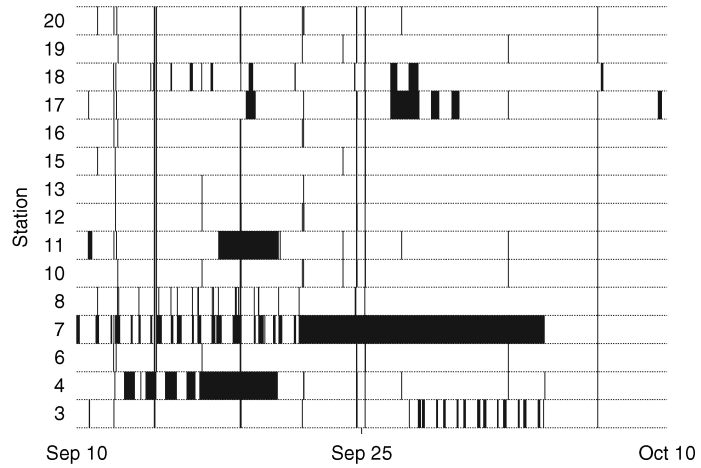
Fig. 12. One week of data from the solar energy system of station 15.

This deployment was a very good opportunity to thoroughly test the autonomy of the stations in real and harsh conditions. Figure 12 shows the variation of energy of station 15 for a whole week and the associated incoming solar power. During the observation period, the sunrise was around 06h00 and the sunset around 21h00. These results are more realistic than considering only the mote's consumption, since we also have external sensors as power consumers.

We can clearly see the depletion of the main battery during periods of low solar radiation, obviously at night, and its charging upon the sunrise until being fully charged during full daylight. On September 18, the weather was very cloudy, but the incoming solar power was still high enough for the battery to



(a) Data gathering reliability.



(b) Time-correlated losses.

Fig. 13. Data gathering statistics over one full month of the Génépi deployment.

charge sufficiently. During the whole week, the secondary battery was actually not used at all, and would have powered the system in case of a failure of the primary one. We did not notice any self-discharging phenomenon over the whole deployment period. Overall, we are satisfied with our energy system, and even if other hardware failures occurred, we did not have to worry about the energy level.

Figure 13(a) provides the sensing reports gathered during one full month, starting from the September 10, 2007, the numbers above the bars indicating the average hop count. We used the same set of parameters as for the indoor experiments and were able to collect almost all the reports from 10 stations.

Because of the importance of this deployment, we absolutely wanted it to be successful, so that we used a conservative approach that resulted, in conjunction with the clear outdoor environment, in having many stations at more or less one hop from the sink. Because of this, there are less duplicates than during the testbed run, but also because some interferences, which we may have in our building, do not exist on a mountain.

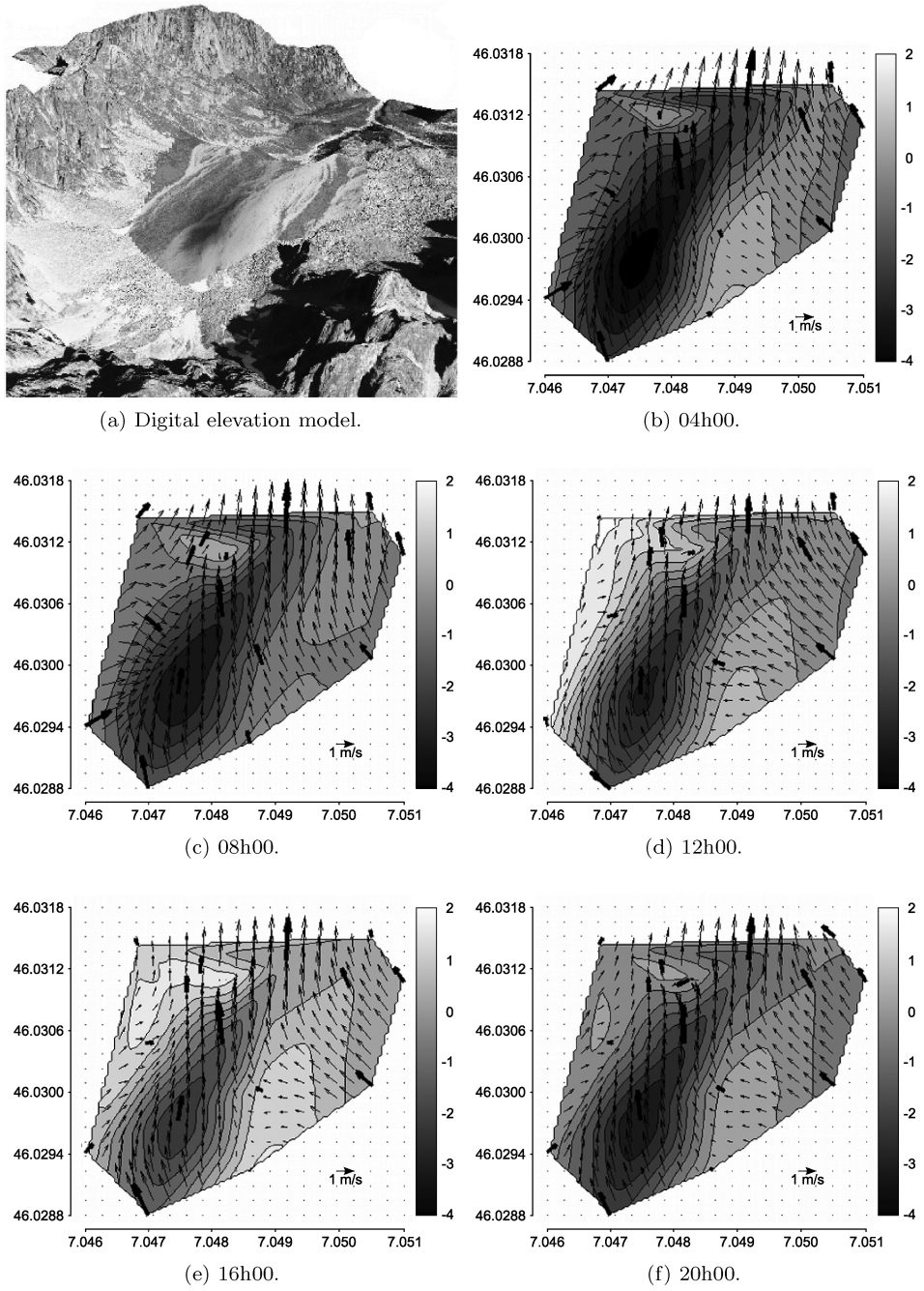
The missing packets were mostly due to hardware failures, such as short circuits, leading to the loss of several consecutive packets. This is apparent in Figure 13(b), which shows the time correlation of losses per station, with each black line representing a missing packet. For instance, station 7 suffered a severe short circuit, requiring on-site repair. Other failures were less drastic, and the corresponding stations were able to recover after some time. This figure also shows problems caused by poor GPRS connectivity, resulting in simultaneous losses for almost all stations.

5.2.4 *Qualitative Data Analysis from the Génépi.* The Génépi deployment occurred mainly because it was impossible to install a traditional weather station on site, due to the harsh environment. While this was a chance for us to prove the superiority of a WSN in such a situation, it also resulted in the lack of any "ground truth" station. Thus, to assess the quality of the gathered data, we had to analyze it manually, by comparison with a predetermined model and by cross-correlation among stations.

Figure 14(a) shows the digital elevation model of the rock glacier. We can see the valley in the center of the picture, where the permafrost is the thickest (around 10 to 15 m of ice under the rocks). This is also where the Durnand river originates, which is the source of the dangerous mud streams. The other maps of Figure 14 show the spatial distributions of air temperature and wind speed during October 23, 2007. During that day, there was perfectly sunny weather, with a light wind from the south. Values are averaged over one hour.

These results show that, while the variation of temperature is around 5°C on the border of the site, the maximal variation of the valley is only around 2°C. This is interesting because the corresponding stations were placed along the same axis and faced the same sun exposure, so they should have observed the same temperature. This difference is actually caused by the thick layer of ice under the granite rocks, located along the valley. Thus, the temperature is kept low at that place, even with exposure to the sun during the day. Furthermore, the cold dense air flows down the granite rocks, and creates a strong *katabatic wind* (between 2 and 4 m/s) at the slope break line of the valley.

Thanks to SensorScope and its multiple lightweight stations, we were able to go on the site and to identify this microclimate, which plays an important role in the model which is currently being elaborated, based on our measurements, to predict the evolution of the frozen soils. Moreover, gathering this set of data would not have been possible without multihopping, since the only possible spot to place the GPRS-enabled sink was on the border of the mountain, where GPRS connectivity was good enough. Single-hop communications would have constrained the other stations to being too close to the sink to collect the necessary data.



(a) Digital elevation model.

(b) 04h00.

(c) 08h00.

(d) 12h00.

(e) 16h00.

(f) 20h00.

Fig. 14. Digital elevation model (0.5×0.5 m resolution), and spatial air temperature and wind distribution over the Génépi rock glacier along the October 23 at 04h00, 08h00, 12h00, 16h00, and 20h00 (local time). Bold arrows are the actual measurements of the wind speed.

6. LESSONS LEARNED

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.

6.1 Hardware and Software Development

6.1.1 *Consider Local Conditions.* It is not always obvious how, possibly drastic, variations in temperature and humidity will affect hardware devices, 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 nevertheless faced many unanticipated hardware failures. For instance, during the Génépi deployment, we 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, simply 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 measurements until it was too late. It is therefore crucial to simulate the anticipated conditions as accurately as possible. Studying the impact of weather conditions 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.

6.1.2 *Sensor Packaging.* Outdoor packaging is difficult, as it must protect electronics from humidity and dust while being unobtrusive [Szewczyk et al. 2004]. International *Ingress Protection* (IP) codes are used to specify the degree of protection for electrical enclosures. The required level for outdoors is IP67, which provides full protection against dust and water, up to an immersion depth of one meter. Any lesser degree of protection exposes electronics to humidity and atmospheric contaminants, potentially leading to irreparable damages. Corrosion may cause the malfunction of a sensor connector, consequently corrupting the data from that sensor. Even more disastrous, humidity may cause a short circuit in the connector, resulting in permanent damage and/or continuous rebooting of the affected station.

A particular example is illustrated in Figure 15, which shows the reported rain falls during the Grand St. Bernard deployment. We can see that the rain meter of station 14 worked well during the first 12 days and reported erroneous values after that. The problem was 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. This example also shows how crucial it is to be able to detect failures as quickly as possible, for instance by using automatic outlier detection.

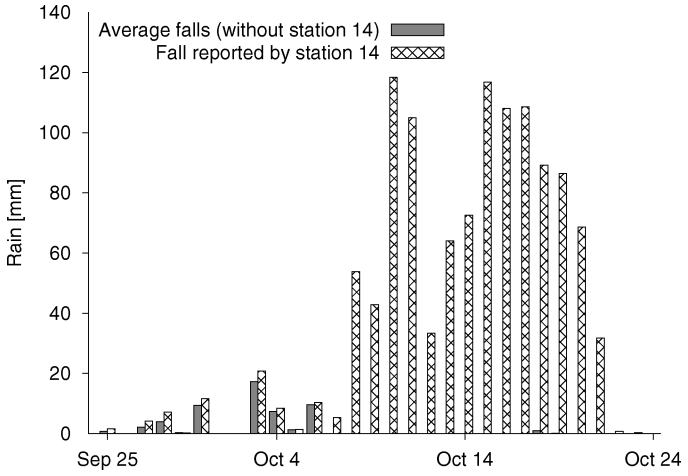


Fig. 15. An example of incorrect measurements caused by corrosion (Grand St. Bernard). After October 6, station 14 reported wrong values due to a short circuit caused by high relative humidity.

6.1.3 *Keep It Small and Simple.* Protocols must be well-fitted to the application, to avoid unexpected interactions between software components as much as possible [Buonadonna et al. 2005]. Sometimes, complexity cannot be avoided, but whenever the benefits are questionable, simple solutions should be preferred. For instance, as our stations are equipped with a solar panel, an overall positive energy balance is sufficient to achieve long-term autonomy. We were thus able to avoid complex, ultra low-power MAC layers, generally requiring high-precision synchronization (e.g., Dozer [Burri et al. 2007]), which may be very difficult to achieve in realistic conditions. Instead, we used a very simple, predictable mechanism. Furthermore, packet losses with such complex protocols are more likely to occur in harsh conditions (e.g., heavy rain), 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, making it easy to read and maintain.

6.1.4 *Think Embedded.* On a computer, code is easy to debug, using debugging statements or tools. It is 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, so that their behavior can be incorrect, even if the code itself is actually fine [Szewczyk et al. 2004]. It is thus important to be able to determine what happens *inside* the network. This issue has already been pinpointed—and proved to be of prime importance—in previous work, but is unfortunately still widely underestimated [Langendoen et al. 2006; Selavo

et al. 2007]. In SensorScope, besides 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 (e.g., number of data/control/non-acknowledged packets sent).

Topology. They contain a dump of the neighborhood table, including the identifier of each neighbor and its link quality.

Once the routing protocol is in place, it is easy to create such packets and let them go to the server like sensing packets. Figure 12, showing the energy level of a station during the Génépi deployment as well as the incoming solar energy, is an example of how our system can be monitored. This allowed us to determine that the backup battery was never used, even in case of consecutive cloudy days. Our next-generation energy board will thus use only a single, larger battery.

6.1.5 *Get All Data You Can.* Our primary goal 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 proven to be insufficient to extract some of the data we would like to analyze now. 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 early on what data is useful for networking issues, like performance analysis, the code to gather needed data can be incorporated into the development process.

6.1.6 *Data that is Useful.* A successful deployment consists not only of gathering data, but also of exploiting it. Generally, networking groups do not pay much attention to the latter. A WSN nevertheless 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 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 [Tolle et al. 2005].

In SensorScope, we all work in close collaboration. We have frequent meetings to discuss the project, allowing us to tackle problems very pragmatically. For instance, since a sampling rate of less than two minutes is useless for our application, we decided to omit network congestion management. Without such interaction, we may have ended up with solutions, potentially non-working, solving nonexisting problems. EFLUM also led all decisions regarding deployments (e.g., place, observations), which allowed us to obtain meaningful results with the potential for scientific impact.

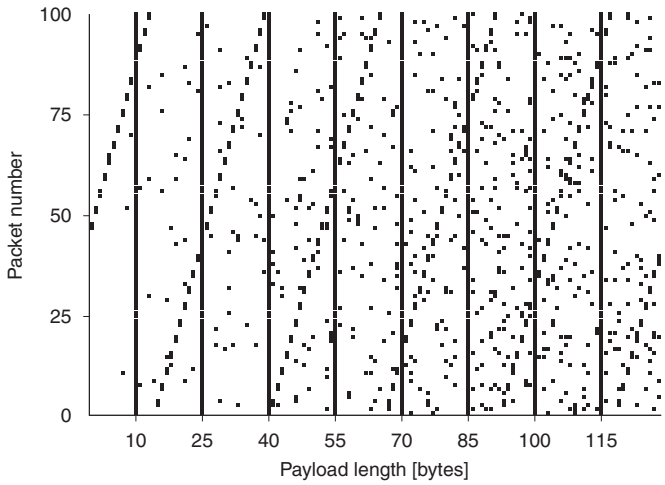


Fig. 16. Temporal distribution of losses on our indoor testbed, black squares being losses. A run of 100 packets was started for each payload length, at a rate of one packet every two seconds. Vertical lines are caused by a bug in our radio drivers; diagonal lines are caused by other sensor motes using the same frequency.

6.2 Testing and Deployment Preparation

6.2.1 *Check for Interferences.* When setting up a deployment, the first priority should be to inspect the radio spectrum to detect possible interferences. The optimal way is to use a spectrum analyzer, but due to its size, weight, and power consumption, it can be difficult to make use of it at the deployment site. A simpler way is to run a test program to determine losses over time. There are actually a lot of radio devices that can create interference, compromising the results and leading into thinking that the code is incorrect. For instance, Figure 16 provides the losses observed during a test run on our indoor testbed. Our primary goal was to see the receiving probability based on the payload length, but we quickly found that there was periodic interference at the frequency we were using. The pattern resulting from this interference (i.e., the diagonal lines) is clearly visible. We now have discovered that another group was using the same sensor motes as ours, using the same frequency.

6.2.2 *Data You Can Trust.* Although sensors should be pre-calibrated, some manipulations (e.g., packaging) can affect their measurements [Buonadonna et al. 2005]. An example is the Sensirion SHT75 for air temperature and humidity. While it is calibrated and should 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 are 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 North-oriented to provide sensible data. We forgot this detail in our first on-campus deployment, so that we had to return to each station to correct its orientation. 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 15, showing corrupted measurements).

6.2.3 *Be Consistent.* At some point one may be tempted to change some parameters or to switch to new 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. The exact same configuration should thus be used during both tests and real deployments. Another possible issue is the "last minute commit," which can kill a complete deployment [Langendoen et al. 2006]. For instance, Figure 16 also shows a bug in our radio drivers: some payload lengths do not work, that is, packets are either not sent or not received. This occurs when 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 one full day of debugging to pinpoint the bug. Obviously, we were glad to discover it 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.

6.3 Deployments

6.3.1 *Consider Local Conditions—Once Again.* Some bugs can be hard to spot before the real deployment, because they do not occur under normal testing conditions. For instance, Figure 13(b) shows simultaneous losses for all stations. After some investigation, we discovered that there was a bug in the GPRS drivers, preventing it from reconnecting to the cellular network upon an unexpected disconnection. Due to this bug, we remotely rebooted the GPRS a few times during the deployment, using GSM text messages. 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 13(b) 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.

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 temperature changes 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 may seem robust compared to wireless communications. We could actually have detected this

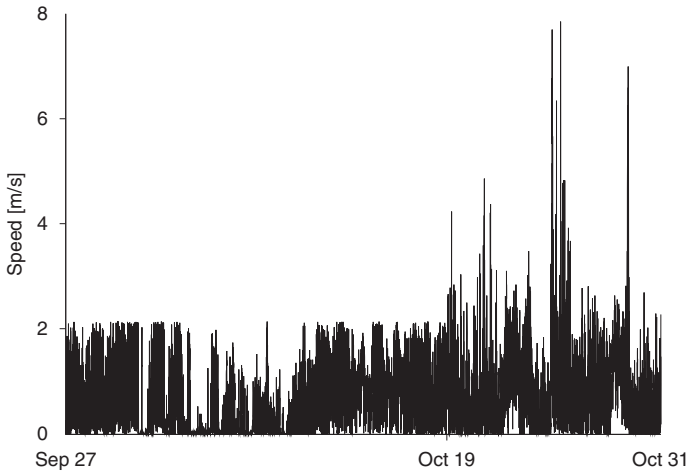


Fig. 17. Wind speed measurements of station 6 (Génépi). Due to a design problem in the sensor's driver, data gathered before October 19th is unusable.

problem by simply putting the GPRS into a freezer, as indicated earlier in this paper.

Manipulating electronics outdoors must be kept to a minimum. When sealed, the hermetic box we use effectively prevents the corrosion of electronics, but it has a flaw that we actually noticed when we encountered bad weather at a deployment site: we need to open the box to replace a sensor. Indeed, all sensors 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 it. This can be a problem, especially when weather conditions are poor (e.g., fog, rain, snow). Hence, our next generation of stations will feature a better box with external, hermetic plugs.

6.3.2 *Get a Watchdog.* Our deployments have taught us that all data must be scrutinized as soon as it reaches the server, to promptly detect problems and malfunctions. However, while some incorrect measurements may be easy to detect (see Figure 15, showing the malfunction of a rain meter), other problems may be more subtle. For instance, Figure 17 provides the wind speed reported by station 6 during the Génépi deployment. At first glance, the data may look correct, while actually it is not. Until October 19, because of a bug in the sensor's driver, all wind speed data is unusable. 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. 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 (Be

Consistent), this particular problem was difficult to detect. Correlating wind speed data among all stations would not have helped, since all of them were affected by this bug.

6.3.3 *Keep All Data.* On the back-end server, there will be various programs processing data. Nevertheless, all the raw data must be securely archived for future reference. There may be some statistics that were not envisioned at first. One may also discover that the equation used to transform raw data into SI units was 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 coming in from the network. Once a day, a new file is opened and the previous one is compressed. This way, we have archived approximately 20 MB of compressed raw dumps for the Génépi deployment alone. As these files are the most precious output from a deployment, they must be carefully backed up. Figure 13(b), showing time-correlated losses during the Génépi deployment, is an example of a plot that has been generated *a posteriori*, directly from these dumps.

6.3.4 *Data You Can Interpret.* 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, Figure 15 shows that there has been a good amount of precipitation on October 3rd 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 working on equipping one or more stations with a camera to provide visual feedback. We have already experimented in this domain, by installing an autonomous camera during the Génépi deployment. Figure 18 provides a sample image taken by that camera, showing a foggy and snowy day, with a thin layer of snow on the rocks. Much work is left though, mainly regarding energy management and bandwidth usage. Our camera prototype was indeed connected to a car battery and had its own, dedicated GPRS connection. On-site image analysis may be part of a possible solution.

6.3.5 *Traceability.* As the software on both server and motes will evolve over time, traceability is extremely important. The aforementioned wind speed bug during the Génépi deployment forced us to reprogram the motes. The raw dumps thus contain packets generated by two different code versions. The obvious problem is to distinguish between correct and incorrect 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, the reordered fields would not have posed a problem.

Fig. 18. Picture taken by an autonomous camera during the Génépi deployment. Weather conditions such as fog and snow are difficult to detect without visual feedback.

Traceability of individual measurements is also important. For instance, when a bad sensor is detected, it is common practice to exchange it. Without any further provision, it is impossible to determine which values from previous deployments should be double-checked. In SensorScope, we plan to tag all our motes and sensors with RFIDs. With the corresponding reader, it will be easy to scan stations during deployments to associate sensors and stations. Storing this information in a database will allow us to retrace the exact history of all devices and measurements.

7. CONCLUSION AND FUTURE WORK

In the course of its various deployments, SensorScope matured into a key project for MICS[6] (Mobile Information & Communication Systems), a national competence center, merging cutting-edge WSN technology (networking, sensing, hardware, software) with leading environmental monitoring (modeling, prediction, risk assessment). In particular, the Génépi deployment has been a thrilling scientific adventure, which resulted in the gathering of a unique set of micrometeorological data. This allowed us to model a particular micro-climate, which will be used in flood monitoring and prediction, potentially reducing a well-known, but poorly understood, environmental hazard. This shows the potential of SensorScope for risk prevention.

This deployment also revealed how remote management is crucial in such harsh conditions. Dynamic reconfiguration of network and motes is our next main objective, and support for a system such as Deluge [Hui and Culler 2004] is of high interest. From the network management point of view, we also plan to implement measures to cope with asymmetric links, which result in transmission failures and an overly high radio usage. Finally, due to the difficult measurement conditions, the measured data is of variable quality. Thus, signal

[6]http://www.mics.org

processing techniques for better calibration, detection of outliers, denoising, and interpolation will be developed.

ACKNOWLEDGMENTS

We would like to thank the following colleagues for their invaluable help in turning SensorScope into a success story: Thierry Bertholet, Davis Daidié, Mounir Krichane, John Selker, Thierry Varidel.

REFERENCES

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

Burri, N., von Rickenbach, P., and Wattenhofer, R. 2007. Dozer: Ultra-low power data gathering in sensor networks. In *Proceedings of the ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*.

Dubois-Ferrière, H. 2006. Anypath routing. Ph.D. thesis, École Polytechnique Fédérale de Lausanne (EPFL).

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

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

Hui, J. and Culler, D. 2004. 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)*.

Jiang, X., Polastre, J., and Culler, D. 2005. Perpetual environmentally powered sensor network. In *Proceedings of the ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*.

Kim, Y., Schmid, T., Charbiwala, Z. M., Friedman, J., and Srivastava, M. B. 2008. NAWMS: Nonintrusive autonomous water monitoring system. In *Proceedings of the ACM International Conference on Embedded Networked Sensor Systems (SenSys)*.

Langendoen, K., Baggio, A., and Visser, O. 2006. 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)*.

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

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

Maróti, M., Kusy, B., Simon, G., and Lédeczi, A. 2004. The flooding time synchronization protocol. In *Proceedings of the ACM International Conference on Embedded Networked Sensor Systems (SenSys)*.

Nadeau, D., Brutsaert, W., Parlange, M., Bou-Zeid, E., Barrenetxea, G., Couach, O., Boldi, M.-O., Selker, J., and Vetterli, M. 2009. Estimation of urban sensible heat flux using a dense network of wireless observations. *Environ. Fluid Mechanics. 9*, 6, 635–653.

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

Raghunathan, V., Kansal, A., Hsu, J., Friedman, J., and Srivastava, M. 2005. Design considerations for solar energy harvesting wireless emebedded systems. In *Proceedings of the ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*.

SCHAEFER, G., INGELREST, F., AND VETTERLI, M. 2009. Potentials of opportunistic routing in energy-constrained wireless sensor networks. In *Proceedings of the IEEE European Workshop on Wireless Sensor Networks and Applications (EWSN)*.

SELAVO, L., WOOD, A., CAO, Q., SOOKOOR, T., LIU, H., SRINIVASAN, A., WU, Y., KANG, W., STANKOVIC, J., YOUNG, D., AND PORTER, J. 2007. LUSTER: Wireless sensor network for environmental research. In *Proceedings of the ACM International Conference on Embedded Networked Sensor Systems (SenSys)*.

SZEWCSZYK, R., MAINWARING, A., POLASTRE, J., ANDERSON, J., AND CULLER, D. 2004. Lessons from a sensor network expedition. In *Proceedings of the IEEE European Workshop on Wireless Sensor Networks and Applications (EWSN)*.

TOLLE, G., POLASTRE, J., SZEWCZYK, R., CULLER, D., TURNER, N., TU, K., BURGESS, S., DAWSON, T., BUONADONNA, P., GAY, D., AND HONG, W. 2005. A macroscope in the redwoods. In *Proceedings of the ACM International Conference on Embedded Networked Sensor Systems (SenSys)*.

WANG, H., CHEN, C. E., ALI, A., ASGARI, S., HUDSON, R. E., YAO, K., ESTRIN, D., AND TAYLOR, C. 2005. Acoustic sensor networks for woodpecker localization. In *Proceedings of the SPIE conference on Advanced Signal Processing Algorithms, Architectures, and Implementations*.

WERNER-ALLEN, G., LORINCZ, K., WELSH, M., MARCILLO, O., JOHNSON, J., RUIZ, M., AND LEES, J. 2006. Deploying a wireless sensor network on an active volcano. *IEEE Inter. Comput. 10*, 2, 18–25.

WOO, A., TONG, T., AND CULLER, D. 2003. Taming the underlying challenges of reliable multihop routing in sensor networks. In *Proceedings of the ACM International Conference on Embedded Networked Sensor Systems (SenSys)*.

ZIMMERMANN, H. 1980. OSI reference model—The ISO model of architecture for open systems interconnection. *IEEE Trans. Comm. 28*, 4, 425–432.

Received November 2008; revised May 2009; accepted August 2009