PhD Thesis Dissertation

International Doctorate School in Information and Communication Technologies

FBK-IRST - Bruno Kessler Foundation
DISI - University of Trento

Guaranteeing Communication Quality in Real World WSN Deployments

Matteo Ceriotti

Advisor:

Dr. Amy L. Murphy

Bruno Kessler Foundation (FBK-IRST)

April 29, 2011

Für Uns

She had never before seen a rabbit with either a waistcoat-pocket,
or a watch to take out of it, and burning with curiosity,
she ran across the field after it

Lewis Carroll

The following document, written under the supervision of **Dr. Amy L. Murphy**, was reviewed by:

Prof. Prabal Dutta	University of Michigan, USA
Prof. Koen Langendoen	Delft University of Technology, The Netherlands
Prof. Leo Selavo	University of Latvia (Riga), Latvia

Abstract

Networks of smart interconnected objects have allowed the integration of the artificial world into the physical one. The interaction over a wireless medium is simultaneously the technology enabler and the primary hindering factor. The complexity and variability of the behavior of low power wireless communication is one of the challenges making the design and deployment of a system based on this technology a unique and demanding experience. In this thesis, we describe the deployment of two operational systems for structural health monitoring and adaptive lighting, undertaken by our research group. Our major contribution, among others, covers the definition and implementation of the system services enabling the monitoring infrastructure to guarantee the required quality. The resulting unique design and reliability provide concrete support to the vision of wireless sensor networks as dependable monitoring infrastructure.

Despite the success in meeting the user needs, the simple yet effective solutions exploited in the aforementioned deployments make apparent the limitations of the widely used approaches to coordinate access to the communication medium. This thesis also argues that the currently employed solutions at the MAC layer are insufficient to provide guarantees to the resource user. Therefore, we introduce REINS-MAC, a Time-Division Multiple-Access (TDMA) communication scheduler that coordinates access to the medium in a fully decentralized fashion. Limited flexibility, scalability, robustness, as well as increased overhead and complexity are commonly recognized shortcomings of TDMA approaches. REINS-MAC overcomes these limitations by adapting the scheduling to match local availability and natural connectivity variations. Moreover, each node is empowered with full control over its own communication resources.

The ability to anarchically apply changes to the communication schedule allows the steering of the resource allocation towards individual needs, dictated by the higher layers in the network stack. The resulting quality and anarchy in accessing the communication resource affect the design and implementation of WSNs, opening new horizons where the application regains control of the primary resource: communication.

Keywords: Wireless Sensor Networks, Low-power Wireless Communications, Network Architecture and Design, Network Protocols, TDMA

Contents

List of Figures

List of Tables

Part I

Introduction

Chapter 1

Introduction

The last decade has seen the progressive interconnection of the physical and the artificial world through the development of intelligent sensing and actuating computing infrastructures. The most prominent and challenging example is networks made by potentially hundreds of tiny devices with scarce resources, also called Wireless Sensor Networks (WSNs). Each node in the network has limited energy, memory, and computational capabilities and interacts with the others using wireless communication. The network as a whole overcomes the deficiency of the individuals by sharing resources. A myriad of traditional applications have already benefited from this technology, as in the case of environmental monitoring, while new ones, such as wildlife monitoring, have been made possible.

Despite the vision of WSNs, the successful design and deployment of a monitoring infrastructure based on such a technology still present open challenges, e.g., programming effort and reliability of data gathering services. In Torre Aquila, our group successfully deployed a system for structural health monitoring, in which we acquired expertise in the field. In our second deployment we further demonstrated the dependability of the technology by integrating it in a closed loop to adaptively control lighting in road tunnels. Our work provided further support to the vision of the technology as capable to concretely impact everyday life.

The access to the wireless communication medium enables the resource sharing and ultimately the provisioning of system services. The experience gathered in the aforementioned deployments provided the groundwork to identify both the limitations of common medium access control techniques and the need for new communication scheduling solutions. The highlighted challenges and limitations lead to the definition of Reins-MAC, a versatile fully-distributed TDMA communication scheduler, the core contribution discussed in this thesis. Positioned low in the communication stack, Reins-MAC provides the foundation for higher level abstractions to rein in the protocol anarchy, providing control of communication quality to the application. In the remainder of this chapter, we underline the contributions of our work.

Deploying Systems

The untethered monitoring infrastructure provided by WSNs is an attractive solution in several domains. Ease of deployment, limited visual impact, and flexible sensing configuration are the key assets interesting structural engineers in monitoring heritage buildings. We designed and installed one such a system in a medieval tower, Torre Aquila in Trento (Italy). The building was instrumented with an untethered monitoring infrastructure composed of heterogeneous sensors, i.e. vibration, deformation and temperature. The contributions of the work, undertaken by our group, range from the hardware to the graphical front-end. Remarkably, we:

- designed a data collection solution able to efficiently handle heterogeneous classes of traffic, with both high and low data rates;
- based the design and development of the system on a tuple space abstraction, TeenyLIME, reducing the programmer effort.

The system has been working for more than two years and the data reported has been helpful in assessing the tower's actual stability. The invaluable in-the-field experience created the base for subsequent deployments.

Reduced installation and maintenance costs, and ease of incorporation in operational systems promote the integration of WSNs with conventional industrial-strength equipment. In the second deployment in which our group was involved, the WSN was used as part of a closed control loop in charge of adjusting the level of the lamps inside operational road tunnels based on current external and internal conditions. In this setting, we:

- demonstrated the feasibility and effectiveness of WSNs in combination with standard tunnel equipment;
- investigated to what extent mainstream solutions can be successfully reused, further taking advantage of the software decoupling fostered by TeenyLIME to use components deployed in Torre Aquila.

The deployment of networks in a small test site with a low traffic volume as well as in a longer and more trafficked road tunnel demonstrated the ability of current WSN technology to meet the final application requirements.

Bringing Quality into Communication

Building reliable and predictable systems based on WSNs is difficult. In addition to the inherent unreliability and variability of the communication links, the techniques employed to share the access to the wireless medium have a deep impact on the services making use of communication. Protocols coordinating access to the communication medium in a random manner are easy to implement and very flexible, avoiding the costs of scheduling communication. This category of MAC algorithms is the de facto standard for WSNs and is what we employed in the aforementioned deployments. Despite the success of the approach, we experienced ourselves the impossibility to effectively implement quality differentiation, and both traffic flow and latency control. In contrast, TDMA protocols offer guarantees by enforcing a rigid slot-based communication scheduling. One commonly recognized limitation of these approaches is their inability to adapt to the dynamic behavior typically seen in networks of low power cooperating objects. Further, their inherent complexity and required compliance to a rigid discipline foster the belief that dynamic changes to the communication schedule are infeasible. In this thesis, we prove the opposite.

Our fully-distributed protocol, Reins-MAC, clearly demonstrates that a dynamically adjustable, flexible solution is feasible. In contrast to existing approaches, in Reins-MAC we:

- defined a distributed online scheduling mechanism that forms and reserves slots of variable size, tailoring medium access to network conditions that vary in time and space;

- provided the foundation for higher level abstractions to express communication quality needs and steer the resource allocation.

Reins-MAC rejects the common TDMA assumption that a single, network-wide slot size is required. To achieve this, each individual node controls the beginning of its own slot by placing it in the middle between the two surrounding slots owned by the neighbors. In this way, it adapts the slot size at each node to match local availability, achieving tremendous gains in bandwidth utilization. The approach that accomplishes this simultaneously allows Reins-MAC to adapt to the natural connectivity variations present in WSNs.

Further, we use a simple mechanism to identify a network-wide shared slot dedicated to coordination, in which each single node can submit requests for changes. Therefore, the flexible slot sizes of Reins-MAC can be explicitly tuned to support dynamic application requirements, providing guaranteed communication. Both through simulations and experiments in a real testbed, we demonstrate the effectiveness and feasibility of the proposed solution. The strengths of the approach arise from its algorithmic simplicity, grounded in the theoretical literature on pulse-coupled oscillators (a.k.a. firefly pulsing), and full decentralization, where each node is empowered to make changes to the communication schedule autonomously. The higher layers, thanks to the application knowledge, are in charge of reining in such protocol anarchy.

The TDMA nature of Reins-MAC is profoundly different from the CSMA one of the MAC protocol we employed in our deployments. For this reason, the implementation of the system services on top of Reins-MAC must be rethought. We investigated such an impact by revising the basic building blocks of the network protocols employed in our deployments. As a result, Reins-MAC:

- promoted a radically different design, as a consequence of the deterministic and flexible resource allocation scheme provided;
- extended its impact on both the definition of higher level abstractions and the way programmers use them.

In addition to typical expects deriving from the difference between scheduled and random communication access mechanisms, Reins-MAC has the distinctive ability to allocate different resources to each individual node in the network. This feature opens new horizon to the definition of new network solutions supporting Quality of Service.

Observing Communication

The effectiveness of any introduced networking solution relies on the characteristics of the underlying physical layer. The influence of the environment both on the actual communication among devices and ultimately on the system services makes building a system a unique and demanding experience. Facing these limitations requires direct expertise and proper supporting tool. The thesis is concluded by a description of a preliminary deployment of a WSN in a primary cloud forest, carried out by biologists in collaboration with our group. In this context, we:

- investigated connectivity, in terms of reliability, stability, and link asymmetries in an unfamiliar scenario;
- studied the combined usage of mobile and stationary nodes as an exploration tool to characterize communication in unknown environments.

This work demonstrates how each WSN installation is a peculiar experience where the environment plays a major role. The installation of a network is destined to manifest problems that can be now solved solely with in-field expertise. To make the vision of WSNs concrete, new deployment and management solutions must be defined to make the technology accessible to scientists.

Thesis Organization

The thesis is organized following the structure presented in this introduction. Our contributions in developing systems to monitor heritage buildings and control lighting in road tunnels are described respectively in Chapters 2 and 3. The discussion leads to the introduction of Reins-MAC in Chapter 4, followed in 5 by its influence on the design of the system services we previously employed in our deployments. Our concluding experience with a deployment in a tropical cloud forest is discussed in Chapter 6. Finally, Chapter 7 closes the work with a discussion of the possible research directions uncovered with this thesis.

Part II

Deploying Systems

Chapter 2

Monitoring Heritage Buildings

WSNs have the clear potential to empower the end user with new and previously impossible ways of gathering data from the real world. However, the characteristics of the low-power wireless communication at the core of the technology depend on the environment in which employed [74, 57]. The scarcity of resources, e.g., individual energy, memory, and communication, requires simple yet effective solutions [69]. Moreover, the limited knowledge in building operational systems makes the creation of a fully functional solution satisfying the final user needs a learning and failure prone experience [40, 1].

In this chapter, we describe the first successful experience of our group with real world deployments of WSNs. We demonstrated that building a reliable monitoring infrastructure based on WSN technology is indeed possible and useful to the final user.[1]

2.1 Scenario, Motivation and Contribution

Heritage buildings are a fundamental constituent of a country's historical memory. Their preservation is thus a major concern. Planning the maintenance of such structures requires a careful assessment of their structural integrity, along with a precise and quantitative understanding of the factors that may affect them. The latter is traditionally achieved through sensors and data loggers monitoring quantities such as vibrations, temperature, and humidity. However, these devices are typically cumbersome to deploy, as they require a nearby power outlet or extensive wiring. Therefore, their number is limited, and so is the monitoring: this is especially true in buildings containing works of art, due to the visual impact and physical encumbrance of the instrumentation.

[1]The content of this chapter is a joint work with Luca Mottola, Gian Pietro Picco, Amy L. Murphy, Ştefan Gună, Michele Corrà, Matteo Pozzi, Daniele Zonta, and Paolo Zanon, published in "Monitoring Heritage Buildings with Wireless Sensor Networks: The Torre Aquila Deployment", 8^{th} ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN'09, SPOTS track), San Francisco (CA, USA), April 2009 (Best Paper Award) [9].

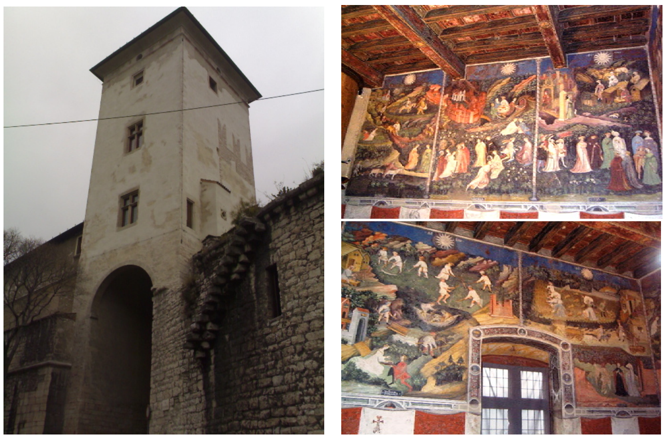

(a) External view. (b) Inside views.

Figure 2.1: Torre Aquila.

In this context, wireless sensor networks (WSNs) enable radically different solutions overcoming the above limitations. Small, self-powered nodes relying on radio communication reduce the invasiveness of the system, allow the deployment of more devices, and enable experimenting with different configurations of the sensing infrastructure.

Torre Aquila. The above requirements were evident in Torre Aquila, where we conducted the study reported in this chapter. Located in the city of Trento (Italy) close to the Buonconsiglio Castle, it is a 31 meter-tall medieval tower whose 2^{nd} floor contains "Il ciclo dei mesi" ("The Cycle of the Months"), a series of internationally-renowned frescoes that represent a unique example of non-religious medieval painting in Europe, attracting thousands of visitors every year. The tower and some of the frescoes are shown in Figure 2.1.

The preservation of the frescoes is the main source of concern for the local conservation board. In ancient times Torre Aquila represented the main entrance to the city from the East: with the expansion of the city in the second half of the 19^{th} century, most of the eastern city wall was demolished and the entrance to the city was moved a few hundred meters south of the original gate. Today this solution is inadequate for the increasing vehicular traffic. The solution to this problem, pursued by the Municipality of Trento, is to bypass the obstacle of the Castle compound with a road tunnel. The construction of the tunnel has been long delayed due to concern by the conservation board that construction work might cause unwanted settlement of the tower foundations. The timely estimation of the potential risk to the frescoes requires real-time monitoring and appropriate response models to reproduce the structural behavior of the tower.

Peculiarity of the WSN deployment. The use of WSN for monitoring the integrity of civil structures is not new, as we discuss in Section 2.2. Nonetheless, Torre Aquila poses peculiar challenges that are not usually found in the deployments reported in the literature:

- *Heterogeneity.* The system contains many kinds of sensors, whose operation is quite different. Deformation and environmental parameters can be sampled at a low rate, but vibration must be monitored at a high rate, which consequently demands efficient reporting of the resulting high volume of data. Both modalities must gracefully co-exist in the same sensing infrastructure.
- *Temporal span.* The time constants of the phenomena of interest require monitoring to span months or even years. In contrast, the systems found in the literature typically operate for at most a few weeks.
- *Online tasking.* The ability to change the behavior of the sensing infrastructure based on external input can be very useful. For instance, it is interesting to monitor vibrations when a visit by a large group of people is expected, or when strong winds are forecast.

Contribution. In this chapter, we present the hardware/software solution we developed to efficiently address the above requirements for monitoring Torre Aquila.

The hardware core is based on TMote-like devices, customized as illustrated in Section 2.3. Deformation measurements are acquired by fiber optic sensors stretching the length of the tower. These sensors, developed especially for our deployment, required custom integration with the motes used to report the measurements. Moreover, high-rate sampling and reporting of vibration data demanded buffering into a short-term storage. The flash memory usually found in motes is ill-suited for this task, due to its high latency, energy consumption, and limited number of writes. Hence, we integrated on the mote a 32 Kbyte FRAM (Ferromagnetic RAM) chip, overcoming all of these problems. To the best of our knowledge, we are the first to use FRAMs in a WSN deployment.

Unlike the hardware, our software layer is not based on what can be considered a "standard" core. Instead of developing directly on top of the operating system, we chose to empower our developers with the higher level of abstraction provided by a WSN middleware, TeenyLIME [16]. We are unaware of studies reporting the use of a WSN middleware in the context of a real-world, long-running deployment. Moreover, as illustrated in Section 2.4, our use of middleware is not limited to the application logic: the lower-level services necessary to the system operation (i.e., data collection, data dissemination, and time synchronization) are all implemented directly on top of TeenyLIME.

Deployment details such as the placement of nodes and sensors are reported in Section 2.5. In the same section, we show and interpret data gathered during 4 months of operation, as an example of the insights gained about the tower status. After looking at our implementation from the end-user's perspective, Section 2.6 analyzes it from a system one. We evaluate the system performance w.r.t. data delivery and lifetime—often considered key metrics in WSN deployments—showing that our implementation achieves a data delivery close to 100% while working at very low power. Moreover, we discuss the benefits brought to development by the use of our middleware, in terms of reduction of programming effort and code reuse.

2.2 Related Work

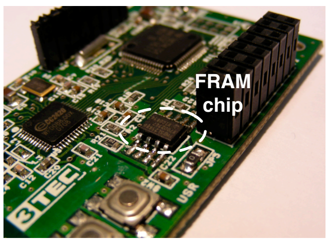
In [36], the authors note that WSN deployments to date can be divided in two categories: environmental monitoring applications (e.g., [50]), designed with low-power operation allowing them to run for long periods, and high-rate, high-fidelity ones running only for a relatively short time. The deployment in Torre Aquila inherits challenges from both classes, as we must deal with high-rate data and yet the system is required to operate for long periods.

In general, although WSNs have been used for monitoring civil structures [36, 11, 47, 27, 85, 12], the combination of requirements we must address is unique. For instance, only a handful of the systems surveyed in [47] can be tasked remotely, and in these cases (e.g., [12]) the implementation lacks support for low-power operations, hampering their use in long-running deployments. Similarly, most deployments deal only with monitoring vibrations [47], without the increased complexity due to heterogeneous sensors, as in Torre Aquila.

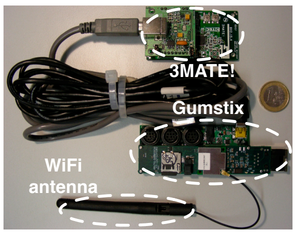
In some cases, the hardware is designed bottom-up for a given deployment. For instance, the work in [36] uses directional antennas, motivated by the peculiar shape of the target area. We cannot afford the luxury of fixing the network topology, as structural engineers are likely to relocate the nodes over time. Moreover, the nodes used by [36] cost ∼$600 each, which in our case would make the WSN solution not cost-effective compared to a traditional one. Instead, one of our customized nodes costs ∼$120.

On the software side, real-world deployments mostly feature ad-hoc implementations [36, 47, 85], which make very difficult extending or adapting their functionality to different scenarios. Moreover, where higher-level approaches have been proposed [27, 12] the deployments targeted short-term use. Our middleware-based one sustains good performance over a long time span, and yet fosters component reuse in other scenarios, as we discuss in Section 2.6 and show in practice in Chapter 3.

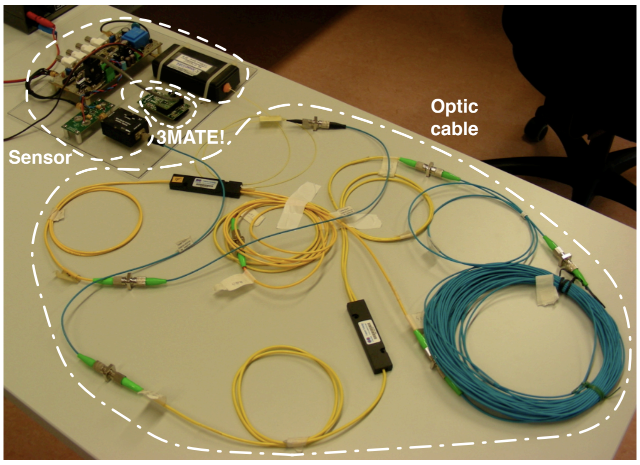
In summary, our goals set us apart from the state-of-the-art. We are neither confirming with a proof-of-concept "the *eventual* ability to cover a large civil structure with low-cost wireless sensors" [11] nor we are validating *already known* models using WSNs instead of conventional systems [36]. Our requirements, set by the structural engineers on our team, are instead to design, implement, and deploy an operational system that, by delivering good performance over a long period, helps them to assess the status of Torre Aquila.



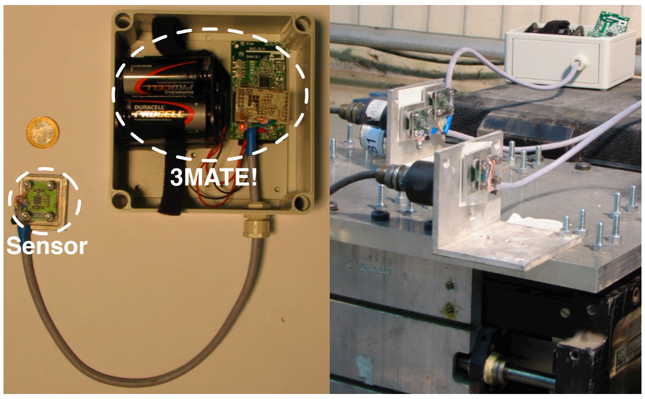
(a) 3MATE! node.



(b) Gumstix device as sink.



(c) Fiber optic sensor.



(d) Acceleration node and calibration.

Figure 2.2: Custom WSN hardware for Torre Aquila.

2.3 Hardware

Our requirements demand customized hardware. We selected as the core platform 3MATE! nodes, developed by TRETEC (`www.3tec.it`), an easily extensible WSN node similar to the TMote Sky [66], shown in Figure 2.2(a). The base 3MATE! is equipped with a TI MSP430

CPU, a ChipCon 2420 radio, and an inverted-F microstrip antenna. Differently from TMotes, the USB interface can be detached if not needed, reducing power consumption once deployed, and the board layout is designed to easily accommodate customized extension boards. Co-location with the manufacturer helped us to accommodate rapidly the needs of our deployment. The nodes have been customized differently according to their sensing goals, as described next.

Environmental nodes. We developed a 3MATE! extension board for environmental monitoring, equipped with simple analog temperature, relative humidity, and light sensors. In the deployment reported in Section 2.5, however, temperature was the only measure required by the end user. Sensitivity to temperature ranges from $-40°C$ to $125°C$ with a typical accuracy of $0.5°C$. This is sufficient to study phenomena such as temperature gradients across different floors.

Deformation nodes. To study the tower deformation, we required a minimally-invasive solution with very high precision. We developed a dedicated Fiber Optic Sensor (FOS) and the corresponding 3MATE! extension board, both shown in Figure 2.2(c). The sensor and its microcontroller-based control electronics, developed by TRETEC and University of Trento, work by differentially measuring the time taken for a laser pulse to travel through a pair of fiber optic cables wrapped around the monitored object. As the latter deforms, the cable stretches, modifying the travel time of the pulse. This solution is immune to electromagnetic noise and can be used to measure deformation on different physical scales, e.g., from individual walls to entire buildings.

The FOS is composed of a read-out unit with a synchronous laser pulser and a high-resolution optical receiver, and the optical path formed by fiber optic cables and splitters. Differently from all other sensors in Torre Aquila, the characteristics of FOS electronics require external power to ensure a stable measurement. The expansion board contains also a temperature sensor similar to the ones above, useful to correlate deformation with temperature in the same location.

Acceleration nodes. To measure vibration we used an analog, ultra-compact, tri-axial acceleration MEMS sensor (ST LIS3L02AL), integrated on a custom 3MATE! board connected through an extension cable that allows the sensor to be placed outside the node package, as illustrated in Figure 2.2(d). The sensor features a full range of $\pm 2\ g$ and is capable of measuring accelerations over a bandwidth of 1.5 KHz, with a resolution of 1 mg over 100 Hz bandwidth. We computed calibration coefficients for each sensor with induced vibrations at different frequencies and amplitudes using a shake table and piezoelectric accelerometers for seismic vibrations, shown on the right of Figure 2.2(d).

High-volume data such as vibration pose severe demands on buffering space. Some deployments [36] use the flash chip on the mote as a temporary buffer. However, this is a viable option only if the system operates for a limited time span, as the bound on the number of write operations eventually results in corrupted data. Instead, we equipped the 3MATE! with a FRAM chip, shown in Figure 2.2(a). Compared to flash memory, FRAM features lower power

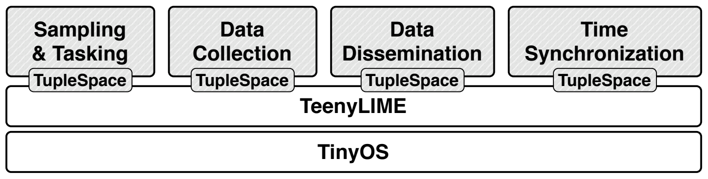


Figure 2.3: Software architecture.

consumption, virtually unlimited write-erase cycles, and faster write speed, enabling higher sampling rates. In our experiments, the flash could sustain at most 500 Hz sampling, whereas the FRAM allowed up to 1 KHz. Nonetheless, the storage area provided by FRAM is generally smaller than flash. In our case this is not an issue, as we use our 32 Kbyte FRAM as a temporary buffer, freed progressively as data is reported to the sink, described next.

Sink node. In Torre Aquila, the sensed data converge from all nodes to a sink where they are collected and stored, requiring a computing device with enough storage space and processing power. Moreover, this device must double as a gateway to interconnect with the front-end, allowing remote users to interact with the system. Finally, the requirement to reduce invasiveness holds also for the sink.

To address these needs, we chose a Gumstix [28] device, shown in Figure 2.2(b). Gumstixs are easily customizable embedded PCs with a very small form factor. We equipped ours with a board to use Secure Digital (SD) storage cards, a WiFi card to reach the external network, and a USB board for connecting a 3MATE! to access the WSN. As shown in Figure 2.2(b), the space required for this configuration is very small: it uses the same packaging of the WSN nodes.

2.4 Software Design

The design of WSN software is often characterized by ad-hoc solutions built directly on top of the operating system. The consequence is that systems become difficult to maintain and reuse is hampered [69, 1]. In our deployment we took a different stand, and addressed since the beginning the challenge of designing the software layer through higher-level abstractions that simplify development and foster code reuse.

Architecture. Figure 2.3 shows the high-level architecture of our software layer. The various macro-components interact *exclusively* through a shared memory space where data is read or written as tuples, sequences of typed fields. The tuple space abstraction is provided by a middleware called TeenyLIME [16], concisely described next. Its constructs are used to implement both application-level functionality (e.g., sensor sampling) and system-level mechanisms (e.g., routing and time synchronization), providing a unifying high level of abstraction throughout the software stack.

The reliance on this shared tuple space yields a highly decoupled software configuration, boosting code reuse both within and across deployments. For instance, the software deployed on acceleration nodes differs from that of environmental nodes *solely* in the sampling functionality, which inevitably depends on the quantity to sense. Moreover, it makes it easier to design alternative deployments by removing or replacing components, without affecting the others.

TeenyLime in a nutshell. As shown in Figure 2.4, in TeenyLime each node hosts a tuple space *shared* among 1-hop neighbors: a node perceives its tuple space as containing the tuples stored locally plus those residing on its neighbors. Software components atop TeenyLime interact locally or across nodes by reading/writing tuples from/to the shared tuple space. If needed, however, the read/write operations can be scoped to access directly the local tuple space of a neighbor. Read operations occur by requesting a match against a *pattern*: its fields express a constraint on the field type or value in the tuples being considered for matching. For instance, a pattern ⟨"foo", ?integer⟩ matches the tuple ⟨"foo", 20⟩ but not ⟨"foo", "boo"⟩. Moreover, TeenyLime provides a form of data listener called a *reaction*, a code fragment whose execution is automatically triggered upon the appearance of a matching tuple in the shared tuple space. This provides a very powerful way to increase the decoupling among different functionality. Other TeenyLime constructs are described in the following, whenever appropriate. TeenyLime is implemented in nesC on top of TinyOS. Therefore, operations are asynchronous and their result is signalled to the caller component through an event. A complete description of the middleware, including API and implementation details can be found in [16].

We now describe the design of the main components in Figure 2.3. In every case, we first highlight the requirements and challenges, and then report on the component design and implementation in TeenyLime.

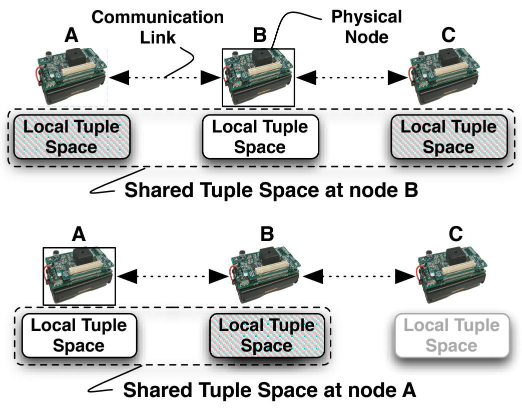


Figure 2.4: Tuple space sharing in TeenyLime.

Node type	Operating parameters	Typical value
Environmental	Sampling period P	10 min
	# of sampling sessions N	infinite
Deformation	# of samples averaged per session A	10
	Sampling period P	10 min
	# of sampling sessions N	infinite
Acceleration	Sampling frequency F	200 Hz
	Sampling duration D	30 s
	# of sampling sessions N	infinite

Figure 2.5: Node types and their typical configuration.

2.4.1 Sampling and Data Collection

Requirements and challenges. The deployment in Torre Aquila is characterized by heterogeneous sensor nodes whose sampling requirements and modalities vary greatly, as seen in Figure 2.5. This affects not only the local processing, but also the routing protocols employed for data collection, where reliability guarantees also play a key role. Based on our scenario, we identify two classes of traffic for data collection:

I. *Bursty, high-rate* data with *strong reliability* requirements, i.e., those coming from acceleration nodes. Large amounts of data are locally stored in a buffer whose elements are all sent in a burst after the sampling session. In this case, the loss of samples can impair the accuracy of the signal reconstruction, and therefore the analysis. Moreover, the volume of data generated requires compression, to reduce the amount of data transmitted and extend lifetime. This poses an additional reliability requirement, as it is impossible to decompress the stream if some of its packets are missing.

II. *Low-rate* data with *weak reliability* requirements, i.e., those coming from environmental and deformation nodes. Even if one sample is occasionally lost, a meaningful data analysis can still be carried out.

Our system also supports best-effort delivery of system data (e.g., battery status) whose loss is not critical. We could design a solution only for the most demanding class I, and use it for all data collected. However, this would constitute a waste of resources. Therefore, we designed a solution able to accommodate each of the above requirements efficiently.

Design and implementation. The sampling of environmental and deformation nodes is straightforward. The only peculiarity of deformation is that a single sample is usually not relevant, as values tend to fluctuate: thus, the data communicated to the sink is actually an average of the last A samples.

Instead, acceleration nodes add significant complexity due to the high volume of data sampled. Each of these nodes buffers the data of an entire sampling session on FRAM. The availability of the entire data set allows us to apply a Huffman [33] compression scheme to reduce

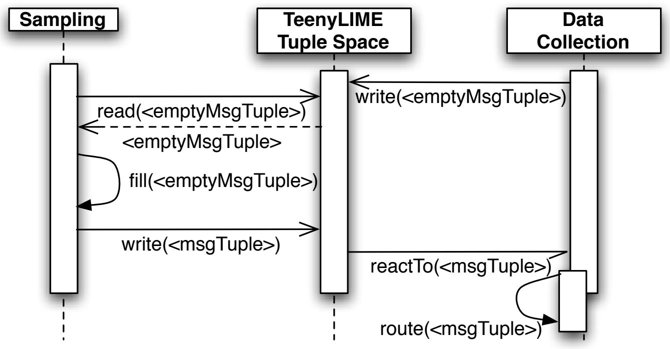


Figure 2.6: Handing sampled data over for routing.

the amount of data transmitted. It is important to note that, unlike other compression schemes mentioned in the literature (e.g., wavelets in Wisden[12]), Huffman is *loss-less* and therefore preserves the semantic richness of the vibration data [48]. The effectiveness of compression, however, greatly depends on the statistical properties of the data set. We observed that different nodes and acceleration axes produce data with different properties, which can be exploited in the Huffman scheme. Therefore, we developed a compilation tool-chain that, using as input the (real) uncompressed data from a node/axis, automatically generates the optimized compression code to be used on that node. This procedure requires an extra step during system deployment, but achieves remarkable improvements in the resulting compression, as discussed in Section 2.6.

At run-time, sampled data is encoded in a tuple that is shared by the sampling component, through TeenyLime, with the data collection component of Figure 2.3. The coordination among the two takes place as shown in Figure 2.6. The sampling component queries the tuple space for an "empty" tuple, indicating the availability of a transmission slot: we describe next how and when this is generated. If such a tuple exists, it is removed from the tuple space, filled with the data to transmit, and output back to the tuple space. Through a previously-installed reaction the data collection component, notified of the presence of the data tuple, can withdraw it and begin the processing necessary for routing.

Our routing protocol builds a tree topology rooted at the sink. The tree is periodically rebuilt to account for connectivity changes. The process is performed by flooding a special control tuple. Each node re-propagates the tuple by writing a copy of it in the tuple space of every node within communication range. There, the appearance of the tuple triggers a previously-installed reaction, which updates the tuple content with path cost information and repeats the process, eventually flooding the entire system. This flooding mechanism is reused also by other components, as mentioned later.

The reliability metric we use in optimizing the shape of the tree is a variant of [84], based on the Link Quality Indicator (LQI) provided by the radio chip. Interestingly, the LQI value

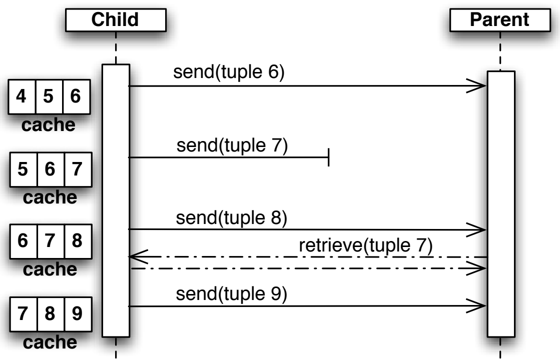


Figure 2.7: Hop-by-hop recovery example.

is also accessed through TeenyLime, using special tuples whose field values are materialized by the run-time, as described in Section 2.4.4. Finally, data forwarding occurs through the tuple space, by writing tuples to the tuple space of the current parent in the tree.

The reliability requirements of the aforementioned class I and II are dealt with through a hop-by-hop recovery scheme, intuitively described in Figure 2.7. Sent tuples are kept in the local tuple space, which effectively serves as a local cache, managed as a circular buffer. The receiving parent in the tree keeps track of the last tuple received from each child, thanks to a sequence number included in it. Upon recognizing a hole in the sequence, the parent pulls the missing tuple from the child's cache, using a read operation. The child node is totally oblivious of recovery: no dedicated processing is required, as the necessary operations are performed directly by the parent through TeenyLime.

Since it is localized, fully distributed, and does not require system-wide flooding of recovery information, our reliable protocol enjoys lower latencies and far less network overhead than end-to-end, centralized solutions such as [36]. On the other hand, it might fail if a tuple is lost right before a node changes its parent. Consider a node C switching its parent from P_{old} to P_{new}. In this situation, P_{new} has no information about tuples previously sent by C, and cannot detect a tuple lost during the switch. These cases do occur in practice: Figure 2.8 shows a lab experiment where the tree is rebuilt every 2.5 minutes, and the occasional tuple losses occur *only* in coincidence with such tree reconfigurations.

Situations like the above must be avoided for class I traffic, which requires 100% delivery. They are taken care of in our protocol with a simple, yet effective, mechanism. Whenever the sink recognizes the beginning of a burst of class I traffic, the time scheduled for the next tree rebuild is temporarily set to infinite. This effectively prevents the tree from changing while class I traffic is routed towards the sink, and thus removes the source of the problem.

Our implementation also considers transmission schedules. Traffic of class II is scheduled opportunistically. In the case of class I traffic, however, a network congestion may develop due to the high volume of data transmitted. To alleviate the problem, we employ a form of slow-

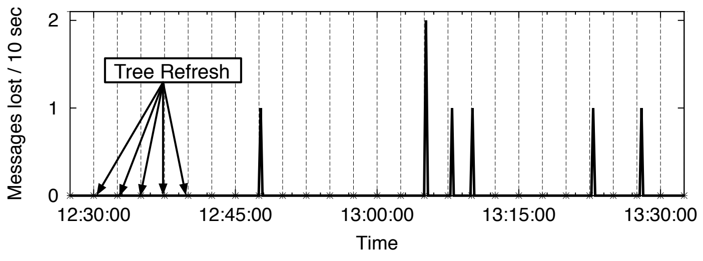


Figure 2.8: Lost tuples and tree refresh operations.

start scheduling for class I traffic, varying the inter-message period at which the empty tuple representing an available transmission slot becomes available. When a transmission failure is detected, the inter-message is set to the highest value then slowly decreased, up to a configured minimum, as data are successfully forwarded to the sink. With a minimum inter-message interval of 1 s, the reporting of a 30-second compressed sampling session at 200 Hz takes around 8 minutes.

2.4.2 Time Synchronization

Requirements and challenges. To investigate the dynamics of Torre Aquila, the readings taken by different nodes must be correlated w.r.t. time. This is especially true for vibrations, e.g., to study how forces applied at the base of the tower propagate to the top floor. The samples must be aligned in time, with a worst-case time drift up to 1 ms [47].

Design and implementation. Several time synchronization protocols for WSN exist. To meet the requirement above, our solution is a modified version of [24]. The protocol works by creating a hierarchy among the network nodes, whose clocks are then synchronized with the root's clock. As depicted in Figure 2.9, synchronization is based on a round-trip tuple exchange between nodes at level i and $i-1$ in the hierarchy. The nodes at level i record the time T_1, at which a synchronization request is issued, and T_4, at which the reply from a node at $i-1$ is received. This reply contains the times T_2 and T_3 at which the node at $i-1$ received the request and replied to it, respectively. These four values enable the nodes at the lower level i to evaluate clock drifts and propagation delays, and adjust consequently their local time w.r.t. nodes closer to (and therefore with a smaller drift from) the root at level 0. Since this process is performed at each hierarchy level, it eventually synchronizes all nodes to the root.

As with the other services, we implemented time synchronization using TeenyLime. The hierarchy is built trivially by relying on the same flooding mechanism described for data collection in Section 2.4.1. However, the information flooded (and therefore the resulting tree) is different, since data collection optimizes the tree shape w.r.t. link quality, while time synchronization minimizes the hop-count from the sink to reduce the impact of the link latency on the time

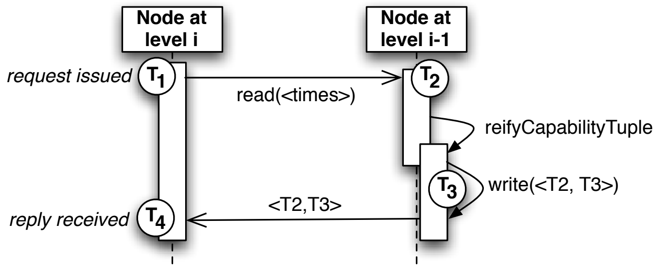


Figure 2.9: Time synchronization using capability tuples.

estimate.

Instead, pairwise synchronization among nodes relies on one of TeenyLIME's unique constructs: *capability tuples* [16]. A capability tuple is essentially a placeholder for the actual data, which is generated on demand. As illustrated in Figure 2.9, when a read operation whose pattern matches a capability tuple is received, TeenyLIME does not simply return, as usual, the latter as result. Instead, it delegates its computation to the component that originally output the capability tuple, using a `reifyCapabilityTuple` event. This is handled by computing and outputting the actual content of the tuple, which is then finally delivered to the query issuer by TeenyLIME. This mechanism essentially enables a node to "advertise" the availability of data without the need to keep it up-to-date by periodically regenerating it—a waste of energy when not used by any query.

In our time synchronization component we use a capability tuple to produce on demand the values of T_2 and T_3, as illustrated in Figure 2.9. It is worth noting that most of the distributed processing is dealt with by TeenyLIME, greatly simplifying the implementation.

Of course, threats to accuracy may come from the unpredictability of processing and message transmission delays. Solving this issue actually led to extensions to the original TeenyLIME API. To alleviate the first problem, we enabled components to be notified when a given operation (e.g., a message send) is completed. This information is used by the synchronization component to periodically re-evaluate processing delays. Message transmission delays, instead, are kept under control by temporarily switching off the radio duty-cycling during a synchronization round. This is achieved by using a newly-designed *tuning interface*, which enables cross-layer interactions by giving developers direct control over the node hardware.

Evaluating precisely the accuracy of our protocol is difficult in the deployment environment. Therefore, we performed a number of lab experiments, using 12 nodes in a chain topology. We used a Tektronix TDS 220 two-channel oscilloscope to measure the time drifts between any two nodes in the network. As expected, the worst-case time drift happens between the root of the tree and the node at the opposite end of the chain. In this case, the time difference was 732 μs, still sufficient to perform meaningful analysis of vibration data [47]. Moreover, Section 2.6 reports that in the deployment we observed at most 6 hops between an acceleration node and the sink. It is therefore unlikely that time drift in Torre Aquila is higher than in our lab experiments.

2.4.3 Tasking and Data Dissemination

Requirements and challenges. The *ideal* configuration of the monitoring system deployed in Torre Aquila, in terms of acquisition rates and intervals, is not known a priori, as often happens when WSNs are employed to study a physical phenomenon for the first time. Moreover, in many cases an external event may suggest a different configuration. For instance, it could be of interest to monitor more frequently vibration and deformation when roadwork is being conducted nearby, people are present in the tower, or strong winds are present. The ability to remotely task the system must be supported by a mechanism that disseminates the new configuration *reliably*, and guarantees that the received data is *eventually consistent* across the system.

Design and implementation. The set of sampling parameters that can be modified remotely are those shown in Figure 2.5. There, we included the values suggested by the structural engineers on our team: each parameter, however, can be changed independently. In particular, the number of sampling sessions N can be a finite number, enabling monitoring of a given quantity only during a given time interval.

A parameter configuration is packed in a *task* tuple with an appropriate format. These tuples are generated on the sink upon a user request, issued through our graphical front-end, and disseminated using the protocol we describe next. On every node, the sampling and tasking component (Figure 2.3) registers a reaction matching task tuples and, upon receipt of a new one, updates the sampling parameters accordingly.

The task tuples must be disseminated reliably throughout the system, a widely studied problem in WSNs [41, 44]. We take inspiration from the state-of-the-art by adapting the Trickle [41] protocol. This achieves eventual consistency of the disseminated data by using monotonically increasing sequence numbers, used to determined if a node is up to date.

This dissemination scheme lends itself to a straightforward implementation on top of TeenyLime. Task tuples are initially flooded by using the mechanism described for data collection in Section 2.4.1. Moreover, the management of missed tuples comes almost for free by using one of TeenyLime's constructs: *neighbor tuples* (or *node tuples* in their original definition) [16]. A neighbor tuple represents the current state of a device, and is made available inside its 1-hop neighborhood. The format of the tuple and the rules for populating its field values are provided by the programmer, but the periodic update of these values and the tuple propagation to neighbors is carried out automatically by the TeenyLime run-time. Therefore, checking whether a recovery is needed in our dissemination protocol is as simple as including the sequence number as a field in a neighbor tuple; installing a reaction that fires whenever a neighbor's sequence number is newer than the local one; and recovering the missing tuple with a read operation on such neighbor.

2.4.4 TeenyLime: Deployment-driven Enhancements

The requirements of the Torre Aquila deployment brought the development of TeenyLime one step ahead. We already mentioned some of the extensions we designed, e.g., the tuning interface in Section 2.4.2. Below is a summary of other enhancements to TeenyLime motivated by our deployment.

Typed tuples and dynamic memory. In the presence of high-rate data such as vibrations, it is imperative to manage efficiently the available memory. To further optimize this aspect in TeenyLime, we introduced a notion of *typed tuple*. Mimicking the generic data types in modern programming languages, developers instantiate tuples as:

```
tuple<uint8_t, uint16_t, float> temperature =
  newTuple(actualField(TEMPERATURE_TYPE),
           actualField(NODE_ID),
           actualField(temperatureReading));
```

where `actualField` indicates a field with actual data, as opposed to constraints on the field type or value. A pre-processor we developed inspects all TeenyLime-based application components to gather a complete view of all tuples used. Based on this, it generates optimized data structures for storing and searching the data.

Typed tuples are managed at run-time by a component providing a form of dynamic memory based on *slabs* [2]. In our case, a slab is a chunk of memory meant to store tuples of the same size. Using slabs does not require de-fragmenting memory, which is difficult to implement on resource-scarce devices. In the application described here, the combination of the techniques above freed 80% of the memory allocated by our previous release of TeenyLime.

Automatic field types. Our data collection component relies on LQI as a measure of link reliability. To relieve the programmer from the burden to explicitly query the operating system for similar low-level information, we make it available in the form of tuples by defining a number of special field types whose value is automatically materialized by TeenyLime as part of the neighbor tuples. For instance, in:

```
NeighborTuple<uint16_t, lqi> myNeighborTuple;
```

the value of the second field of the neighbor tuple reflects the LQI value towards a particular neighbor. This way, low-level data becomes straightforwardly available to the application, greatly simplifying the development of routing protocols.

Reliable, low-power operations. In TeenyLime, programmers can explicitly choose whether the execution of a remote operation is reliable or not. In our deployment, this feature is support by a dedicated reliable communication layer exploiting mixed software/hardware link-layer acknowledgements. This solution occupies only 252 bytes of program memory.

To provide low-power operations, we integrated in our run-time the Low Power Listening [77] layer available in the TinyOS distribution. TeenyLIME's operating parameters (e.g., the timeout for remote queries) are exposed to make them adjustable based on the expected message delays.

2.5 Deployment

The tower contains four floors, the ground one isolated from the others and used as a public walkway. The plan is C-shaped 7.8 m $\times$ 4.5 m, and the height is 25.6 m. The 14th century enlargement closed the tower to the West and raised the gate by an additional storey. The two parts of the masonry body have completely different properties. The lower level walls consist of two 40 cm thick stone blocks, with an incoherent filling. At the upper levels, the older portion of the masonry is built of 80 cm thick stone blocks, while the most recent one is brick and blocks of varying sizes. Visitors enter the tower from the nearby Buonconsiglio Castle, arriving through a long corridor directly on the 2nd floor where the frescoes are.

Node placement. As shown in Figure 2.10, we deployed 16 nodes plus the sink #0. This is placed at the top floor, the only spot guaranteeing access to the external WiFi network.

The sensor position is chosen to detect early symptoms of deterioration of the structure. The joint between the ancient parts of the tower and the more recent ones is today perfectly visible (bottom of Figure 2.10), but the degree of structural connection of this joint is still a

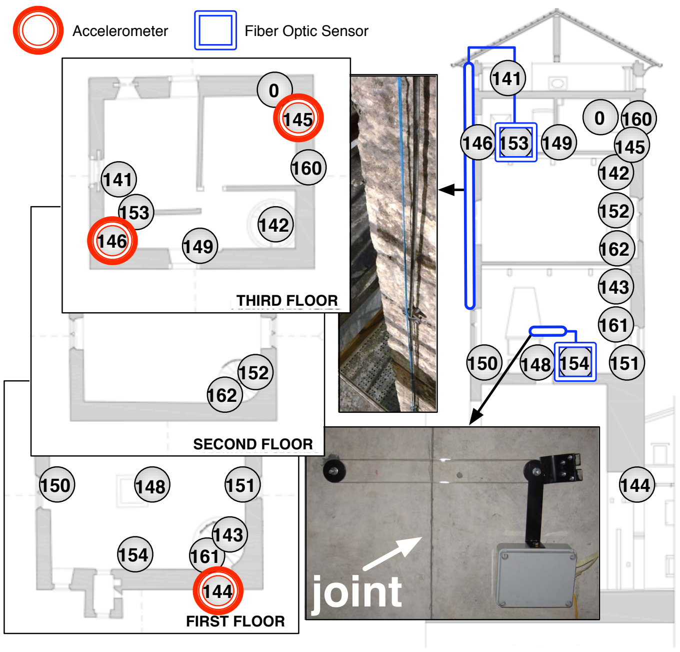


Figure 2.10: Deployment map.

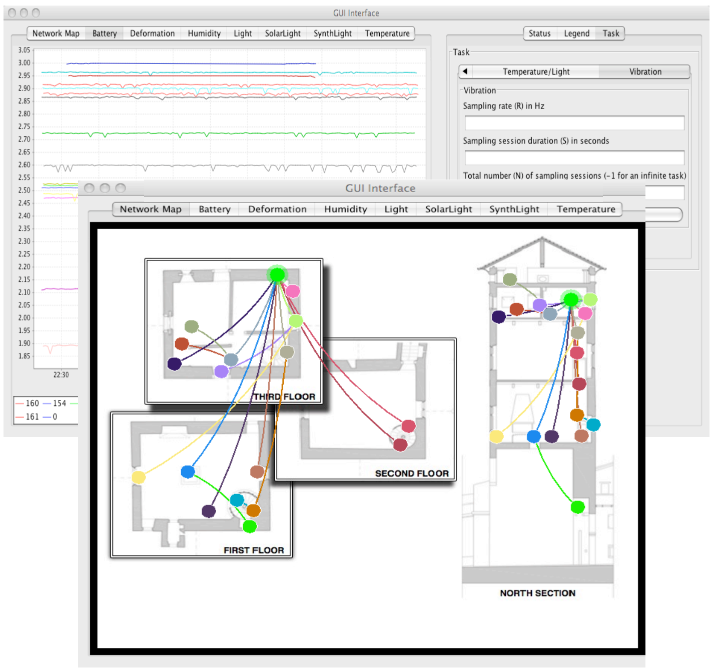


Figure 2.11: Graphical user interface.

major point of uncertainty. The deformation across the connection is measured on the 1st floor by FOS #154. This is a 0.6 m gauge wrapped as an optical coil to magnify the sensor precision, and anchored to two expansion bolts at the sides of the joint, as shown in Figure 10. Another FOS is used to detect vertical elongation at the S-W corner of the tower, from level +5.7 m to +25.6 m. In this case the measuring path is a protected optical fiber loop pre-tensioned between two metal anchorings. An extension cable connects the sensor to node #153 at the 3rd floor.

The vibrations induced by traffic and, to a minor extent, by wind are recorded by acceleration nodes #144, #145, and #146, the first at the base and the others at the top of the tower. The analysis of acceleration readings allows to understand the dynamical behavior of Torre Aquila. Indeed, the vibration response of a building is not completely random, but concentrates mainly around some specific frequencies, know as natural frequencies. Daily and seasonal thermal excursions also affect the structural response of the tower, and the knowledge of these variations is needed to process and compensate the strain and acceleration signals recorded by FOS and accelerometers. This motivates the presence of a number of environmental nodes distributed all over the tower.

Data visualization and access. Effective access to the information gathered by the system is crucial in supporting the structural engineers in their analysis. To this end, we provide a custom graphical user interface, shown in Figure 2.11, implemented through a major re-factoring of Octopus [60]. The GUI shows the current network topology and serves as a control center from

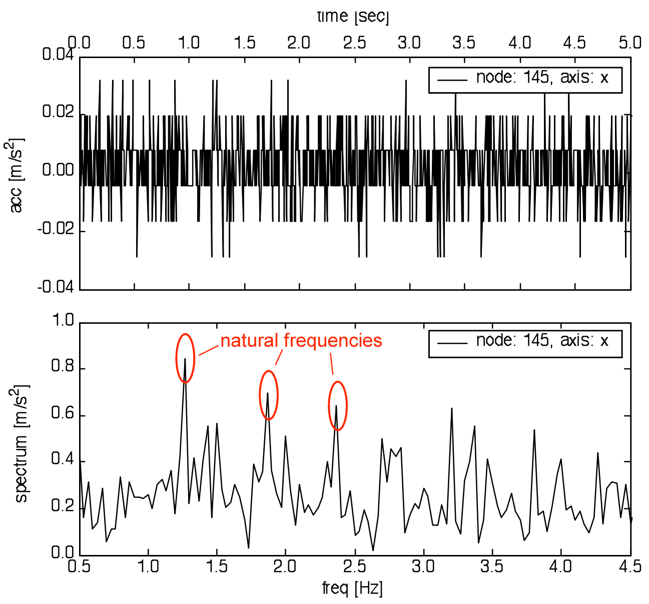


Figure 2.12: The acceleration signal from #145.

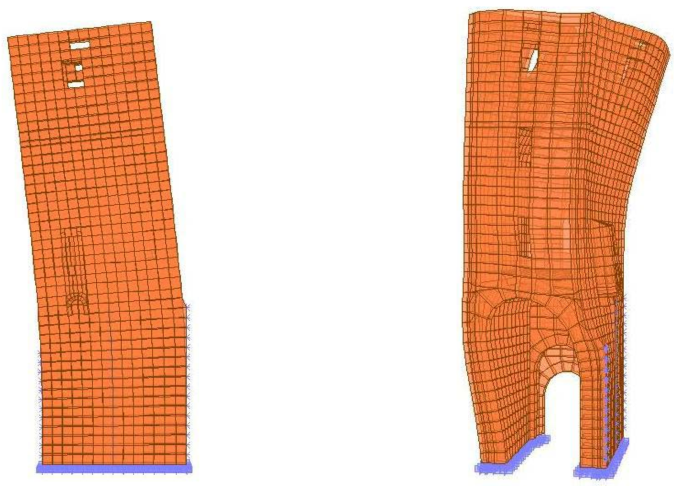
which the user can remotely task the WSN. Moreover, it displays the data collected, which are also persistently stored in a database.

Preliminary data analysis. The data collected is processed by a Bayesian algorithm that provides the user with the real-time probability of an ongoing structural disease. The algorithm can identify a hazardous condition many days in advance w.r.t. to the actual occurrence of the damage [87], and it has already been applied for risk analysis of historic buildings [88]. In the following, we provide a few examples of collected data and discuss the insights that the structural engineers on our team gained from them.

Figure 2.12 shows the acceleration measured on the X axis of #145. The top chart reports the time history over 5 s, while the bottom one shows the corresponding frequency spectrum. The peaks in the spectrum indicate possible natural frequencies of the structure, at 1.25 Hz, 1.80 Hz and 2.40 Hz. Every natural frequency follows a specific deflection shape, usually referred to as vibrational mode. For instance, Figure 2.13 shows the first two vibrational modes of the tower computed by a numerical model, respectively associated to natural frequencies of 1.25 Hz and 1.80 Hz[2].

Figure 2.14 reports the strain measured, in microstrains ($\mu\epsilon$), by the FOS #154 placed across the joint. To eliminate the high frequency instrumental noise, we applied to the signal a moving average filter with a 60-sample long window. The graph shows the well-known "breath" of the structure due to daily thermal variations. The joint is forced to open because of thermal expansion when sun rays hit the southern facade, and then closes during the night. We also

[2]For sake of clarity, in the picture the amplitude of the vibrational modes have been artificially magnified.



(a) (b)

Figure 2.13: First (a) and second (b) vibrational modes.

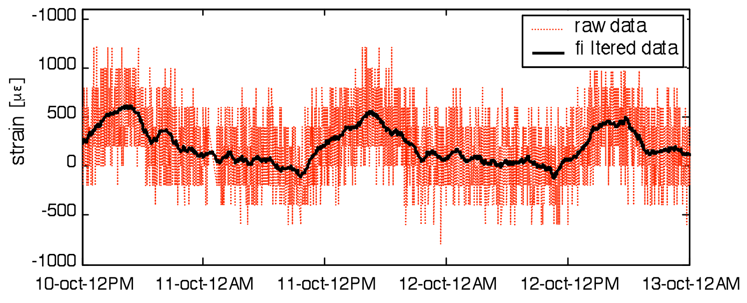


Figure 2.14: Strain measurements from FOS #154.

note a delay between the maximum irradiation at mid-day and the maximum joint elongation, presumably caused by the thermal inertia of the walls. The analysis of this behavior also allows assessing the sensitivity of the joint to temperature. As for the latter, the temperature data shown in Figure 2.15 for nodes on different floors confirm the presence of a gradient along the tower, as well as significant seasonal changes. The daily strain variation (on the order of 500 $\mu\epsilon$) agrees with the numerical prediction under the assumption that the joint is fully released. To date, the strain response of the tower has not shown trends which may rise concerns about its stability.

The benefit of the above analysis is twofold: on one hand, in the short-term it permits identification of a reliable model for the structure response, and prediction of the behavior of the tower during exceptional events, e.g., earthquakes or subsiding. On the other hand, the data are stored in a database that will remain available in the long-term and constantly compared

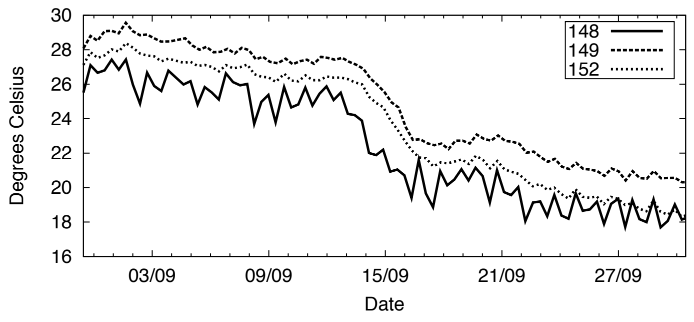


Figure 2.15: Temperature on three floors of Torre Aquila.

with more recent data, so that any change in the tower behavior can be detected, triggering specific analyses.

2.6 Evaluation

In this section, we study the effectiveness of our design along two lines. We report first on the system performance in Torre Aquila, showing that our solution performs reliably and efficiently. Next, we consider the benefits of using a middleware during the development process.

2.6.1 System Performance

To assess the effectiveness of our middleware-based design we report on three key performance issues: i) reliable delivery of data, ii) effective compression of acceleration readings, and iii) energy consumption and system lifetime.

Reliable delivery. During the last four months of operation, the overall loss rate always remained below 0.01%. This performance is striking if compared to the average yield of long-running WSN deployments reported in the current literature [1], and even more so if we consider that ours is one of the few WSN deployments featuring high-rate data reporting for more than

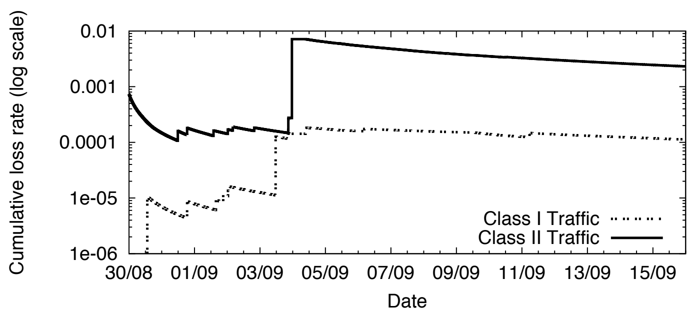


Figure 2.16: Cumulative loss rate.

a few weeks.

The effectiveness of our reliability mechanisms for traffic of class I and II is exemplified in Figure 2.16, showing the cumulative loss rate (in log scale) over time. The loss rate for class I traffic generally remains an order of magnitude lower than that of class II traffic. In the morning of September 3^{rd} a malfunctioning acceleration node lost a number of tuples, which generated the spike relative to class I traffic. Later on the same day we replaced the faulty node and performed a few maintenance operations on the sink, temporarily suspending its operation. This caused the spike in class II traffic. After these two events, the loss rate decreased steadily.

This performance is achieved in spite of the peculiar characteristics of the deployment scenario. Although Torre Aquila is not particularly tall, the thickness of its walls greatly hinders wireless propagation. As an indication of this, Figure 2.17 reports the percentage of time some nodes spent at a given distance from the sink. Notably, the latter in some cases reaches the value of 6 hops. Moreover, we observed how small changes in the node placement drastically change the connectivity. Figure 2.17 shows two periods: in the latter we moved the sink because of some restoration work taking place in the tower. Although the sink was moved at most by 1 m, the topology drastically changed: for instance, #148 became able to reach the sink directly for most of the time, rather than through the 4-5 hops experienced previously.

In any case, our data collection protocol adapts effectively to topology changes. For instance, Figure 2.18 shows how, in the context of the same sink movement, nodes select a new, better

Figure 2.17: Distance in hops from the sink.

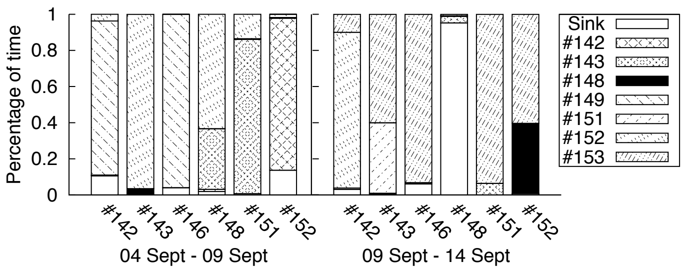


Figure 2.18: Time spent with a given parent node.

parent. However, topology changes are more frequently induced by connectivity fluctuations caused by people visiting the tower and humidity gradients: the reaction to these common causes is equally effective. For instance, we observed nodes relying on up to 4 different parent nodes, according to the observed link reliability.

Compression. We used an Agilent 34411A digit multimeter to measure the processing time over 166 sampling sessions of 30 s at 200 Hz, for a total of ~1,000,000 raw acceleration samples. Although the code was not optimized for this data set, the worst compression time was 17.32 ms, which supports our choice of Huffman coding and confirms the efficiency of the compression code we generate automatically.

Our tool-chain also enables optimization of the compression scheme according to the specific node (i.e., position) and axis. In Torre Aquila, this brings considerable advantages w.r.t. a compression tuned using all acceleration samples regardless of their source and axis, as illustrated in Figure 2.19. Interestingly, the maximum improvement is achieved by generating the custom compression code for the Z axis. Indeed, this axis is subject to the gravitational field, and therefore its values are rather different from those of the X-Y axes: a dedicated compression scheme better captures the statistical properties of the corresponding data sets.

Energy consumption and lifetime. We observed that energy consumption essentially depends on the node functionality, as shown in Figure 2.20 using battery voltage. Acceleration nodes draw more current than environmental ones: not only are they used more intensively, but they must also continuously power the FRAM chip. Consequently, acceleration nodes deplete their available energy more rapidly.

Estimating the expected system lifetime of our system is tricky due to the non-linear behavior of commercially available batteries [61]. The first version of the system used a radio duty-cycle of 100 ms and used the on-board LEDs for debugging. Under these conditions, and using one pair of size C batteries, we observed *one* node dying after 3.2 months of operation. The system is currently operating with a radio duty-cycle of 250 ms, yielding the same reliability. Moreover,

Input Node	**Input Axes**	**Compression Ratio**	**Reduction in Data Traffic**
All	All	17.9%	17.9%
144	All	31.45%	27.7%
145	All	24.91%	
146	All	26.76%	
144	X-Y	47.11%	51.23%
	Z	69.34%	
145	X-Y	41.65%	
	Z	64.66%	
146	X-Y	43.56%	
	Z	62.43%	

Figure 2.19: Compression ratios with different input sets.

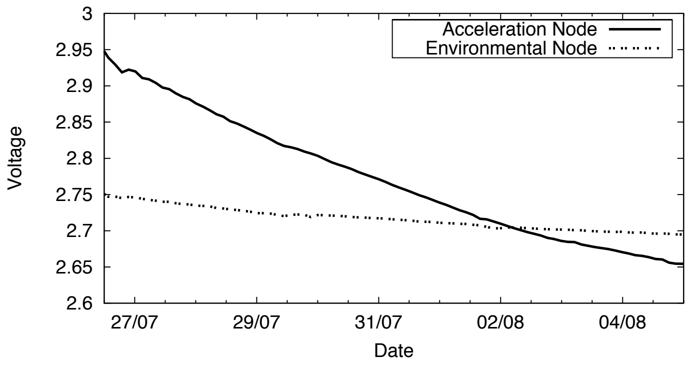


Figure 2.20: Battery voltage readings.

Component	Lines of code
Sampling & Tasking	235–962
Data collection	993
Data dissemination	339
Time synchronization	916

Figure 2.21: Lines of code for our core components.

our packaging can accommodate two pairs of size C batteries. Assuming the single dead node as a worst case, we expect the system lifetime to extend beyond one year.

2.6.2 Beneficial Impact of Middleware

We discuss the impact of our middleware-based design on programming effort and re-usability.

Programming effort. Quantifying the programming effort is hard, as it is affected by factors difficult to measure (e.g., the complexity of the processing). Research in WSNs has hitherto considered the number of lines of code (LOC) as a simple indication. Figure 2.21 reports this metric for the core functionality of our system. It is interesting to compare these figures against similar functionality available in TinyOS libraries, where it is built directly on top of the OS. The CTP [77] collection protocol and the DIP dissemination protocol [44] have almost twice as many LOC as our solutions, and yet the former addresses only low-rate data. The original implementation of the time synchronization protocol [24] contains 80% more LOC than our version. We maintain that the significant reduction in LOC is achieved by delegating part of the processing to the middleware. For example, most of the recovery processing in our data collection component takes place within TeenyLime, as described in Section 2.4.1. Parsing recovery requests, finding the message to be re-sent, and re-trying the transmission are captured by a *single* remote read operation.

Decoupling and re-usability. The use of TeenyLime fosters *asynchronous* and *data-centric* interactions, which increases decoupling. As a result, the design for Torre Aquila can be easily extended to meet different requirements. For example, consider adding distributed data aggregation. This functionality is usually embedded within routing, resulting in the two becoming entangled. Instead, in our design this would require no modification to the data collection component. It is sufficient to tag differently the tuples carrying raw data, and make the new data aggregation component react to them. Aggregated data would then be output as message tuples triggering a reaction in the data collection component, as already happens in our current design. All these changes would not even require a wiring of nesC interfaces.

The high decoupling is also beneficial w.r.t. memory consumption. The size of the binary image installed on our nodes ranges from 37 KB (environmental nodes) to 47 KB (acceleration nodes). The latter is close to the 48 KB limit on TMotes, but it is the most complex as it also includes the compression code. Using components from the TinyOS libraries to provide similar functionality (i.e. CTP, DIP, and the implementation of [24]) would yield a binary of at least 51 KB, which would not fit the program memory.

2.7 Concluding Remarks

The deployment in Torre Aquila demonstrated that a monitoring infrastructure based on WSNs can effectively meet the requirements of, in our specific case, the civil engineers. The sensor heterogeneity defined a unique set of requirements, which we satisfied with a custom design based on TeenyLime. The employed communication abstraction supported the development of system services, simplifying the programming effort and shifting the focus from OS-level details to the exchange of data among components and nodes. Finally, the deployment provided a unique and valuable in-the-field experience, which can hardly be obtained by only looking at the related literature.

Chapter 3

Adaptive Lighting in Road Tunnels

The deployment described in Chapter 2 presented an example of a monitoring infrastructure in which data are delivered to a central base station and afterwards made available for analysis. The defined requirements imposed on the network involved the different data types and their reliable delivery, as typical of a monitor-only system. However, the vision of WSNs places them at the interconnection between the physical and the artificial reality. In such a vision, the WSNs enable not only monitoring, but also controlling of the real world. When the application influences the environment under observation, the loop is closed and new requirements rise to successfully accomplish the control goal.

In this chapter, we describe a control loop in which the light values reported by a monitoring infrastructure are used to actuate lamps inside road tunnels. By meeting the requirements with mainstream WSN solutions, we demonstrated the suitability of the technology in systems required to operate in scenarios where safety is crucial.[1]

3.1 Scenario, Motivation and Contribution

The system described in this chapter has been developed in TRITon (Trentino Research & Innovation for Tunnel Monitoring, `triton.disi.unitn.it`), a project carried out by research centers and companies, funded by the local administration in Trento, Italy, with the goal of reducing the management costs of road tunnels and improving their safety. Our WSN-based control system is to be installed in *operational* tunnels on a high-traffic freeway—an ambitious goal given that WSNs have never been used in this context.

[1]The content of this chapter is a joint work with Michele Corrà, Leandro D'Orazio, Roberto Doriguzzi, Daniele Facchin, Ştefan Gună, Gian Paolo Jesi, Renato Lo Cigno, Luca Mottola, Amy L. Murphy, Massimo Pescalli, Gian Pietro Picco, Denis Pregnolato, Carloalberto Torghele, published in "Is There Light at the Ends of the Tunnel? Wireless Sensor Networks for Adaptive Lighting in Road Tunnels", 10^{th} ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN'11, SPOTS track), Chicago (IL, USA), April 2011 (Best Paper Award) [8].

Adaptive lighting in road tunnels. In state-of-the-art solutions, tunnel lighting is either pre-set based on current date and time, or set by an open-loop regulator relying on an external sensor. Both solutions disregard the actual lighting conditions inside the tunnel, and may endanger drivers or consume more power than needed.

In the system we describe here, a WSN deployed along tunnel walls measures the light intensity and reports it to a controller, which closes the loop by setting the lamps to match the lighting levels mandated by law. Unlike conventional solutions, our system adapts to fine-grained light variations, both in space and time, and dynamically and optimally maintains the legislated light levels. This enables energy savings at the tunnel extremities, where sunlight enters, but it is also useful inside the tunnel to ensure the target light levels even when lamps burn out or are obscured by dirt. We detail further the adaptive lighting problem in Section 3.3.

Motivation for WSNs. We are sometimes asked: *"Why should one use a WSN in tunnels, where power and network cables are already available?"* Although power cables are present along with lighting, realizing the shunts necessary to operate the distributed sensor nodes at the right voltage is expensive. Similar considerations hold for network cables, actually found only in medium and long tunnels. Finally, the untethered WSN nodes can be placed *anywhere* along the tunnel—i.e., where lighting engineers say it is best to sense light—and not only where cables already exist.

A WSN solution drastically reduces installation and maintenance costs, especially when the target is an *already-existing* tunnel, where changes to the infrastructure should be minimized. This is often the case in Trentino, the province managed by the administration funding TRITon, a mountainous area of 6.200 km^2, 500,000 people, and over 150 tunnels for a total of 50 km, the majority of which are old and under 500 m. In these tunnels, a small investment can significantly improve safety and reduce energy bills.

Challenges. As we discuss in Section 3.4, tunnels are harsh environments, relatively well-studied but for which real-world WSN experiences are largely missing. In our case matters are complicated by vehicular traffic, which affects wireless communication, and light itself, which is notoriously difficult to measure accurately and yet whose (abrupt) variations are the essence of our application. These challenges notwithstanding, the practical goal of TRITon is to deploy a WSN-based adaptive lighting system in a 630 m, two-lane, double-carriageway operational tunnel with an average traffic of more than 27,000 vehicles per day. The design decisions for the WSN supporting closed-loop control in such a safety-critical environment are dominated by real-world constraints, including:

I. extended lifetime is paramount: changing batteries can be easily performed during tunnel maintenance, but tunnel operators expect *at least* a 1-year lifetime;

II. continuous operation implies that the WSN cannot fail: node failures are important, but sink failures are critical;

III. sensed data must arrive timely: we do not face hard real-time constraints, yet delays induced by node and communication failures may jeopardize control;

IV. the quality of sensing impacts directly the quality of control: sensor accuracy and noise reduction are key;

V. integration with conventional, industrial-strength equipment poses complex engineering challenges.

Contribution. We deliberately choose to tackle the challenges above by reusing existing techniques whenever possible, as the target scenario already entails several complex engineering and deployment issues. However, the staple WSN mechanisms and protocols in monitoring-only deployments have essentially never been tested in such a challenging setting, also including closed-loop control. Bearing this in mind, our contribution lies precisely in:

I. verifying that a WSN-based solution to adaptive lighting is *feasible* in road tunnels;

II. understanding to what extent the solution can be achieved by relying on *mainstream* WSN technology;

III. identifying a *combination* of techniques, among the many reported in the literature, successful in our peculiar setting;

IV. demonstrating the above in an *operational* testbed where the WSN is integrated with standard tunnel equipment.

We also believe that gaining practical insights into the aspects above reaches beyond the specifics of our road tunnel scenario. Some of the requirements we are forced to cope with are akin to related scenarios where the use of WSNs is envisioned but only partly accomplished; for example, metropolitan subways [10], underground mines [42], and service pipes [75]. The real-world lessons we learned may be an asset for the designers of these systems.

Section 3.5 illustrates the system architecture by concisely describing each functional component. The focus of our work is however on the WSN one. Section 3.6 describes how we tackled the aforementioned challenges by relying on a popular platform: TelosB-like motes running TinyOS. The motes host custom-made sensor boards we calibrated for our tunnel setting. The software deployed on the motes includes dedicated communication protocols, whose design however relies on the combination of well-known techniques. A distinguishing aspect of our software layer is that both application and system-level services (e.g., routing) are built atop middleware [16] that, compared to using directly the operating system as in the vast majority of reported WSN deployments, greatly reduces the programming effort and yields a smaller binary footprint.

The high volume of vehicular traffic in our final tunnel deployment prevents us from using it for experimenting with parameters and performing validation tests. Therefore, in this chapter we report on results in a second tunnel that, although operational, is less trafficked and offers a more flexible experimental testbed to analyze and tune our system, which must work right away upon installation in the final tunnel. The equipment we installed is described in Section 3.7. The testbed experiments, over a 7-month period, are reported in Section 3.8, where we analyze both the quality of control and the WSN performance. Results show that our system accurately closes the control loop even in the presence of noisy and inappropriate lighting equipment. Moreover, they confirm that the WSN meets the above challenges by guaranteeing a 99.98% data yield, a reporting delay compatible with the operation of the control system, and an (under-)estimated lifetime well beyond a year.

Section 3.9 concisely reports on experiments hinting at the fact that the WSN we designed for adaptive lighting can be reused effectively to detect fire, with only very minor modifications.

3.2 Related Work

The literature related to this work concerns the use of wireless technology, including WSNs, in tunnels and similar environments, and the design of closed-loop control systems relying on WSNs.

Wireless technology in tunnels. The behavior of wireless transmissions in tunnels and similar environments has been studied extensively, e.g., for what concerns path loss [76] and radio propagation [55]. Existing works show that the shape of tunnels determines an "oversized waveguide" effect [55]. As for WSNs, we discussed our own experience with the wireless topology of two tunnel deployments in comparison with a vineyard one in [57]. Section 3.4 summarizes some of the main findings.

Existing WSN applications in road tunnels focus on monitoring for emergency services [6] and disaster management [15]. These, however, are sophisticated proof-of-concept systems, not designed to sustain long-term operation like the one we present here. WSNs have also been applied in tunnel-like environments, including subways [10], coal mines [42], and service pipes [75]. However, none of these systems involves closed-loop control, and integration with existing, industrial-strength infrastructure is usually not an issue. These are instead some of the characterizing features of our work.

WSN-based closed-loop control systems. Few WSN experiences involve closed-loop control. Lynch et al. [49] integrate a WSN with a semi-active damper to mitigate the structural response of civil infrastructures during earthquakes and similar phenomena. Singhvi et al. [73] rely on mobile nodes to acquire information on the users' behavior and context, to perform adaptive lighting in buildings. Both works focus almost exclusively on the design and optimization of the control algorithms. In contrast, the safety concerns and practical deployment issues concerned with an operational setting play a fundamental role in our work. Kim et al. [37] deploy five wireless sensing stations to perform feedback-driven site-specific irrigation. Their setup is much simpler than ours: each sensing station enjoys permanent power, communicates directly with the base station, and is mapped to a single actuator. Han et al. [30] rely on a WSN to drive the operation of a numerical simulator for plume detection and movement prediction. However, unlike our system, the control loop is entirely within the software realm, and does not affect the physical environment. Finally, Park et al. [62] report on a WSN design for closed-loop light control for entertainment and media production. While the goal of their system is somewhat more sophisticated than ours, their implementation is limited to a small-scale lab proof-of-concept, which therefore is not confronted with the complexity and engineering challenges of a *long-term, operational* system in a *real-world* environment, which is instead one of the defining features of the work reported in this chapter.

3.3 Problem and Approach

Designing an appropriate lighting for roads is challenging, as it directly affects safety and requires huge amounts of energy. Tunnels inherit these challenges and pose additional ones. Illumination varies significantly along a tunnel's length, unlike on roads, and requires a more sophisticated control in response to environmental conditions. Moreover, and most importantly, the light conditions at the entrance must match closely the external ones to ensure that drivers can still discern obstacles when entering the tunnel.

Satisfying this latter requirement during daytime hours has a huge impact on energy consumption. Indeed, daylight is several orders of magnitude larger than that sufficient for night vision, due to the ability of the human eye to adapt to darkness. To get a concrete feel of the values at stake, solar light may reach in excess of 100,000 lx while night road illumination is usually 5-10 lx. Therefore, the initial few meters of a road tunnel can easily consume in daytime hours the equivalent of kilometers of road lighting at night. On a broader perspective, the 150+ road tunnels in Trentino consume 20 GWh per year, as much as 16,000 people in the same region. Therefore, even a small improvement of the tunnel lighting system can return significant savings on the energy bills.

Tunnel lighting must abide by an illumination curve defined by law [13], that specifies the light level as a function of the distance inside the tunnel, as shown in Figure 3.1. At the entrance, the curve aims at ensuring continuity of light conditions from the outside to the inside, to avoid that drivers perceive the tunnel as too bright or too dark. As the distance from the entrance increases, the light level is allowed to decrease, as the human eye adapts to darkness.

Conventional solutions. The legislated curve is currently met by simple solutions that over-approximate the safe light levels, therefore wasting energy. The most common solution is a simple timer, that automatically sets the light intensity along the tunnel based on date and time, entirely oblivious of the conditions inside or outside the tunnel. More sophisticated solutions employ an open-loop regulator relying on an external sensor: the light setting is based on the outside conditions, but the inside ones are still disregarded.

The desired illumination levels are achieved by relying on two separate circuits. The first one (*permanent lighting*) guarantees a constant illumination and is always on. The second circuit (*reinforcement lighting*) provides the extra light necessary to match the daytime external light, and is therefore normally switched off at night. As lamps are typically set in groups, the curve generated by the reinforcement has a step-wise shape. Each step is typically set well above the target legislated curve—it is not uncommon to see designs that over-approximate by a factor of two. Given that pre-set and open-loop solutions lack information from inside the tunnel, this conservative choice ensures a safety margin accommodating the aging of lamps (which reduces their light) and other problems such as burned out lamps. However, it clearly induces a waste of energy, shown in Figure 3.1 as the area between the step-wise curve used in conventional systems and the one mandated by law.

Closed-loop adaptive control. The figure shows a third line, representing our closed-loop adaptive control where light measurements inside the tunnel are used as feedback to tune the intensity of the lamps, which are *individually* controlled. The knowledge of the actual conditions *inside* the tunnel is the key to *dynamically* match the legislated curve without unnecessary, costly over-provisioning. This knowledge allows us to reconcile the goals of high safety and low energy consumption, unlike pre-set or open-loop solutions. Moreover, knowledge of the inside conditions enables us to leverage natural light to reduce consumption at the entrance—the largest energy drain. Indeed, sun rays entering the tunnel may contribute enough light to push further inside the point at which the artificial lighting becomes necessary. To achieve optimal, dynamic control of the tunnel lighting three "components" are required:

I. An external sensor measuring the *veil luminance*, i.e., the contrast between the tunnel entrance and its background. This parameter is used by regulations to define when a driver can be negatively affected (e.g., dazzled) by the tunnel lighting.
II. A grid of light measurements along the tunnel length, used to compute the error between

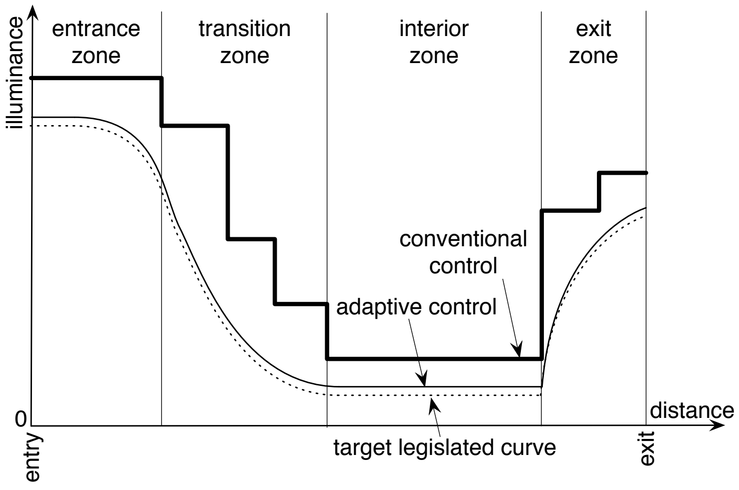


Figure 3.1: Conventional vs. adaptive control.

the legislated target curve (determined as a function of the input veil luminance) and the actual lighting conditions in the tunnel.

III. A control algorithm to drive the above error to zero.

The first component recently became available on the market. The internal measurement system is the main contribution of this work, reporting on a WSN-based solution. Finally, the design of a control algorithm for adaptive lighting is complicated by the high number of individually-controlled lamps and the mutual influence between these and the sensors. Before presenting our system architecture, however, we describe the characteristics of our scenario.

3.4 Peculiarities of Tunnels

Road tunnels are largely unexplored by WSN deployments. Tunnels are harsh environments, where dirt and dust accumulate rapidly and therefore affect sensing, as we discuss in Section 3.6.2. Periodic tunnel cleaning constitutes an additional threat for the nodes, as it is often performed using high-pressure jets of aggressive detergents. The node packaging is therefore of paramount importance, as it must also meet the general tunnel regulations, e.g., concerning resistance to fire. Vehicular traffic further complicates matters, as the metallic vehicles create interference with the WSN radios, and create occlusions and noise to the light sensors. Moreover, traffic limits access to the tunnel for deployment and debugging purposes, as each visit requires blocking one lane, if not the entire tunnel.

We touch on some of these issues in the rest of the chapter. Hereafter, instead, we focus on two aspects that are key to understand our contribution: the characteristics of light in a tunnel environment and how the tunnel shape affects wireless communication.

Light variations. Light is a physical quantity whose precise measurement is already very difficult per se. Tunnels introduce an additional complication, as the light levels vary greatly along its length. Figure 3.2 shows some of the high-rate (5 s) light measurements we collected for one of the tunnels in TRITon, to understand the light variations and properly design the on-board management of the sensed data. Distance from the entrance determines how much the external sunlight affects the reading, with clouds and direct sunlight contributing to the largest daily variations. For example, on the second day shown in the figure, direct sunlight entered the tunnel at sunset, causing readings to go off the scale of this chart. Deeper inside the tunnel, the artificial lights have instead the most influence. In Figure 3.2 one can clearly see the steps caused by changes in the light levels set by the (conventional) control system. Moreover, vehicle headlamps produce transient high readings (barely visible in the night portions of the chart) while trucks occlude sensors and cause the dips visible in the figure. The sensor uncertainty, combined with these phenomena, suggests on-node processing for properly filtering and compensating the data before relaying them to the control system, as we discuss in Section 3.6.3.

Communication. As discussed in [57], tunnels enjoy better connectivity (i.e., longer links) than outdoor, due to the waveguide effect. This solves and creates problems: better connectivity improves robustness, but also increases the probability of packet collisions. Moreover, the network links are more stable in tunnels w.r.t. what is typically reported in the literature for more conventional environments, therefore impacting the relative performance of link estimators. Even in the presence of vehicular traffic, both intermediate and high quality links are accurately identified, and there is a stronger linear correspondence between LQI and packet error rate. As a consequence, LQI performs in tunnels similarly to popular choices such as ETX [17]. We confirmed these findings in the tunnels described here and therefore, as discussed in Section 3.6, our routing solution relies on LQI.

3.5 System Architecture

The functional components of our closed-loop solution for adaptive lighting are shown in Figure 3.3. The system relies on the WSN for acquiring dense light measurements in the tunnel and wirelessly relaying them in multi-hop to a gateway. Multiple gateways are deployed, to reduce the network diameter and provide redundancy against gateway failures. The gateways forward the sensed data to a Programmable Logic Controller (PLC), the "brain" of the control system. The PLC takes as input the value of the veil luminance measured by an external sensor, along with the data from the WSN. The former is used to determine the target reference lighting, while the latter is used to measure the error from the reference. The PLC directly actuates the lamps to reduce the error and meet the legislated lighting curve. The PLC has access to all the equipment in the tunnel, and can be supervised through the Supervisory Control And

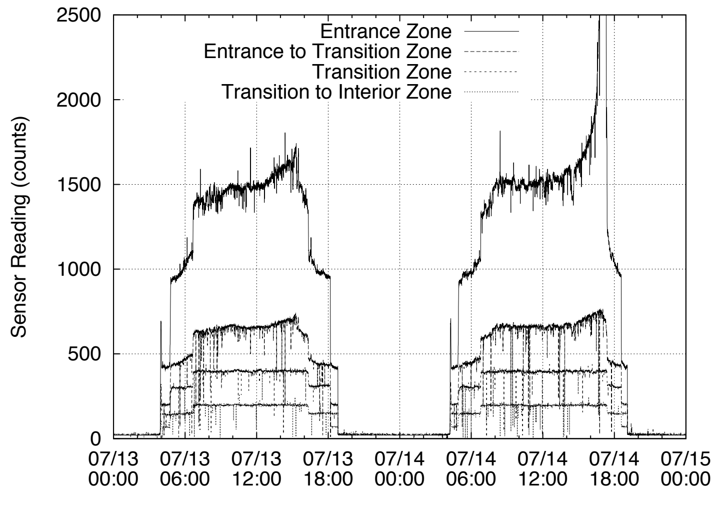


Figure 3.2: Light levels inside the tunnel over two days.

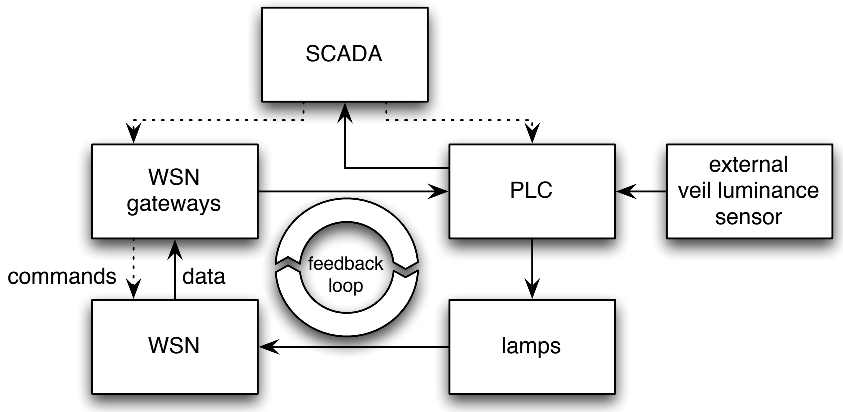


Figure 3.3: Functional components of the architecture.

Data Acquisition (SCADA) component. PLC, gateways, and SCADA are interconnected by a standard industrial Ethernet LAN, running a firewalled TCP/IP suite. The lamps and the veil luminance sensor are instead connected as peripherals of the PLC.

Our system is designed with fault-tolerance in mind. We already mentioned that multiple gateways provide redundancy in the WSN. In the unlikely case that all gateways fail, the PLC switches to an open-loop control relying solely on the external sensor. If this fails too, the PLC defaults to a pre-set lighting curve guaranteeing safety.

As this work focuses on the WSN component, the corresponding hardware and software architecture is described separately and in more detail in Section 3.6. The rest of this section illustrates the remaining components to the extent necessary for this chapter.

Veil luminance external sensor. The regulations determine the lighting inside the tunnel based on the veil luminance at the entrance. The latter requires a dedicated external sensor, in our case a device by Reverberi Enetec designed specifically to operate in road tunnels. The device is based on a camera-like 1.3 Mpixel CCD sensor, whose output is sent to the device's CPU. The veil luminance value is computed according to regulations [13] and output as an analog 4-20 mA line signal delivered to the PLC, described next.

PLC and control logic. In harsh environments like tunnels, the control functionality is usually implemented by means of a PLC. Its computation is cycle-based: tasks are executed within each cycle, based on the timing requirements of the control algorithm. The hardware and computing power of the PLC depends on the complexity of the tasks and on the number of I/O variables that the PLC must acquire and control. We use a Siemens SIMATIC S7-400H with redundant CPUs, equipped with the appropriate peripherals.

The control logic of the PLC is complicated by the following:

- the number of lamps is large (even hundreds depending on the tunnel) and each must be controlled independently;

- the number of measurement points is also large, to enable a dense-enough sampling of illumination;
- a sensor is affected by many lamps and a lamp affects many sensors, requiring the computation of a complex transfer function from each lamp to each sensor.

This scenario defines a multi-in, multi-out control problem that is highly under-determined, i.e., with fewer measured inputs than controlled variables. Although the control logic is not the focus of this chapter, we briefly summarize the problem to the extent allowed by space limitations, to properly place our contribution in context.

Let $\boldsymbol{\Phi} = [\phi_1, \ldots, \phi_L]$ be the vector of light flows from each lamp, $\boldsymbol{M} = [m_1, \ldots, m_M]$ the sensed light measurements, and

$$\boldsymbol{H} = \begin{bmatrix} h_{11} & h_{12} & \cdots & h_{1L} \\ h_{21} & h_{22} & \cdots & h_{2L} \\ \vdots & \vdots & \ddots & \vdots \\ h_{M1} & h_{M2} & \cdots & h_{ML} \end{bmatrix}$$

the transfer function of the light intensity from each lamp to each sensor. We can define $\boldsymbol{M} = \boldsymbol{H}\boldsymbol{\Phi}^T + \boldsymbol{N}$ as the set of measured light samples, where $\boldsymbol{N}$ is a vector of additive noise samples affecting the sensors' measurements, and $\boldsymbol{R} = \boldsymbol{H}\boldsymbol{\Phi_0^T}$ as the target reference working point, computed based on the external sensor and the tunnel lighting standards. The difference between the two, $\boldsymbol{\Delta} = \boldsymbol{M} - \boldsymbol{R}$, represents the error between the target lighting and the one actually measured. The control problem consists of identifying $\boldsymbol{\Phi_0}$ and actuating the lamps to obtain it. Due to the noise term $\boldsymbol{N}$, the direct and exact computation of $\boldsymbol{\Phi_0}$ is not possible, and we resort to minimizing the mean square error

$$\boldsymbol{\Phi_0} = \underset{\boldsymbol{\Phi}}{\text{argmax}}(E[||\boldsymbol{\Delta}||^2])$$

where $||$ is the standard Euclidean norm.

The minimization problem above is a convex hull by construction, since all coefficients in $\boldsymbol{H}$ are non-negative and $\boldsymbol{\Phi}$ is strictly positive. The solution can thus be obtained by employing either a Least Square Error (LSE) or Recursive Square Error (RSE) technique [45]. The complexity of LSE is $O(L)$ in the number L of lamps, and leads to a simpler implementation, potentially allowing us to use a cheaper PLC. On the other hand, RSE is $O(L^3)$ but its convergence is faster thanks to the presence of "memory" in the form of an integral component in the control loop. The problem, however, is further constrained by the fact that the light intensity of lamps cannot be set arbitrarily high, and is bound to a maximum value which depends on the lamp characteristics and technology. We are currently evaluating through in-field experiments in our testbed which approach provides the best trade-off among performance, noise sensitivity, and implementation complexity.

Lamps and actuators. The lighting of our final tunnel includes LED (Light Emitting Diode) and HPS (High Pressure Sodium) lamps. The latter, recognizable by their yellow light, are commonly used in road lighting due to their high emission and relatively low consumption. They can be controlled only within 30% to 100% of their illumination range, and changing their intensity takes minutes. LED technology appeared only recently in tunnel lighting. Its white light enables better vision, the lamps have much lower energy consumption, and can be controlled over their entire illumination range almost instantaneously. Our LED lamps have been developed specifically for TRITon. However, in the testbed deployment described in Section 3.7 we only had HPS lamps installed.

Lamps are controlled *individually* by the PLC through a digital bus, which enables setting illumination precisely at the level required. Instead, conventional solutions control large sets of lamps at once, yielding constant illumination over long sectors of the tunnel with the consequent energy waste mentioned in Section 3.3.

SCADA. The overall system is completed by a SCADA subsystem connected to the PLC. The SCADA provides an interface to a human operator e.g., to visualize alarms, manually force light settings, and perform other configuration and management tasks remotely. The SCADA also logs all the data coming from the PLC, for statistical as well as legal reasons. In TRITon, we customized the SCADA to be able to access directly the gateway and therefore the WSN, e.g., to collect data and status from the sensor nodes, as well as change configuration parameters such as the sampling rate.

3.6 WSN Architecture

We illustrate the WSN hardware, the calibration of sensors, and the architecture of application and communication protocols.

3.6.1 Hardware

WSN node and sensors. We used WSN nodes functionally equivalent to TelosB motes [66], equipped with an MSP430 microcontroller, a Chipcon 2420 radio chip, and an on-board inverted-F microstrip antenna. We did not use an external antenna since the on-board already performs well in tunnels, as mentioned in Section 3.4.

Our nodes are however more easily expandable than TelosB, thanks to three external connectors that simplify the integration of sensor modules as expansion boards. This proved useful in our case, where we custom-designed an expansion board for the needs of TRITon. The board contains 4 ISL29004 digital light (illuminance) sensors, and 1 TC1047A temperature sensor. As shown in Figure 3.4, the board is mounted orthogonally to the node, in a “butterfly” configuration. The 4 light sensors are spread in pairs across the horizontal dimension of the board. As discussed in Section 3.6.3, the reading reported to the gateway is a (filtered) average of

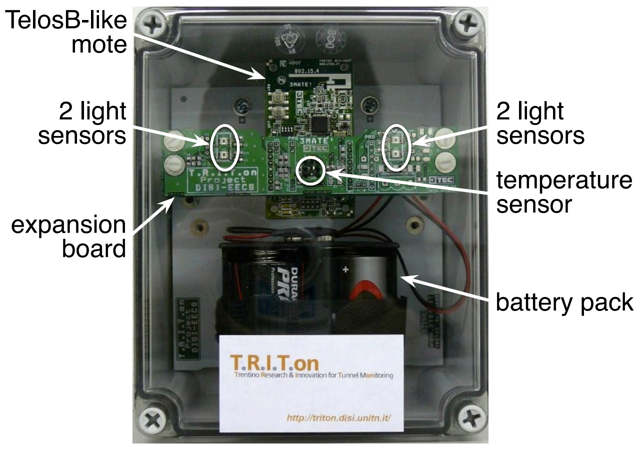


Figure 3.4: A deployment-ready WSN node.

the 4 sensor readings. The temperature sensor is not required by light control, but it is useful to analyze the WSN behavior, especially w.r.t. battery discharge phenomena, as we discuss in Section 3.8.2.

Node and sensor board are powered by 4 Duracell Procell D-size batteries. All the above is packaged in a certified IP65 (water and fireproof) polycarbonate box with a transparent cover.

Gateways. We used a Verdex-Pro embedded computer by Gumstix, equipped with two expansion boards providing Ethernet, USB, and RS232 ports. The former is used to ensure connectivity with PLC and SCADA, while the communication between gateway and WSN sink occurs through the on-board USB port. The RS232 ports are used for debug purposes only. The operating system (Embedded Linux) and applications are stored on the 32 MB flash memory and use the 128 MB of RAM. A 1 GB microSD card provides additional storage. This hosts the database logging WSN light samples, which can be queried directly by the SCADA, and is also useful for debugging the WSN. The database size depends on the number of nodes: for a 20-node WSN it can easily reach 100 MB in a week.

3.6.2 Calibration of the Light Sensors

We must ensure that the off-the-shelf light sensors are accurate and precise enough in our tunnels. We verified that their response is linear, reducing the calibration task to determining the "right" coefficient α converting from the raw sensor (count) readings to lux. However, the calibration is still complex because *i)* light measurements suffer from a non-negligible, intrinsic uncertainty—at least $\pm 10\%$ in real-world, non-controlled environments—and *ii)* the calibration factor α strongly depends on the light source employed.

Our laboratory setup included professional equipment (e.g., mechanical, high-precision positioners) to guarantee uniformity of the light source (an HPS lamp) and fine control over the light emission. The setup is shown in Figure 3.5. As a reference gauge we used an ILT1400A radiome-

	Attenuation
Clean cover	9,3%
Clean cover (45°)	10,8%
Dirty cover (A)	30,7%
Dirty cover (B)	33,9%

Table 3.1: Impact of cover and dirt on light readings.

ter along with its companion SCL110 illuminance probe. Both products are NIST-complaint.

We based the calibration measures on 10 sensors, randomly selected. The measured response included the entire sensor board circuitry, as the integration of the ILS29004 sensor may affect its response. The measurements yielded a value $\alpha = 0.596$, with a determination factor $R^2 = 0.9986$. We estimated the uncertainty at the end of calibration as $s = \pm\sqrt{(2\sigma)^2 + s_g^2 + s_{NIST}^2}$ where $\sigma = 3.2\%$ is the standard deviation of the uncertainty between the measures from sensor and reference gauge, $s_g = \pm 4,4\%$ is the uncertainty of the reference gauge, and $s_{NIST} = \pm 0,5\%$ is the intrinsic uncertainty of the gauge w.r.t. NIST standards. Our measurements yield a total uncertainty $s = \pm 7,8\%$.

Impact of cover and dirt. The measurements above were carried out by exposing the sensor directly to the light source. However, once deployed, the sensors receive light through the transparent cover of the package described in Section 3.6.1. Dust and other agents quickly deposit a dirt film over it, especially at the entrances where nodes are easily splashed by water and mud.

Determining how these factors affect light readings is key to ensuring that adaptive control relies on the *actual* tunnel conditions. We performed experiments with clean and dirty covers, comparing the results with the previous ones. We used dirty covers from one of our testbeds on a high-traffic road, after an entire winter with heavy rain and snow. The results are shown in Table 3.1. A clean, transparent cover is enough to cause a 10% attenuation. The incidence angle does not affect this value significantly. Dirt induces an additional 20-30% attenuation: Table

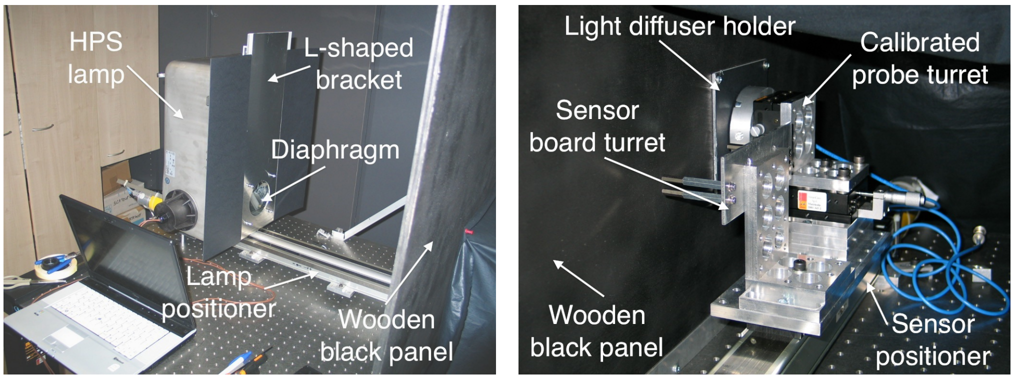


Figure 3.5: Setup for the calibration of light sensors.

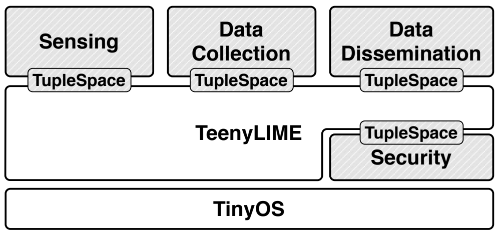


Figure 3.6: Software architecture.

3.1 shows two of the dirtiest covers. Although tunnels are periodically cleaned, we are studying ways to prevent or mitigate attenuation, including online calibration and repellent coating.

3.6.3 Software and Communication Protocols

The software and communication protocols of our WSN rely on TinyOS [31]. However, unlike the vast majority of deployments where application and system software sit directly on the operating system, we built the TRITon software on top of the TeenyLIME [16, 52] middleware. Our choice was motivated by our successful experience in Torre Aquila, described in Chapter 2, and the possibility to verify the effectual ability to reuse the components developed in such a context.

Overview. Figure 3.6 illustrates the architecture of the WSN software. Similarly to the design described in 2.4, all macro-components in the TRITon application sit atop TeenyLIME, which offers a 1-hop shared memory space abstraction in the form of a tuple space, i.e. a collection of typed sequences of fields, on which components can insert, query and receive notifications regarding the data. TeenyLIME both replaces the default message passing communication constructs provided by TinyOS, and bestows upon the architecture a flat layout. Indeed, both application (e.g., sensing) and system components (e.g., data collection) lie on the same level and interact exclusively through the tuple space. We next offer details on these components.

Sensing. The control algorithm requires input values every 30 s from nodes in the entrance, transition, and exit zones, where solar light has a higher influence, and every 5 min from the nodes in the interior zone. During these intervals, the sensing component collects samples from all 4 light sensors at a configurable rate, set to 5 s by default. As described in Section 3.4, vehicles produce transient noise—abnormally low or high readings caused by big vehicles and headlights, respectively—that must be filtered out. These outliers are eliminated as follows. Each time a sample is taken, the value of the 4 sensors is averaged into s_i, by excluding the saturating ones, if any. When reporting to the sink, the average s_{all} of all the values s_i is computed. For each s_i, if the difference $|s_{all} - s_i|$ differs from s_{all} by more than 50%, s_i is discarded and s_{all} recomputed.

Data collection. Single-sink collection trees, the most common solution in the literature, are less effective in tunnels. The tunnel linear shape yields a larger network diameter: nodes closer to the root bear a heavier load, funneling information from a large portion of the WSN. Moreover, the larger number of hops increases the probability of data loss between the leaves and the root. Reliability is complicated further by the permanent asymmetries present in tunnel links [57], reducing the effectiveness of the link-layer acknowledgments commonly used to ensure successful transmission.

To provide load balancing and mitigate the risk of message loss on long, multi-hop paths, we adopted a solution collecting data at multiple sinks, ideally spread evenly along the tunnel. Each sink periodically and independently builds a collection tree by flooding a control tuple containing path cost information. The latter is determined by aggregating the per-hop LQI, similar to MultihopLQI [79], as this technique in tunnels produces overlays similar to those obtained with ETX-based protocols but with much less overhead [57]. Sink selection occurs implicitly, as in CTP [26], by choosing as parent the neighbor with the smallest node-to-sink routing cost. The tree is periodically reconstructed with a sink-initiated message. This allows the routing topology to adjust to topology changes and simultaneously serves as a "keep-alive", enabling nodes to detect when a sink is no longer available. This, together with the implicit sink selection scheme above, provides an automatic hand-over functionality in case a sink fails.

Data reliability is achieved with a hop-by-hop recovery scheme. Data tuples contain a sequence number; upon forwarding, a small number of tuples are cached in the tuple space. When a communication failure occurs, the parent identifies a gap in the sequence and "pulls" the missing tuple from the child's cache.

Dissemination. The WSN nodes can be configured remotely by exploiting one-to-many communication from the gateways, e.g., to change the light sampling frequency or modify MAC parameters. This is useful both to implement the functionality necessary to the SCADA and to manage experiments on the WSN. To disseminate the necessary configuration commands from the gateways to the WSN nodes we employ a Trickle-like scheme [41], which guarantees eventual consistency of the information available at all nodes.

Security. In our environment, *physical* security is the greatest concern, since the WSN nodes are easily accessible and can be easily damaged or stolen. Nevertheless, we designed a simple message authentication scheme based on dynamically-distributed symmetric keys, to ensure that light readings come from legitimate nodes. This component is placed between the operating system and the middleware, effectively providing a secure channel on top of TinyOS.

3.7 Testbed Deployment

As mentioned, our final deployment is on a high-traffic road. Carrying out experiments in this site would be impractical, due to the need to block the road partially or totally, and also risky, given that we affect illumination, a key constituent of road safety. Therefore, we were granted access to a shorter, lower-traffic tunnel that served as a testbed we could more easily access to

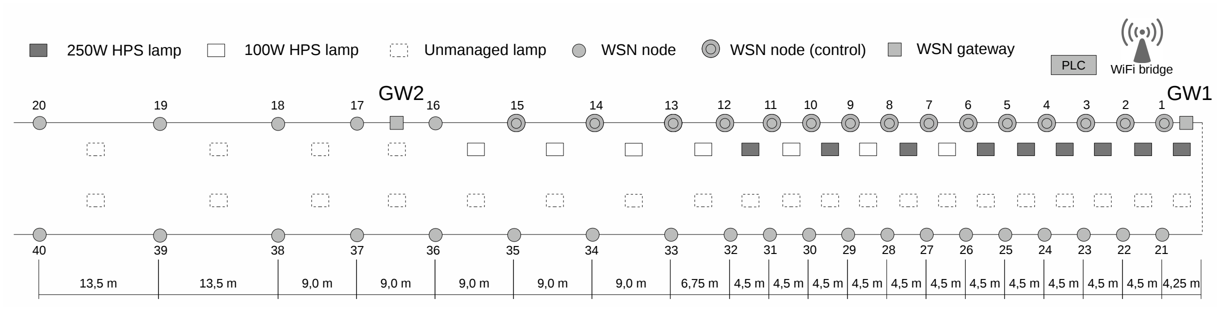


Figure 3.7: The equipment deployed in our testbed tunnel.

setup our experiments. However, the downside is that we were allowed to replace only partially the tunnel lighting infrastructure. The tunnel is a 260 m-long, two-way, two-lane tunnel. Neither automation nor communication infrastructure was present.

Figure 3.7 illustrates the equipment we deployed to match the architecture described in Section 3.5, along with the positions of the various devices. We replaced the first 16 HPS lamps in one of the lanes with 9 250 W HPS lamps (used as reinforcement) and 7 100 W HPS lamps (used as permanent), shown in the figure as dark and white rectangles, respectively. Each lamp is equipped with a ballast containing the electronics necessary to control the emitted luminous flux. These are the only lamps we can control: the others, shown as dashed rectangles in Figure 3.7, are set by the pre-existing infrastructure through a simple timer.

The lamps are controlled by the PLC, housed in an industrial rack at the tunnel entrance. The PLC bases its decisions on the data collected from the WSN, which contains 40 nodes. The nodes are split evenly between the tunnel walls, and placed at a height of 1.70 m, compatible with regulations. It is important to note that the spacing among nodes, shown in Figure 3.7, is driven more by the need to stress-test the WSN and the rest of the system rather than to *optimally* close the control loop. Indeed, in our final deployment the position of the WSN nodes is determined by lighting engineering considerations. These are difficult to derive in our testbed because we manage only a fraction of the lighting system in the tunnel. Therefore, the number of WSN nodes is higher and their placement denser than needed. The PLC relies only on the first 15 nodes: the others are extra, used for our experiments. On the other hand, we are putting ourselves in a situation that is *worse* than the one we will find in the final deployment. Indeed, while in the latter we plan to have 44 nodes over 630 m in each pipe, in our testbed we have about the same number of nodes over one-third of the length, increasing the number of collisions and retransmissions. We analyze this factor with dedicated experiments in Section 3.8.2.

The data from the WSN is collected by 2 gateways, installed on the same wall at 2 m and 80 m from the entrance. These test the effectiveness of our techniques for dividing the load of data collection and enabling one gateway to take over when the other fails. The gateways are connected to the PLC via Ethernet and powered by cables run from the tunnel power panel.

A WiFi bridge at the entrance connects the PLC with a SCADA in our labs, allowing remote configuration of the experiments and collection of results.

3.8 Evaluation

We evaluate our system first from the point of view of the application, assessing the ability to effectively and accurately close the control loop based on the data sensed by the WSN. Then, we look at the performance of the WSN itself.

3.8.1 Closing the Control Loop

First, we evaluate the response to *artificial* step-wise changes to the reference, to verify stability and convergence. Second, we evaluate the complete system according to its intended operation, with the reference properly set based on *real-world* light conditions.

Before these tests, we verified that the light readings of our sensors are indeed accurate in the tunnel, by comparing them against the illuminance probe we used for calibration in Section 3.6.2. Finally, recall that, as mentioned in Section 3.7, our testbed is realized by *partially* replacing the pre-existing lighting infrastructure. The latter includes old and unreliable lamps we do not control, influencing (and sometimes interfering with) the operation of our system.

Step response. The response of a closed loop system to step-wise changes in the reference

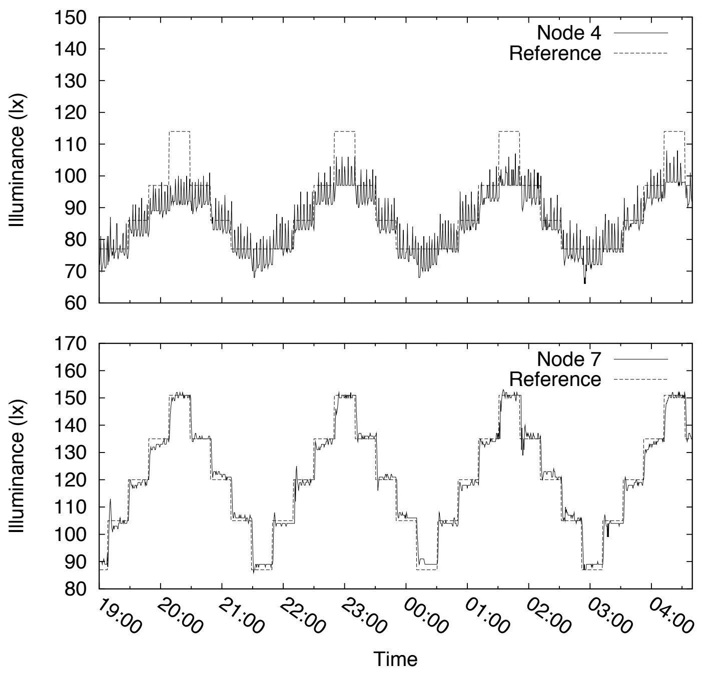
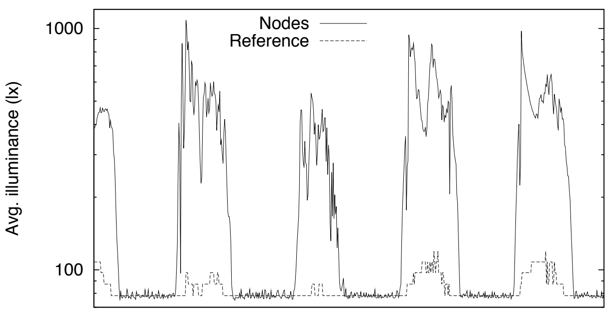


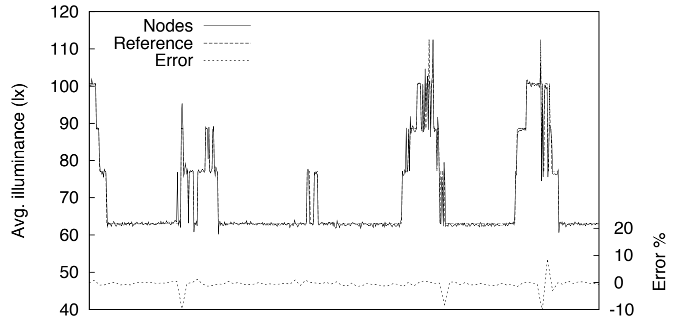
Figure 3.8: Evaluating the step response.

point is fundamental to assess both the stability of the algorithm and its ability to track the reference. Moreover, for implementation reasons, the PLC changes the reference point of all sensors in small steps and not continuously. We ran the step-response tests at night, to avoid the bias induced by daylight, thus obtaining a controlled experiment in a real-world deployment.

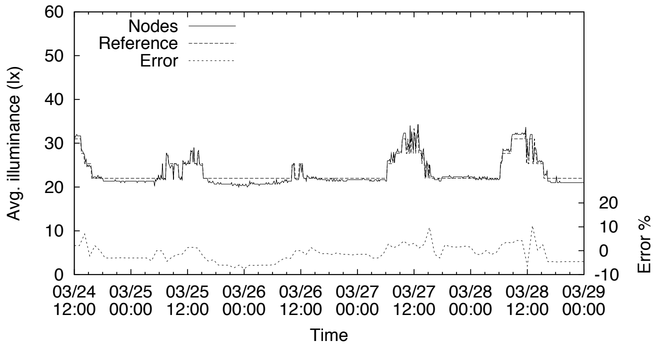
Figure 3.8 focuses on two nodes, showing their target reference value (dashed line) and the light value actually sensed (solid line). The node position relative to the lamps bears a great influence. Node 4 is in an unfortunate place, receiving only little light from controlled lamps



(a) nodes 1-6: entrance zone



(b) nodes 7-12: transition zone



(c) nodes 13-15: interior zone

Figure 3.9: Performance of control in the testbed tunnel.

and a copious amount from uncontrolled ones, some of which are old and flicker. This situation is reflected in the noisy measures and imprecise convergence shown in the figure, still the system is able to track the step-wise reference variations. The position of node 7 is instead closer to what lighting design suggests, and its tracking of the reference is very good. The behavior of the other nodes is closer to node 7 than node 4.

In our final tunnel deployment, node placement follows from an accurate lighting design: we expect the performance to be similar to or better than the one of node 7, due to newer lighting equipment. However, the impossibility to fully redesign the lighting infrastructure of our testbed led to an interesting (albeit involuntary) worst-case experiment. Indeed, the results for node 4 confirm that, even with noisy measurements and an incorrect sensor placement, our system is robust enough to follow the reference trend.

Real-world reference. Figure 3.9 shows the results of experiments over 4 days. For convenience we group sensors in zones, roughly corresponding to the entrance (nodes 1–6), transition (7–12), and interior zone (13–15). In each chart, the solid line represents the light measured in the zone, while the dashed line is the reference. The dotted line in Figures 3.9(b) and 3.9(c) is the percent error between the two, whose scale is shown on the right-hand side y axis.

The dynamic range of any control system is limited by the actuators' capability. Due to the presence of a single power circuit, the actuation at the entrance of a tunnel is limited to a maximum of 150 lx. This prevents correct control in the entrance zone, shown in Figure 3.9(a). External light enters the initial part of the tunnel and the luminous flow achievable by the installed lamps is insufficient to match it—note the log-scale of the y axis. The reference set at night is instead achieved with precision.

Figure 3.9(b) shows the situation in the transition zone: node readings follow the reference so closely that the two are almost indistinguishable. Finally, as shown in Figure 3.9(c), in the interior zone the system matches the reference closely during the day, but remains slightly below it at night. This is due to node positioning that is not the result of an appropriate lighting engineering study. However, the error remains within $\pm 10\%$ of the reference.

Despite the testbed limitations, these results show that the system can adapt effectively to the tunnel conditions.

3.8.2 WSN Performance

We report about experiments over a 7-month period from August 13th, 2009 through February 2010. We initially separated the nodes in two networks on different radio channels: one for testing the control algorithm, and the other for testing routing and sampling. In mid-October, all 40 nodes became part of the same network, reporting to gateway GW2. We added the second gateway GW1 in mid-January. Apart from these major interventions, we performed other maintenance, e.g., to modify the software on the nodes. However, we never changed the batteries: Figure 3.10 shows energy consumption at select nodes, over the entire 7-month period.

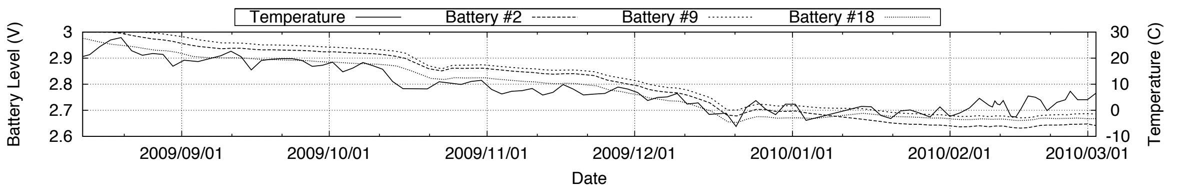


Figure 3.10: Temperature and battery levels on sample nodes over a 7-month period.

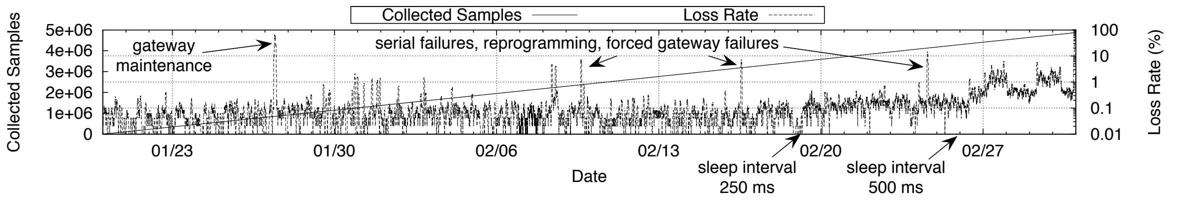


Figure 3.11: Total samples collected and loss rate over 1.5 months. The impact of the MAC sleep interval was also tested.

As we discussed in Section 3.6.3, the control algorithm relies on different sampling rates along the tunnel. Nevertheless, we configured all nodes to report at the highest one (every 30 s), regardless of their position. This yields more data to test the control algorithm, and allows us to analyze the WSN behavior in more challenging conditions w.r.t. the final deployment. Besides light samples, each node reports once per minute other data made available through the SCADA (i.e., battery voltage, temperature, and routing parent) or used for debugging and experimentation purposes.

The WSN in our testbed is much denser, and challenging, than our target deployment, as we already noted. Therefore, at the end of this section we also report about experiments in a sparser deployment matching more closely our final one.

Data yield. A fundamental metric to analyze the performance of our routing layer is the amount of data correctly received from the WSN. Our application imposes a significant workload: the required reporting frequency for light samples results in an aggregated goodput (i.e., application messages collectively flowing in the WSN) of 1.3 msg/s, that increases to 2 msg/s if one includes also the reporting of system information. Nevertheless, the loss rate typically remains between 0.1% and 0.2%, as shown with a logarithmic scale in Figure 3.11. The spike on January 27th is caused by a 2-hour intervention required to update the gateways' software due to failures of the local connection to the sink. The other major spikes up to 10% are due to other transient errors on this connection, to forced shut-down of one of the gateways to test our redundancy mechanisms, and to minor maintenance to individual nodes. The remaining, smaller, variations are actual data losses in the WSN.

The reliability of communication is influenced also by the configuration of the underlying MAC layer, in our case the low-power listening (LPL) MAC available in TinyOS[78]. Prior to February 19th the MAC was configured with a sleep interval of 100 ms. We then changed it, as shown in Figure 3.11, initially to 250 ms and, on February 26th, to 500 ms. The increase to 250 ms results in a slight, still acceptable increase in loss rate. Instead, the 500 ms interval appears to be incompatible with our high throughput and network density, as the time spent transmitting and waiting for the receiver to wake up becomes significant. Apart from these experiments, the rest of the 7-month period used the 100 ms sleep interval.

Timely delivery. In a closed-loop system, high data yield alone is not sufficient. For data to be useful, it must arrive *on time*. In our system, the control algorithm runs every 30 s on the data collected during that interval. Each data sample reported by a node is timestamped at the gateway, allowing the PLC to recognize stale data. The jitter between samples from the same node is therefore of paramount importance, as shown in Figure 3.12. If two samples from the same node are received more than 30 s apart, the PLC *may* execute one of its cycles without a sample from the node, as in the case of node A on the right-hand side of Figure 3.12. However, if two samples are more than 60 s apart, as in the case of node B, the PLC *will* miss a sample from one or more intervals.

The critical interval is therefore between 30 s and 60 s. Figure 3.13 shows the cumulative distribution function of the sample reporting jitter in this interval, for the same LPL settings considered earlier, and for the sparse network described at the end of this section. As expected, the largest 500 ms sleep interval generates an excessive jitter: because of packet losses, about 3.5% of the samples miss the 30 s deadline, and a small fraction (<0.5%) misses the 60 s one. When using the smallest 100 ms sleep interval, 1.5% of the samples misses the 30 s deadline. Increasing the sleep interval to 250 ms introduces an additional 0.5% loss. However, in both cases the system recovers the situation within 60 s. Therefore, it is never the case that the PLC misses a sample from the same node for two or more consecutive intervals. This performance, achieved without routing mechanisms devoted to reducing jitter, is perfectly in line with the requirements of our control problem: the only consequence of these delays is a minor increase in convergence time.

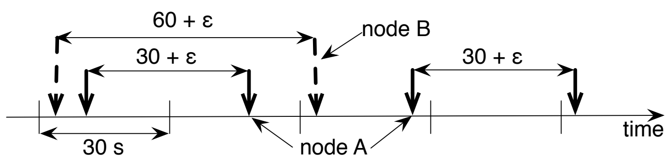


Figure 3.12: Impact of delays on the control algorithm. Vertical arrows denote arrival of a sample at the PLC.

Resilience to gateway failures. As our closed-loop control system must guarantee continuous operation, the WSN must automatically recover from failures, and limit their effects. In our target scenario, the WSN will be sparser than our testbed but still dense enough to allow alternate routes in the presence of one or more node failures, as we describe at the end of this section. The worst-case scenario is instead failure of one of the gateways, as this would prevent delivery to the PLC of *all* the data funneled through the failed sink. To reproduce this situation, we remotely forced the sink to disable its radio, disconnecting the gateway from the WSN. In this experiment, we killed gateway GW2, restored it after 2.5 hours, then killed the other gateway GW1. The two steps in the top chart of Figure 3.14 show the increase in data loss when either gateway fails. The failure of GW2 is more disruptive, as it collects data from more nodes, due to its position. After each failure, the cumulative loss rate decreases, and eventually converges to the previous values. As expected, losses are induced by gateway failures only: as shown in Figure 3.14, restoring GW2 did not affect the loss rate.

The bottom of Figure 3.14 shows instead that jitter increases sharply in the presence of a gateway failure. The amplitude of the peak corresponds to the time required by the nodes to switch to the new gateway. This occurs when the next tree refresh message is received: as

Figure 3.13: CDF for the sample reporting jitter.

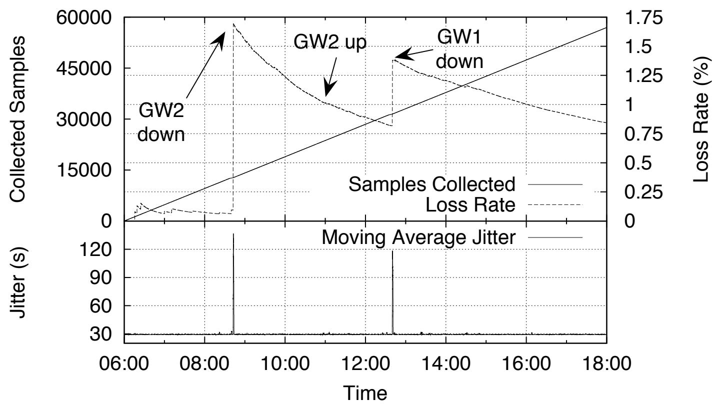


Figure 3.14: Forced gateway failures.

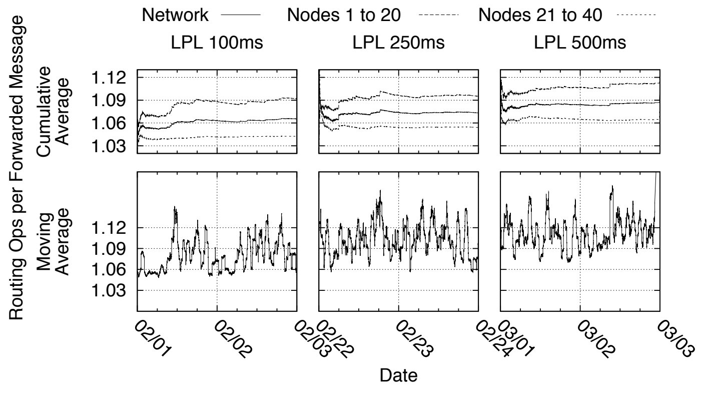


Figure 3.15: Average routing operations (send and recovery) per message forwarded per node.

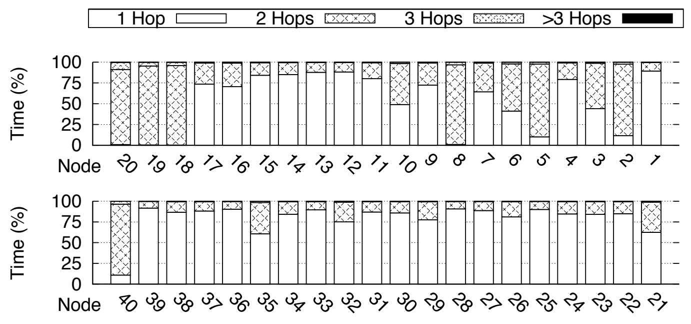
one of the gateways is missing, all nodes receive the message only from the remaining gateway, and select their parent accordingly. In our case, a tree refresh message is sent every 3 minutes: therefore, the worst-case delay is twice this time.

A closer look at routing. We obtain a high data yield thanks to mechanisms that overcome communication failures, i.e., retransmissions of non-acknowledged data and recoveries from the child cache when missing data is detected. Figure 3.15 shows the average number of *operations* issued by the routing protocol per message successfully forwarded by each node to its parent. In an ideal network, this value is 1. The chart shows the results with the usual LPL sleep intervals, each during a 2-day experiment. An operation in this context is the sending of a data tuple, or its recovery due to link failure. This high-level cost does not consider the operation details, e.g., the messages actually required to recover the lost data from the child cache, or the MAC overhead due to competing on the wireless medium. Nevertheless, it offers an indication of the effort made by the routing layer to sustain the high delivery rates in our dense setting. We provide a more detailed estimate of lifetime next. The top, cumulative average plots show that the cost to the whole network (the middle line in each plot) increases from approximately 6% for the smallest sleep interval to 10% for the largest. The bottom plots show the actual data as a moving average. For the smallest interval of 100 ms there is a higher cost during the day, when vehicular traffic slightly interferes with communication.

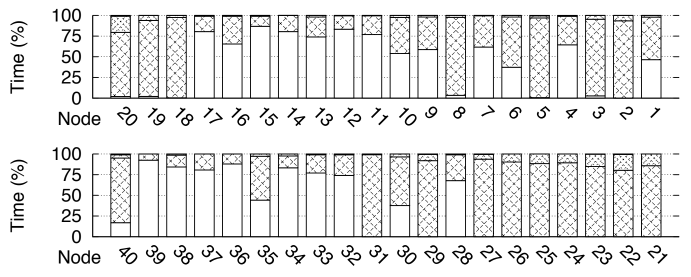
Additional insights can be gathered by considering separately the costs of nodes on opposite walls. Figure 3.15 shows that the costs for nodes on the same wall as the sinks (nodes 1-20) are approximately 5% higher than for nodes on the opposite wall. To understand why, we compute the fraction of time a node spends at a given hop count from the sink. Figure 3.16(a) shows that nearly all nodes that spend more than 50% of their time at 2 hops are located on the same wall as the sinks. Indeed, nodes communicate better with nodes on the facing wall: they select one of these as their parent, which then crosses the tunnel back using a second, high quality link to the sink. Although a multi-hop path costs more than a 1-hop link, the extremely low cost observed for 1-hop links to sinks can be attributed to the fact that these do not duty-cycle like all other nodes, therefore transmissions towards them have very low failure rates.

With this in mind, we note that when both gateways are active, nodes always select the one with the shortest path. This is clearly seen in the other two plots in Figure 3.16, corresponding to the gateway failure experiments of Figure 3.14. Interestingly, a comparison among these charts shows that Figure 3.16(a) (both gateways active) appears to be the "union" of Figure 3.16(b) and 3.16(c) (only one gateway active): a node behaves in the WSN with both gateways as in the best of the configurations with only one gateway active.

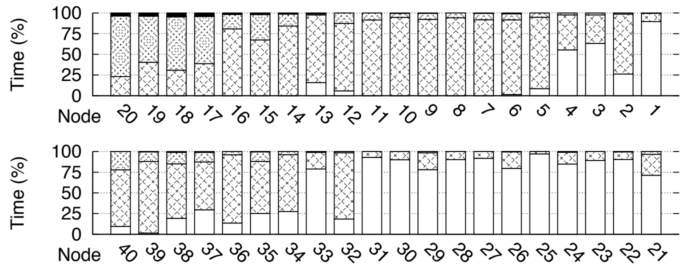
Expected lifetime. It is generally difficult to predict the lifetime of WSN nodes. This quantity is indeed affected by many unpredictable aspects, including the effect of environmental conditions on battery performance—an important factor in our target scenario.



(a) Both sinks GW1 and GW2 active.



(b) Only sink GW2 active.



(c) Only sink GW1 active.

Figure 3.16: Percentage of time spent by each node at a given distance (hop count) from a sink.

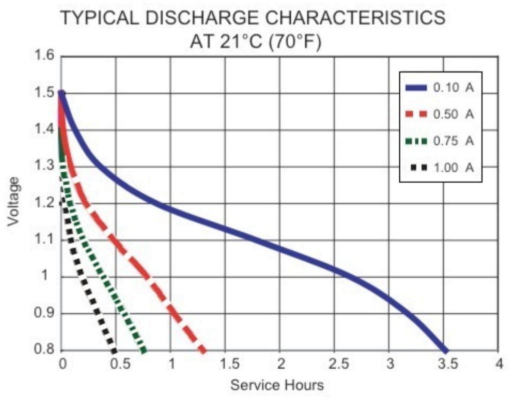


Figure 3.17: Battery discharge vs. discharge current.

To obtain a lifetime estimate, we equipped 6 nodes evenly distributed along the testbed with new batteries, and recorded their voltage readings using the on-board sensor during 22 days of continuous operation. We determine the expected lifetime as follows:

I. For each day of our experiments, we match the day-long average voltage and average temperature against the battery discharge profile we obtained from the manufacturer [32]. This determines the service hours provided by the battery, given the current voltage and temperature.
II. We compute the average temperature for every day in 2009, based on publicly-available temperature data gathered by a weather station close to the testbed.
III. Using the temperature data at point 2, we replicate "in the future" (i.e., beyond the experiment duration) the battery discharge behavior we observed, essentially simulating the latter until the number of available service hours reaches zero.

The procedure above greatly *underestimates* lifetime. First, battery discharge profiles depend on the discharge current. In point 1 above, we use the profile for a 100 mA average discharge current, the lowest value in our battery data sheet. However, the current for WSN nodes running a LPL-like MAC protocol in configurations similar to ours [63] is expected to be a few mA. As shown in Figure 3.17 for our batteries, an order of magnitude difference in discharge current determines a significant increase of service hours. Second, as we replicate the behavior observed at the beginning of a battery's life, we are considering the portion of the discharge profile where the battery loses service hours more rapidly.

Figure 3.18 illustrates the results of our analysis according to the LPL configuration. The minimal requirement of 1-year lifetime we stated in the introduction is always satisfied. The best performance always corresponds to a 250 ms sleep interval, the best trade-off between the power consumed in channel checks and packet strobing during transmissions. Instead, running LPL with a 500 ms sleep interval yields the worst performance in most cases. We conjecture that in this setting the power consumption due to long strobing outweighs the gains yielded by less frequent channel checks. Node 31, on the other hand, performs worse with a 100 ms sleep interval. Presumably, its location in the topology diminishes the bad effects of the 500 ms setting, which provides slightly longer lifetime than the 100 ms one. Instead, node 20 shows a markedly higher expected lifetime in all settings: this node is frequently a leaf in the routing tree, and therefore experiences a reduced routing load.

Towards the final deployment. The WSN behavior in our testbed is deeply affected by the high density of nodes. The final tunnel deployment will be much sparser. The placement of nodes is driven by considerations of the lighting engineers, who place the nodes in the interior zone—where fine-grained measurements are less important—with an inter-node distance up to 60 m. To gather insights on long-range links, we setup experiments using 8 new nodes, divided evenly between the two walls. The experiments ran from December 4th to the 14th, on a different radio channel w.r.t. the rest of the WSN in Figure 3.7. For these tests we relied on GW1 (recall that the two-gateway tests began in mid-January). The 8 nodes were positioned as follows. Starting from the sink on GW1, nodes on the same wall were placed 60 m apart, and approximately 30 m from the closest node on the facing wall. To investigate if connectivity was retained in the presence of node failures, we disconnected two of the central nodes (node 4 and 5) on December 10th and 11th, yielding an inter-node distance of 120 m on the same wall. These experiments ran with a 100 ms LPL interval.

Figure 3.19 shows the results. In the sparse WSN we setup, our routing achieves 99.98% delivery with an extra cost (i.e., #operations) less than 4%, as shown by Figure 3.19(a). Spikes in the moving average for cost are due to the selection of a low quality parent, and are absent in times of low vehicular traffic (e.g., the weekend of December 12-13). The overall improved performance w.r.t. previous experiments is motivated by the decreased density: channel contention is reduced and the MAC layer handles communication more effectively. Notably, the forced failure of the two center nodes has no impact, confirming the considerable length of reliable links in the tunnel environment. The routing tree, shown in Figure 3.19(b), has a higher depth w.r.t.

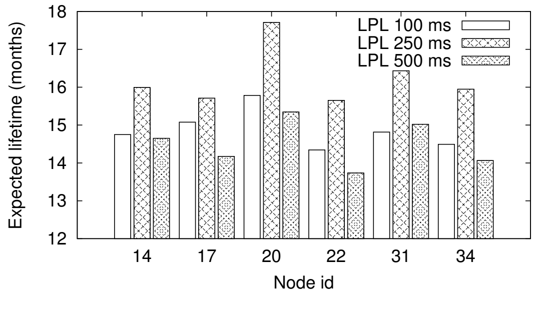
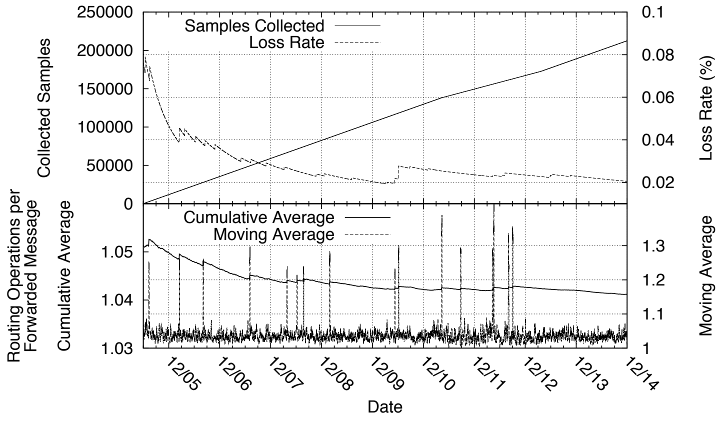
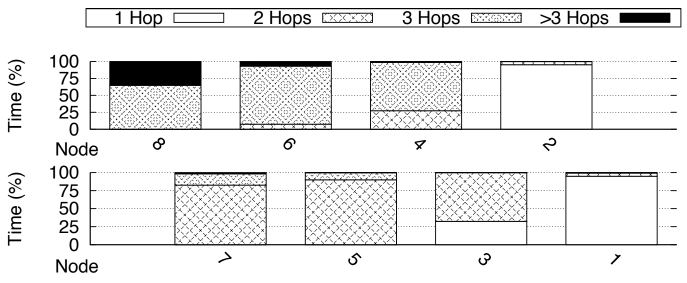


Figure 3.18: Expected lifetime, beyond one year.



(a) Data loss and cost in terms of number of operations



(b) Time spent by each node at a given distance from the sink

Figure 3.19: Experiments in a sparse setting.

the situation in Figure 3.16. Nevertheless, this does not negatively affect the jitter, as shown in Figure 3.13.

Although the ultimate answer will come from the final tunnel deployment, these results show that the latter should perform better than the overly-dense experimental testbed reported in this chapter.

3.9 Beyond Adaptive Lighting: Fire Detection

TRITon is a large project encompassing several technologies. At one point, another team tested their camera-based fire detection system with real fires staged by the local fire department. As this occurred in our testbed tunnel, we took advantage of the event to investigate whether the WSN we conceived for light sampling could be used also for fire detection.

Indeed, the ISL29004 illuminance sensor we used relies on two photodiodes: the first one (D_A) is sensitive to both visible and infrared (IR) light, while the second one (D_B) is sensitive mostly to IR. Measuring illuminance requires that the wavelength of incident light is compensated to follow the human eye response, which is achieved by “subtracting” the response of D_B from the

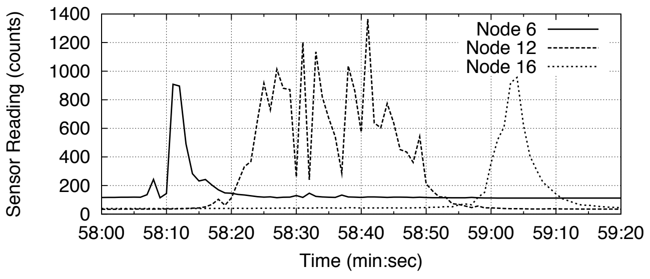


Figure 3.20: Detecting fire through infrared light sensing.

one of D_A. It is well-known that fire, unlike tunnel lamps, emits a significant fraction of light in the IR spectrum. Detecting fire with the ISL29004 sensor then essentially means monitoring the output of D_B: in the presence of fire, this diode immediately reports a value much higher than D_A. Therefore, in practice, running this experiment came at negligible cost. We modified the sensor driver to report IR along with illuminance, and made minimal changes to the sampling rate (1 s) and message format.

In the experiment we report here, a fireman on the back of a truck held a tube connected to a propane tank, which continuously fueled a flame at the free extremity of the tube. The truck moved from right to left w.r.t. Figure 3.7. Figure 3.20 shows the data reported by our WSN, charting over time the IR values from nodes at different distances inside the tunnel. The presence of a flame causes a distinct and instantaneous increase in the IR value. The many peaks at node 12 are due to the fireman waving the flame in front of it.

Full-fledged fire detection requires more in-depth studies. However, this impromptu experiment hints at the fact that, once a WSN is deployed in a tunnel, applications other than lighting become feasible, possibly with only minimal changes to the base design.

3.10 Concluding Remarks

Differently from the experience described in Chapter 2, we reported on a WSN monitoring infrastructure closing a control loop. Despite differences in the requirements on the WSN, e.g., timeliness, we successfully demonstrated the possibility to reuse components developed for the system in Torre Aquila, as fostered by the usage of the TeenyLime middleware. The dependability of the employed solution was proved by several months of operation in an experimental testbed and, from August 2010, in the final deployment site.

Part III

Bringing Quality into Communication

Chapter 4

Bringing Anarchy to TDMA in the Versatile, Fully-Distributed Reins-MAC

The experience acquired by building two operational monitoring infrastructures presented our concrete background on real world deployments. Their definition clearly manifested a set of requirements imposed by the end user on the network: reliability, efficient support for heterogeneous classes of traffic and timeliness. In the development and implementation of these systems, we made full use of the default medium access control mechanism provided by the operating system we used. Despite the fact that no guarantee is explicitly provided by such a solution, the system services built on top demonstrated themselves to be sufficient to support the required service quality in both our deployments.

Nevertheless, the impact of different configurations of the protocol on the system performance, as experimentally faced in Chapter 3, provided an intuition of the restrictions imposed by the specific employed solution. Motivated by our own direct experience and further inspired by the recognized limitations of the current state of the art [14, 39], we defined Reins-MAC. We provided, in the fully distributed and versatile nature of the introduced solution, an innovative access control scheme able to bring quality into the definition of communication.

4.1 Motivation and Contribution

Effectively managing access to the shared communication medium of a WSN has been one of the fundamental challenges faced since the emergence of the vision of large wireless networks built by resource constrained devices. A plethora of MAC protocols has been proposed [39]. Due to the generally successful employment of CSMA solutions in real deployments [59, 9, 20], the problem is often considered solved. Nevertheless, no solution exists to support application driven Quality of Service (QoS) requirements in an effective manner.

When such guarantees are needed, systems adopt TDMA-like solutions and enforce disciplined coordination through rigid slot-based scheduling [72, 5]. One commonly recognized limitation of these approaches is their inability to adapt to the typical dynamic behavior typically seen in networks of low power cooperating objects [14]. Further, their inherent complexity and required compliance to a rigid discipline foster the belief that dynamic changes to the communication schedule are infeasible [39].

We introduce Reins-MAC, a new TDMA-based protocol that clearly demonstrates that a dynamically adjustable, flexible solution *is* feasible and the resulting coordination offers a solid foundation for QoS requirements arising from higher layers in the protocol stack. Reins-MAC rejects the common TDMA assumption that a single, network-wide slot size is required. Instead it adapts the slot size at each node to match local availability, achieving tremendous **gains in bandwidth utilization**. The approach that accomplishes this simultaneously allows Reins-MAC to **adapt to the natural connectivity variations** present in WSNs. Further, the flexible slot sizes of Reins-MAC can be explicitly tuned to **support dynamic application requirements**, providing guaranteed communication. These points are expanded upon in Section 4.2.

The Reins-MAC protocol builds on the theoretical foundations of Pulse-Coupled Oscillators (PCO) (Section 4.3), an intuitive and easy to implement mechanism to organize individual node behavior in a network, originally inspired by firefly flash-synchronization. To successfully adapt PCOs to support scheduling in Reins-MAC, we extend the basic scheme to make explicit the time (allowing flashes to have duration) and space (restricting flashes to be seen at defined distances) dependencies among node activities (Section 4.4). The result is Reins-MAC, a TDMA-protocol with adaptable slot size and flexibility to adapt to network variability (Section 4.5), capable of both facing typical communication shortcomings experienced in real world scenarios (Section 4.6) and support QoS guarantees (Section 4.7).

These achievements are supported by extensive experimental evaluation of Reins-MAC (Section 4.8) showing scalability, quick response to varying conditions, and solid support for higher level quality of service requirements. Finally, the implementation of the solution in a real testbed proves the feasibility of the approach. These results make Reins-MAC a clear step forward with respect to the current state of the art (Section 4.9) overcoming TDMA limitations and supporting Quality of Service requirements.

4.2 Why (Another) TDMA?

By dividing communication in periodic frames with slots allocated to unique senders, TDMA solutions offer guaranteed communication. This section offers motivation for a new approach, first discussing why TDMA techniques are generally appealing for a variety of applications, showing a challenge that places a barrier to efficient TDMA solutions today, and concluding with a summary of the benefits of our approach, including how it overcomes the previously outlined challenge as well as other common criticisms to TDMA.

4.2.1 Why Consider TDMA

To support the claim that a TDMA solution offers many benefits over a CSMA (e.g., B-MAC [65], X-MAC [4], or their combined TinyOS implementation, BoX-MAC [56]) solution, we offer few back-of-the-envelope calculations based on our experiences with the road tunnel deployment described in Chapter 3, and the one in Torre Aquila described in Chapter 2.

Homogeneous Network with Timeliness Requirements. In the road tunnel scenario, we deployed a WSN to monitor light. The information provided by the sensors is used to adaptively govern lamps accordingly to current internal and external lighting conditions. This closed control loop ultimately saves energy and improves safety. Our testbed involved a dense deployment of 40 nodes with a maximum network diameter of 3 hops. Given that data collected at a gateway is used to control actuators, it is required that one sample from each node arrives at the sink every 30 s. For a standard TDMA solution, we must define the frame size and the number of slots per frame.

To guarantee that data, moving in the worst case only one hop per frame, reaches the sink on-time, the frame could be as long as 10 s (30 s / 3 hops), however, taking a conservative approach, we suggest a frame length of 5 s. To determine the number of slots, we conservatively suggest 40 slots (one per node in the network), which yields a slot size of 125 ms (5 s / 40 nodes). Our 100 bytes data packets require approximately 5 ms to transmit; in each slot, a node can therefore send more than 20 messages, either to deliver local data or to support multi-hop transmission. If fewer packets must be transmitted, the node is allowed to sleep for the remainder of the slot, saving energy.

One criticism of TDMA is the requirement that nodes wake up at the beginning of each slot to potentially receive a packet from the neighbor assigned to that slot. In the scenario just described, with a maximum 1-hop neighborhood size of 20 nodes (again, a conservative, over estimate), the 5 s frame length requires a node to wake up on average every 250 ms. This is in line with the wake-up requirements of BoX-MAC with a reasonable sleep interval.

Further, once the schedule is built, sending a packet in TDMA requires only data and headers to be transmitted. Instead, a CSMA protocol must also send some amount of preamble (an average of half the sleep interval for BoX-MAC). Additionally, TDMA, by construction, ensures collision free communication, bounding the additional costs for retransmissions and acknowledgments to the actual link losses. Further, TDMA has the potential to *guarantee* the timeliness requirement of the application; a CSMA implementation does not.

Heterogeneous Network with Bandwidth Requirements. The deployment in Torre Aquila supports offline structural monitoring in a medieval tower with a 15 node, 6-hop network. This system includes three accelerometers, each sending bursts of 1422 packets. To avoid overloading the network, our BoX-MAC-based implementation sends maximum one packet per second; data from a single sampling session, without compression, takes 30 minutes to reach the sink.

For a TDMA solution, we conservatively suggest a frame length of 4 minutes with 15 slots. The resulting 16 s slot size is sufficient for all packets in the burst to be forwarded one hop, meaning that data will arrive at the sink in the worst case after 24 minutes (4 minutes * 6 hops).

The energy expenditure between TDMA and CSMA in this case is notable. In our BoX-MAC implementation, a node on the data path will, for each packet (optimistically): (1) transmit on average for 50 ms (half of the employed 100 ms sleep interval) and (2) receive for 10 ms.[1] In other words, the radio will be active for at least 85 s in the 30 minute interval, ignoring retransmissions. With a TDMA solution, instead, a node will (1) transmit for 15 s and (2) receive for 15 s, for a total active time of only 30 s per 24 minute interval. This means that the network lifetime is expected to double, simply by adopting TDMA.

Additionally, in our BoX-MAC implementation, our experience suggests that adding more accelerometers would overload the system due to increased channel contention and collisions. Instead, with TDMA and a 4 minute frame, in 24 minutes we could transmit data from 6 accelerometers without creating bottlenecks. Alternately, we could increase the data rate from the existing sensors. Notably, for a TDMA solution we can easily *calculate* these possibilities, instead the CSMA behavior depends heavily on the random timings to access the channel, which possibly cause collisions, and complicate similar calculations.

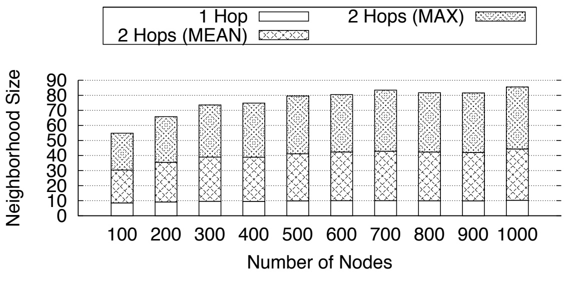
4.2.2 Why *Not to* Consider TDMA

While the rough calculations above clearly support the use of TDMA for a range of applications, accurately selecting the slot size remains a challenge. Here we consider this issue for fixed ideal networks, and outline additional challenges arising from the natural dynamics of WSN environments.

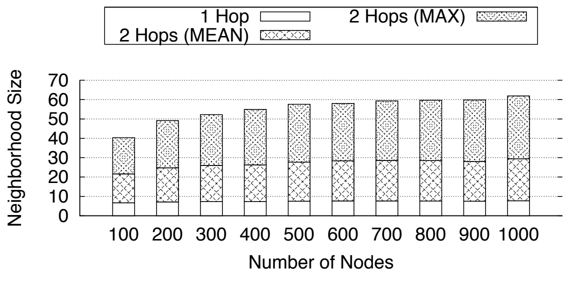
TDMA in Wonderland. Under the assumption of an ideal channel, communication properties among nodes remain constant over time, without the manifestation of intermediate delivery probabilities. In this scenario, the *interference sets*, i.e., the direct and indirect neighbors whose simultaneous communication results in a collision at a potential receiver, are invariant over time. A TDMA protocol, to avoid collision by construction, must assign a different slot to each node in the same interference set. As defined by the hidden terminal problem, the area of possible communication interference spans the 2-hop neighborhood.

Figure 4.1 (we defer details of our simulation setup to Section 4.8.1) shows the maximum (Max) 2-hop neighborhood size for networks of different sizes and densities. Notably for networks

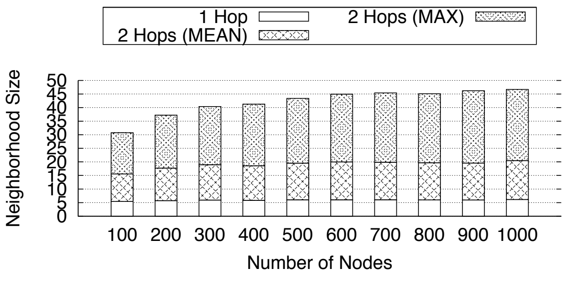
[1]We ignore the BoX-MAC detail according to which a node remains in listening mode for some time after receiving a packet.



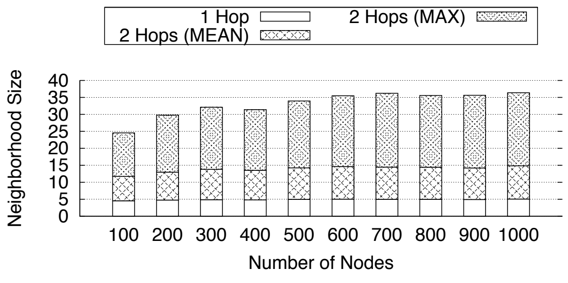
(a) Distance 24.



(b) Distance 28.



(c) Distance 32.



(d) Distance 36.

Figure 4.1: Variability of the 1-hop and 2-hop neighborhood for multiple network sizes and densities. *Distance* indicates the average distance between nodes (smaller distances imply higher density).

of 100 nodes, more nodes are on the edges of the network, lowering the average neighborhood size. Instead, for networks with 500 or 1000 nodes, this effect is minimal, and average sizes are more consistent. In any case, a typical TDMA solution will identify a priori a single, network-wide slot size corresponding to the maximum 2-hop neighborhood size, as it ensures that a collision-free schedule can be devised. Identifying the actual maximum density of the network is challenging, usually resulting in worst case provisioning and reduced bandwidth utilization [35].

The figure also shows the mean (Mean) 2-hop neighborhood size; the gap between Mean and Max indicates the variability of network density throughout the topology. Intuitively, in areas of low density, allocating the number of slots according to the maximum density will leave some slots unassigned, wasting bandwidth. For example, in the mid-density scenario defined by an inter-node distance of 28, shown in Figure 4.1(b), the difference between Mean and Max is approximately 30, implying a sizable waste.

The assumption of no slot reuse, used so far, is a simplification that does not necessarily hold in solutions available in the literature. Mechanisms to assign unused slots have been defined at the cost of an increased algorithmic complexity and reduced scalability.

TDMA in the Real World. The above considerations are based on an idealized scenario with stable networks. Instead, low-power wireless channels deviate from the ideal one [74]: links manifest intermediate packet delivery probabilities, asymmetries and variability over time.

In real world conditions, the definition of the interfering set is non trivial. In addition to perfect links with either 100% or 0% probability of delivery, it is common to find intermediate links. Moreover, considering the variability over time, a node may become part of a 2-hop neighborhood different from the one in which the schedule was formed. As the interfering set changed, the schedule must be updated, which usually reduces to a rebuild from scratch. As a result, a TDMA solution should implement a monitoring mechanism running frequently enough to adjust the schedule to the current conditions. Moreover, these changes in the connectivity may vary the size of the 2-hop neighborhood, and their occurrence in a dense area may invalidate the slot size chosen before deployment under the initial network conditions.

Finally, the discretization of time, defined by the slot size, requires time synchronization, which needs to be executed continuously to account for clock drifts. Usually the time alignment is handled with dedicated protocols, independent from the scheduling ones. This forced decoupling of protocols, heavily affecting one another, hampers the efficiency of the whole solution. The need for continuous realignment due to the poor clock quality and the intrinsic price of the synchronization mechanism are commonly recognized obstacles to the usage of TDMA solutions in practice.

4.2.3 Why Consider Reins-MAC

With Reins-MAC, we have developed a TDMA protocol that dynamically adjusts to both network density and connectivity changes, thus wasting less bandwidth and tolerating natural WSN dynamics.

Bandwidth Gains. To concretely demonstrate the potential gains, Figure 4.2 shows the bandwidth allocated by Reins-MAC in comparison to two idealized slot allocation schemes: `MAX` and `MEAN`.

`MAX` is representative of common TDMA solutions and establishes a single, fixed slot size for the entire network by dividing the frame equally among the largest 2-hop neighborhood: $1/max$(2-hop neighborhood size). `MAX` is only an indication of what an optimal solution can achieve, assuming perfect information about a stable topology and without considering possible subsequent neighborhood set fluctuations. At the same time, it does not consider any possible slot reuse, which can increase the bandwidth individually available.

`MEAN`, instead, considers a hypothetical solution with variable slot sizes set according to the local density. Specifically, `MEAN` divides the frame among the average 2-hop neighborhood: $1/avg$(2-hop neighborhood size).

Figure 4.2 shows that Reins-MAC achieves an increase in the average slot size for all scenarios, up to more than 200%. This means that Reins-MAC can support *up to three times the rate* of a typical TDMA solution. Further, the same mechanisms that allow it to adjust the allocated bandwidth according to variable densities allow it to dynamically adapt to network

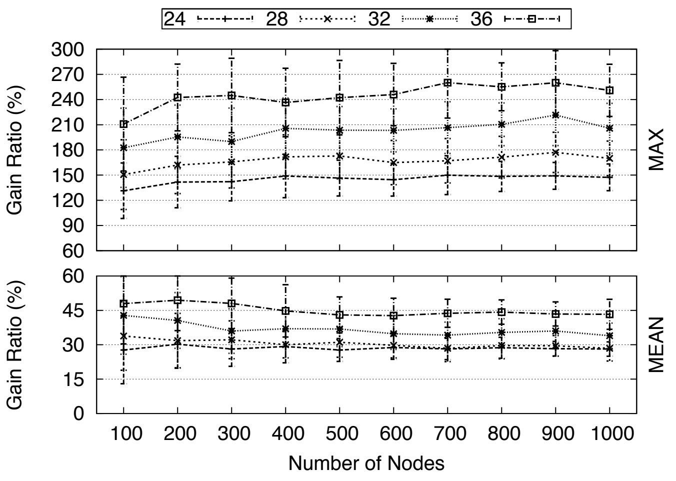


Figure 4.2: Increases in allocated bandwidth with REINS-MAC with respect to two idealized TDMA schemes: `MEAN` and `MAX`. (Note the different y-axis scales of `MEAN` and `MAX`.)

changes. Details on the REINS-MAC mechanisms enabling these significant gains are offered in Section 4.5.

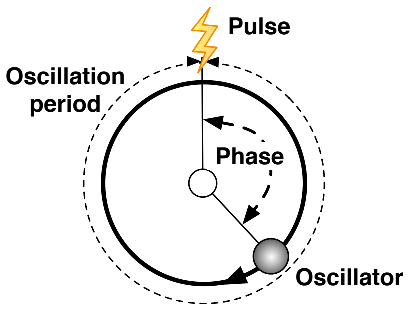
Support for dynamic applications. To this point we have considered only applications with static constraints. Instead, many applications vary their requirements during operation.

For example, in the volcano monitoring [82] application, WSN nodes periodically report summary information, but, when something "interesting" is detected, the gateway requests detailed data from specific nodes, dynamically changing the data profile of the application from periodic low data rate transmission to bulk transmission along the path from the selected data sources to the sink. A single slot size common to all nodes may not support the required bursts.

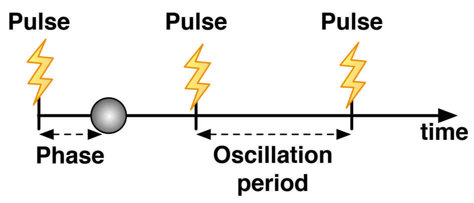
Instead, a better approach would adapt the slot size to accommodate the required bursts, specifically reserving bandwidth on the required paths. Similarly, latency requirements can vary over time with the application. Section 4.7 describes how REINS-MAC supports these dynamic needs.

Simplicity of approach. TDMA is also criticized for the complicated mechanisms required to establish collision free schedules, the overhead required for time synchronization, and the overall complexity of the implementations.

We argue that REINS-MAC successfully addresses these concerns. By basing the protocol behavior on the PCO mechanism, REINS-MAC takes on a simplicity that is both easy to understand and implement. Further, its fully-distributed nature reflects the locality of forming a 2-hop collision-free schedule, allowing it to scale to large deployments. Finally, REINS-MAC does not rely on any external time synchronization mechanisms, as the required synchronization is guaranteed in the core functionality of the protocol.



(a) Phase Domain.



(b) Time Domain.

Figure 4.3: Pulsing Oscillators scheme.

4.3 Background

Pulse-Coupled Oscillators [64, 54] are a distributed coordination scheme inspired by firefly behavior. Here we show how PCOs can naturally describe distributed solutions to synchronization and scattering problems. The minor modifications required to apply PCOs in wireless networks (Section 4.3.2) combined with our novel extensions (Section 4.4) form the groundwork for describing Reins-MAC.

4.3.1 Core Concept: Pulse-Coupled Oscillators

PCOs draw inspiration from firefly behavior [54]. Each beetle periodically emits a light. Other fireflies observe these flashes and alter the timing of their own flashes, eventually yielding a unison blinking of the whole swarm. We focus first on individual behavior, then on mutual coupling.

Formally, the periodic behavior can be described as a cyclic *oscillator* manifest as a *phase* variable that *(i)* assumes a value in a given range, e.g., $[0, 1)$, *(ii)* increases linearly over time and *(iii)* resets to the first limit when the second is reached. To mimic the flash of a firefly, resetting the phase variable can be combined with a *pulse* that is essentially a notification of the reset. As shown in Figure 4.3, the resulting behavior is a periodic pulsing at the oscillation frequency. We formalize the notion of pulse observation by defining for each oscillation period k, the set of observed pulses as:

$$\Theta_i^k = \{\theta_{i,j}^k | j \in c(i)\} \tag{4.1}$$

where $c(i)$ contains the node identifiers whose pulse i can observe, and $\theta_{i,j}^k$ is the value of the phase at node i when j pulsed. For notational simplicity, we remove the assumption that the local pulse happens when the phase variable resets; instead we define ϕ_i^k as the phase of the local pulsing.

In fireflies, pulsing and observing are not sufficient to produce synchrony, which is achieved by either actively anticipating or delaying the individual pulse. Therefore, the phase variable

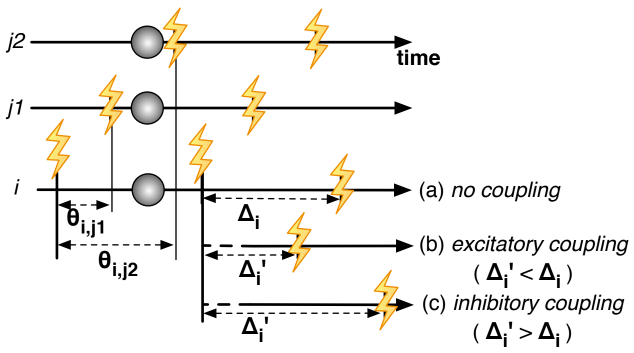


Figure 4.4: Coupling functions applied at node i: (a) no coupling function (the interval between pulses equals the oscillation period); (b) excitatory coupling (reduced interval); (c) inhibitory coupling (increased interval).

no longer varies as a constant, linear function over time, but exhibits spontaneous changes in response to observations. The rules driving such changes are captured in a *coupling function* that establishes the phase variable at which the flash will occur in the next oscillation round $k+1$:

$$\phi_i^{k+1} = f(\phi_i^k, \Theta_i^k) \tag{4.2}$$

. As shown in Figure 4.4, early pulsing is referred to as excitatory coupling, while delayed pulsing is inhibitory coupling. Adapting the local phase according to observation can be exploited to drive the emergence of different network behaviors, such as pulse synchronization or scattering.

Pulse Synchronization. Continuing the firefly analogy, PCOs have been successfully applied to achieve synchronization without requiring a master, reference anchor. The original mechanism [54] applies only excitatory coupling by moving pulses *more* when far from synchrony, and *less* when closer to being synchronized. Interestingly, another time synchronization technique [43] can be simply expressed in PCO terms. In contrast to the original approach, it combines inhibitory and excitatory coupling, modifying the pulse according to the average of the perceived pulses:

$$\phi_i^{k+1} = \frac{\sum_{\theta_{i,j}^k \in \Theta_i^k} \theta_{i,j}^k + \phi_i^k}{|c(i)| + 1} \tag{4.3}$$

Intuitively, such a combination of inhibitory and excitatory coupling brings the system to convergence by iteratively decreasing the distance between the extreme pulses and reducing the synchronization error round by round.

Pulse Scattering. Another distributed problem that can be reduced to the PCO scheme is *pulse scattering* [25, 19], whose goal is to obtain pulsing schedules where all the nodes that can hear each others beating are evenly spread throughout the oscillation period. A simple but effective solution is to distance the local pulse as much as possible from the surrounding ones; this is achieved by placing the beating almost exactly in the middle between two perceived pulses:

$$\phi_i^{k+1} = (1-\eta)\phi_i^k + \eta(\theta_{i,PREV(i)}^k + \theta_{i,NEXT(i)}^k)/2 \tag{4.4}$$

, where `PREV(i)` and `NEXT(i)` are the preceding and following nodes in the pulsing schedule and η is the coupling strength that defines how close to the center of the two pulses the local pulse is placed.

4.3.2 PCOs in Wireless Networks

The scheme presented can naturally be adopted in WSNs to couple node activities. In addition to synchronization [83], scattering has been applied to spread sensing activities among duty cycling nodes [25]. Specifically, if sensing tasks are scheduled when a local pulse happens and the sensing area covered by a node is comparable to its communication range, the pulse notifications a node perceives come from the nodes sensing similar events. Therefore, aligning sensing to the pulsing schedule reduces the temporal overlaps when neighboring nodes actively sense the same area.

PCOs can be easily implemented in WSNs with a local clock to drive the oscillation and a broadcast message to notify others of the phase reset. By receiving notifications, a node builds the set of observed beatings. In multi-hop networks, direct observation is limited to the nodes in communication range. This set is then used to compute the changes in the pulsing according to the employed coupling function.

Wireless communication imposes limitations due to the shared nature of the medium, where simultaneous transmissions can collide, as well as the cost of radio activations. Random delays (termed *deferred pulsing*) have been employed to reduce the probability of collisions. Moreover, although the original PCO scheme [54] applied a coupling function after each received pulse, straightforward modifications achieve the same effect when applying the function only one time in each round [83], reducing both the number of radio activations and the probability of pulse collisions.

4.4 Extending PCOs

Although the presented PCO scheme offers a solution path to different problems, we identify two fundamental constraints that limit its applicability. First, the pulse is instantaneous, while node activities driven by the pulsing have a finite *duration*. Further, node behavior can couple only with nodes immediately observable, whereas inter-node dependencies may span beyond the perceivable *horizon*. We consider these as temporal and spatial restrictions and propose straightforward extensions to PCOs to accommodate them.

4.4.1 Time and Space Dimensions

Pulse Duration. In PCOs, a pulse is an instantaneous event with an immediate effect. E.g., when used for time synchronization, the event (the unison beating) occurs suddenly. Consider, in contrast, that in the example of pulse scattering each node is awake to detect events for a finite time. We can conceive of situations where nodes may have different awake intervals, e.g., keeping nodes with more energy active for longer time. To support this, we make the pulse dependent on the activity length with a *pulse duration* adding a temporal dimension. Each pulse is described as:

$$\Phi_i^k = \langle \phi_i^k, \delta_i^k \rangle \tag{4.5}$$

, where ϕ_i^k is the pulsing reference and δ_i^k is the duration of the pulse i at iteration k.

Pulsing Distance. As defined thus far, a node establishes its own pulse time by coupling with other nodes it can directly communicate with. In general, the influence between activities inside a community need not be restricted to the members in immediate contact with one other. E.g., the hidden terminal problem arises when simultaneous transmissions by nodes not able to hear each other interfere at an intermediate node. Generalizing this, we make explicit the range or distance of influence in the coupling scheme by introducing *pulse distance*: the symbolic distance at which a remote pulse can be perceived. Pulses that can be directly perceived are at distance 1; the pulses heard by nodes at distance 1 have distance 2, etc. In the end, the set of perceived pulses is:

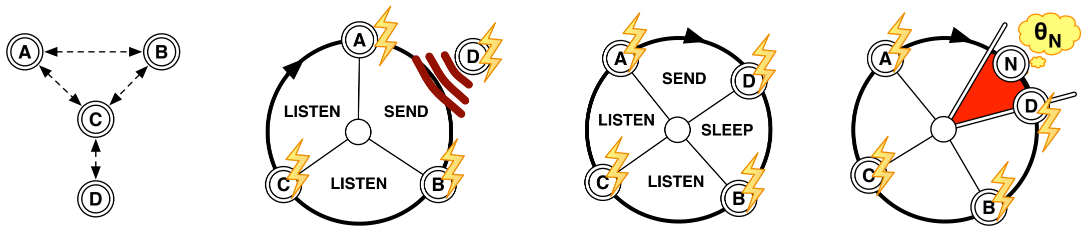
$$\Theta_i^k = \{\langle \theta_{i,j}^k, \delta_j^k, h_j^k \rangle | j \in c(i) \text{ and } h_j^k <= H\} \tag{4.6}$$

, where h_j^k is the distance of the pulse at node j and H is the *coupling horizon* of the coupling function. Therefore, all the pulses *h-distant* from the point where the function is evaluated combine to influence the pulsing behavior.

4.4.2 Extended PCOs in Wireless Networks

Nature implements the pulsing mechanism in fireflies with a light that inherently conveys the required information, i.e., the time of the flash. Migrating to the artificial, wireless network domain introduces both advantages and disadvantages. Using a broadcast message to model the phenomenon, the transmitted packet can be exploited to carry arbitrary information. As mentioned, collisions, not present with light pulses, can be overcome with the *deferred pulse* mechanism, with packets advertised at a random offset after their actual firing. To compensate for this delay, the offset is carried in the message.

To implement the proposed extensions, we define the structure of a single pulse as the 4-tuple ⟨*source identifier, distance, duration, offset*⟩. Multiple tuples can be aggregated, e.g., allowing a node to append its own tuple to the tuples of nodes from distances within the coupling horizon. By combining tuple aggregation with deferred pulsing and allowing several messages to compose the pulse, each node can send information about multiple pulses at once.



(a) Network Topology. (b) 1-hop Scattering at A. (c) 2-hop Scattering at A. (d) Polite Joining of N.

Figure 4.5: Core Reins-MAC functionality.

4.5 Towards Reins-MAC: PCOs Exploited

The extended coupling scheme of Section 4.4 can be used as basis to solve many problems of coupling behavior in distributed settings. One such problem in WSNs is sharing communication resources. As sketched in [18], and further developed in [58], PCOs offer the foundation for a TDMA-like MAC protocol overcoming many TDMA shortcomings.

Our approach creates slots and assigns them to senders, but unlike most TDMA approaches, the slot size of each node can vary. To achieve this, we use *pulse scattering* to identify the beginning of the slot for each node, and apply a *pulsing distance* to ensure communication does not overlap within a 2-hop radius. Further, we use *pulse synchronization* to identify a slot common to all nodes that is exploited to coordinate schedule changes, e.g. allowing node additions without interfering with ongoing communication. The resulting protocol, Reins-MAC, provides the basic mechanisms required to share communication resources in an application agnostic manner. In this section, we introduce our TDMA scheduling assuming an ideal scenario with perfect links; in Section 4.6, we face the common problems of realistic links.

4.5.1 TDMA with adaptable slot size

Problem Definition. In TDMA-based communication, each member is assigned a portion of time in each frame to allow collision-free interactions. Such an allocation must take into account interference caused both by nodes in direct communication range as well as by those connected by an intermediate node. The main challenge in a multi-hop scenario is, in fact, the classic hidden terminal problem in which two nodes, A and D in Figure 4.5(a), that cannot communicate with one other, simultaneously attempt to communicate with the same node, C, causing interference at C. Preventing such collisions is the main goal of requiring all 2-hop neighbors to have non-overlapping slots. It is worth noting that this property is localized to the 2-hop neighborhood.

Classic Solution. In typical TDMA solutions, time is divided into periodic frames composed by a sequence of slots assigned to each node. The slot size is defined a priori and is equal at all nodes. The value must be tuned for each deployment. As discussed in Section 4.2, basing slot size on maximum density can result in bandwidth waste. Further, most TDMA protocols need extensive information to form schedules, e.g., requiring complete information at a sink or complex distributed mechanisms to guarantee collision avoidance. Such rigid schemes cannot easily adapt to changes in connectivity arising from either changing environmental conditions or explicit addition or removal of nodes.

Through the PCO lens. We approach slot size selection and slot allocation through the lens of PCOs, noting the similarity between this communication problem and that of pulse scattering in Section 4.3.1. Consider the case when all nodes are within communication range. Nodes spread their pulses evenly throughout the frame. If a transmission slot for a specific node is defined to begin at the moment at which it pulses and end when the next node pulses, a TDMA schedule with variable slot sizes emerges. This solution has been studied in [19]. This solution is applicable only for 1-hop scenarios.

Consider directly applying such an algorithm in a multi-hop network. While node C in Figure 4.5(a) will scatter with respect to both A and D, because A and D are not within range, they do not scatter with respect to one another and their slots may overlap. Simultaneous transmissions from A and D will collide at C, as shown in Figure 4.5(b). This issue is solved by applying pulse scattering with a horizon of 2 as in [18, 58] and shown in Figure 4.5(c). When the node's own pulse fires, it can transmit; when a 1-distant pulse is received, the node listens for incoming transmissions; when a 2-distant pulse is expected the node, if it is sending, stops any transmission.

In [18, 58], no duty cycle scheme is described. However, in the assumption of ideal channels and perfect clocks, a simple scheme can make each node wake up right before its own slot, send the local pulse and the data messages, followed by a concluding message indicating the end of the transmission. After this last message, the sender can turn off the radio. Then, right before a 1-distant pulse is expected, the node can wake up to receive both pulse and messages. Finally, once the sender signals the end of the transmission, the node can return to sleep. This scheme fails in real settings, where clock drifts and lossy links can cause a node to miss a pulse. We face this problem in Section 4.6.

It is worth noting that the described scheduling mechanism does not require time synchronization, as the schedules are based on a per-node reference system. Although nodes must know *when* the other nodes in their schedule will transmit, the nodes are not required to share a common *zero* time.

4.5.2 Polite TDMA

Problem Definition. The simple solution presented above naturally adapts to changes due to node removals and it is already able to handle node additions: new nodes simply begin pulsing, and the schedules in the 2-hop neighborhood scatter accordingly. However, in order to recognize such changes, nodes must periodically listen for an entire oscillation period to perceive pulses not yet integrated in their local knowledge. Moreover, while nodes *can* join the network by simply starting to pulse, it is likely that this new pulse will interrupt an ongoing communication. As our goal is to support communication guarantees, such interruptions are not acceptable.

Classic Solution. Non disruptive changes to the communication scheme are usually handled by effectively *reserving* a slot, known to all nodes, for control messages such as the intention to join the schedule. In classic TDMA protocols, time synchronization is already required to align the beginning of the time frames on all the nodes. This service can therefore be exploited to declare the beginning of each frame for coordination. However, time synchronization is commonly defined as a stand-alone service, independent from the solution in charge of defining the communication schedule.

Through the PCO lens. We first recall that no synchronization is required in the solution presented above. In fact, a node wanting to join the system can learn the current communication schedule by listening to the pulsing of its neighbors, and independently decide the best placement of its own pulse. However, this solution does not avoid collisions with ongoing communication. Therefore, we adopt the approach employed in classic TDMA solutions, allocating a coordination slot, common across all schedules. Augmenting our MAC-solution with a synchronized slot is trivial: we require nodes to pulse twice. One pulse is still controlled by the scattering equation, while the other is updated according to a synchronization coupling function, exploiting Equation (4.3).

The synchronized pulse identifies a small, fixed size slot, shown as a shaded region in Figure 4.5(d), during which all nodes must listen, and any node can transmit using a simple CSMA approach. For a node to join the schedule, it transmits in the control slot a deferred pulse indicating its *preferred* location in the schedule. E.g., to maximize the assigned slot size, a node should identify the largest distance between two consecutive pulses in the schedule, and place its new pulse in the middle. All nodes incorporate the pulses indicated in the control messages, and after a stabilization period, the new node can begin to transmit at its pulse-assigned slot without disrupting ongoing communications. Further, notification of both scattering and synchronization pulses is combined and transmitted at the beginning of the assigned transmission slot. This, as previously noted, naturally avoids collisions among pulse notification messages and makes the cost of the synchronization procedure negligible.

4.6 Bringing Reins-MAC into Real World

As discussed in Section 4.2.2, the scheduling mechanisms introduced must account for the typical characteristics of low power wireless communication. Several real world experiences [74, 57] reported the presence of links with intermediate reception probabilities, asymmetries, and variability over time. In this Section, we describe how REINS-MAC faces these challenges.

4.6.1 Intermediate Links

In low-power wireless the presence of intermediate links is common. As a consequence, the existence of a link between nodes is insufficient to guarantee the delivery of a message. In contrast, in ideal scenarios links are perfect; all messages sent along a link are always either received or lost. Notably, intermediate links may lead to transmission failures despite a perfect schedule.

Manifestation in REINS-MAC**.** Scheduling, as a consequence of the PCO nature, is a continuous process that runs at every oscillation epoch. The coupling function uses information from the pulses collected in the previous epoch to compute the new pulsing phase. If all pulses from the neighbors are always received, the local schedule is up to date with the changes in the interference set and the node can follow it straightforwardly.

However, if a pulse notification gets lost, the local schedule contains wrong information. E.g., we can assume a small network made by two nodes, A and B, with an intermediate link connecting them. If a pulse sent by A is not received by B, B does not know if A's pulsing time changed and can only use the last information it has. Given ϕ_A^{t-1} and ϕ_A^t as the previous and current phase of A, the former is the time when B will wake up to wait for A's pulse whereas the latter is the one at which A will pulse. Due to the scattering coupling function, the current phase may move either ahead or behind the one of the previous round. If the phase did not change, B will be awake at the right time for A to pulse. If ϕ_A^{t-1} precedes ϕ_A^t, B will be awake before A's pulse; after some time, B will nonetheless receive the pulse notification. Instead, if ϕ_A^{t-1} follows ϕ_A^t, A will pulse before B will be awake to receive the notification. In this case, B will loose the reference to A's pulse possibly forever, resulting in a wrong schedule.

Through the REINS-MAC **lens.** As described in the example, the problem manifests when the current pulse precedes the prior ones from the same node, and its previous notification is lost by a neighbor. The local node, however, can identify the phase range in which neighbors that lost previous pulses would wake up waiting for the local notification. In fact, it knows the extremes in its pulsing history. If the pulse notification is sent at the latest phase in such a range, all the possible receivers are already awake. Similarly, all the possible interfering transmitters have certainly ended transmissions.

The notification message, therefore, includes both its delay from the actual pulsing phase and the information of the latest phase in the pulsing history. The former is needed to compute the schedule as described in Section 4.5 and to control the wake up time; the latter allows the receivers to go to sleep if no notification is received, saving energy in case of lossy links. Finally, to keep the schedule updated and limit the memory occupation, a `prune` timeout, of the order of few rounds, is used to remove no longer valid information and limit the size of the pulsing history.

4.6.2 Asymmetric Links

The statement on non ideal links can be extended to asymmetric delivery probabilities, where nodes cannot assume reciprocal knowledge of one another.

Manifestation in REINS-MAC**.** Extreme asymmetric links can affect the correctness of the protocol. E.g., consider that A cannot hear B, but A directly precedes B in B's schedule. In this case, A's transmission slot will not end when B's begins, causing interference. Moreover, the asymmetry can affect the synchronization, resulting in the impossibility to align the coordination slot in the extreme case of an isolated node that cannot hear any pulse.

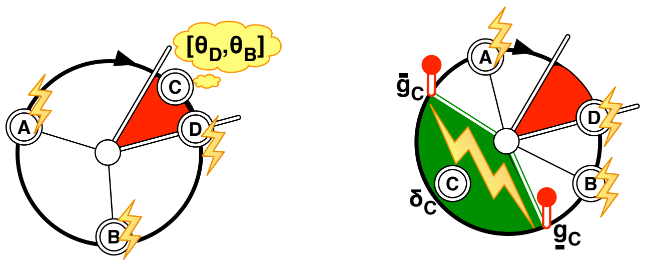
Through the REINS-MAC **lens.** Fortunately, our protocol already sends connectivity information, which can be exploited to clearly identify asymmetric links. In fact, each node provides its own neighborhood view inside the pulse. If an asymmetric link exists, a node B will not see itself in the schedule of a 1-hop neighbor, A, and can realize the impossibility to communicate from B to A. In this case, if A precedes B in B's schedule, B can relocate its slot to a different position in the schedule through a *jumping* operation: the node stops pulsing in its own slot, pulses in the coordination slot notifying the desired position, and after a while moves in such position. The node can start using the slot right after recognizing the moving of the pulses surrounding the desired position. Finally, for what concerns synchronization, we both discard the pulses related to the coordination slot received along asymmetric links and avoid using such a slot on nodes without perceived neighbors.

4.6.3 Link Variability over Time

The above link characteristics are known to vary with time. Seasonal and environmental changes can redefine the physical topology. This can also be forced by the physical addition, removal or relocation of nodes.

Manifestation in REINS-MAC**.** The schedule is affected when the neighbor set changes. A node leaving the interference set is automatically removed from the schedule thanks to the `prune` timeout and its slot is overtaken by the surrounding nodes in the schedule. A node physically added to the network starts notifying in the coordination slot its desired pulsing position, and is included in the schedule automatically. The problem manifests when temporal variations create new links previously unavailable. In this case the two end points do not have any information about the pulses of one another, causing possible interference.

Through the REINS-MAC **lens.** Similarly to the case of physical addition, each node can notify its own pulse in the coordination slot. Given the fact that the coordination slot is globally agreed upon and all the nodes are listening, a notification can then be easily integrated in the schedule. Two approaches can be used to trigger the notification in the coordination slot:



(a) Reordering for Latency. (b) Guarding Bandwidth.

Figure 4.6: REINS-MAC extensions to support QoS.

proactive or reactive. Our proactive scheme is based on nodes periodically notifying their pulse. The period frequency depends on the local density of the network and is tuned to decrease the probability of collision in the coordination slot. Our reactive mechanism adds sequence numbers to the sent messages; losses in the sequence of messages are counted and notified in the pulse for each possible sender. If a node receives a pulse indicating a high number of missed messages corresponding to itself in the recent epochs, it schedules the notification of its own pulse in the coordination slot.

4.7 Reins-MAC: QoS Served with a Side of Anarchy

The protocol described thus far offers a basic medium access scheme, able to adapt to topology changes in a fully decentralized fashion. As analyzed in Section 4.8, the computed communication schedule offers advantages with respect to typical TDMA approaches. The foundation of its effectiveness is the ability to autonomously exploit local properties such as network density instead of requiring explicit tuning, often resulting in worst case configurations. However, the resulting schedule remains independent of application requirements.

QoS support at the MAC layer is usually driven by latency and bandwidth constraints arising from the application. These requirements are typically seen as difficult to support and yield rigid solutions. In this section, we finalize the definition of REINS-MAC by integrating QoS requirements. Latency control is supported with the same simple, but effective *jumping* technique introduced in Section 4.6.2. Bandwidth reservation exploits our *pulse duration* extension, defining a new coupling function to exploit it. The resulting solution enables each node to obtain communication quality guarantees in a flexible, unsupervised, anarchic manner.

4.7.1 Local latency control

Problem Definition. The first QoS parameter we address is latency, defined as the time for a single message to move from source to destination. In this chapter our focus is on the MAC layer, which has no knowledge of the data route. However, if we assume a higher layer in the protocol stack provides this information, the MAC layer should minimize the forwarding latency by placing the sending slot as soon as possible after the slot when the message is received.

Through the PCO lens. In the scheduling solution described thus far, the sequence of node pulses is fixed. Put another way, a node is constrained to always pulse between the same `PREV` and `NEXT`. Our solution is simply for a node to remove itself from the schedule, then re-enter the schedule at its desired location, effectively *jumping* to a different position. This is shown in Figure 4.6(a) as node C removes itself from its original location in the schedule between A and B, and uses the control slot to announce its new, desired position between nodes D and B.

4.7.2 Bandwidth reservation

Problem Definition. The mechanisms described reserve one sending slot to each node. Because slot sizes are variable, the allocated slot may not satisfy application needs. In principle, we could exploit our basic scheme and allow each node to pulse more than once, increasing the number of allocated slots. While this solution likely increases the total allocated bandwidth, further modifications to the schedule by other nodes can arbitrarily affect slot assignment, and no guarantees are possible. Moreover, this solution increases both the number of times each node in the neighborhood wakes up and the overhead in terms of messages exchanged.

Through the PCO lens. A better solution is to increase the single, assigned slot to the size requested at run-time. To accomplish this, we integrate a pulse duration equal to the requested slot size, δ_i, and indicate the beginning and end of the pulse with *guarding* phases, $\underline{g}$ and $\overline{g}$, spread equal distance from the center of the pulse. These phases are described as:

$$\underline{g}_i^k = \phi_i^k - \frac{\delta_i^k}{2} \text{ and } \overline{g}_i^k = \phi_i^k + \frac{\delta_i^k}{2} \tag{4.7}$$

. While equation 4.7 is expressed in terms of the local variable ϕ_i^k, the value of the guarding phases can be similarly computed at any observing node j in terms of $\theta_{j,i}^k$. To accommodate duration, the coupling function expressed in Equation (4.4) is applied with respect to the guards:

$$\phi_i^{k+1} = (1-\eta)\phi_i^k + \eta(\underline{g}_{PREV(i)}^k + \overline{g}_{NEXT(i)}^k)/2 \tag{4.8}$$

. As depicted in Figure 4.6(b), by extending the pulse duration of C to begin at the first guard, $\underline{g}$, and end at the second, $\overline{g}$, the slots of A and B shift, leaving a slot of at least size δ_C that provides the requested bandwidth guarantee to C. Setting the duration to the minimum, zero, reduces this to Equation (4.4). On the other side, the total reserved duration must be less than the distance between the pulses of `PREV` and `NEXT`, A and B in our example, as placing the guards beyond these limits results in incorrect pulse scattering.

Notably, the reservation effectively *steals* bandwidth by forcing the surrounding pulses to move away from the guards, ultimately losing their own non-guarded resources. This pulse shift-

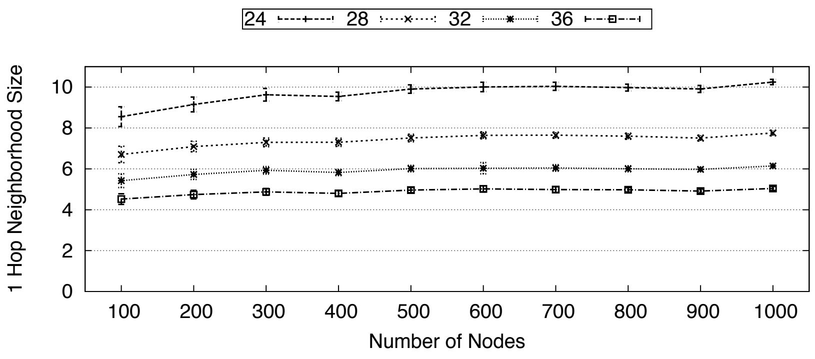


Figure 4.7: Average 1-hop neighborhood in the simulation settings.

ing process propagates recursively among neighboring nodes, *balancing* the cost of the reservation throughout the surrounding area. Additionally, as previously noted, the maximum allowed reservation is restricted by the original pulses of `PREV` and `NEXT`, while in theory additional bandwidth *may* be available. As shown in Figure 4.6(b), once the guards have been placed by C, the subsequent movement of A and B does make additional resources available that can be reserved by further increasing δ_C. In a sense, this iterative procedure refines the actual resource availability.

4.8 Evaluation

In this Section we evaluate Reins-MAC. After describing the simulation settings, we turn to the fundamental properties of the protocol, then to its ability to support QoS. Finally, we evaluate the implementation of Reins-MAC on real hardware.

4.8.1 Simulation Settings

All evaluations are performed with the Castalia 3.1 [3] simulator. Results are obtained for runs over 40 different uniformly random node placements in square fields, and show standard deviation. We control the size of the field to yield an average target density of between 4 and 10 1-hop neighbors, as shown in Figure 4.7. Each target density is described by the average distance between nodes: the larger the distance, the smaller the size of the neighborhood. We set the duration of the oscillation period (frame length) to 10 seconds, with a coordination slot of 250 ms. The setting of these values is dependent on the actual application scenario, as discussed in Section 4.2.1. Further we use a coupling strength of $\eta = 0.3$; as discussed in [25, 19], low values of this parameter decrease the convergence speed but increase the stability and robustness of the convergence.

One of our motivations in selecting Castalia is its realistic channel and radio models and clock drifts, all of which clearly affect the behavior of Reins-MAC. Clock drifts have the potential to de-synchronize a synchronized solution. Fortunately the very nature of PCOs supports and manages this. We set the `prune` timeout, described in Section 4.6.1, to 10. Notably, a too large `prune` value results in over provisioning, while a too small value affects convergence. We experienced this in simulations by setting the `prune` timeout to 5 and observing continuous changes to the schedule be due to links with very high loss rates. Finally, each node defines a minimum slot size of 50ms; if the slot shrinks and its length does not respect the limit, the node relocates its own pulse.

4.8.2 Evaluating Core Reins-MAC Properties

To verify the effectiveness of the scheduling mechanisms, the main metrics we consider are gain over potential competing solutions, convergence, duty cycling and ability to support topology changes such as the addition of nodes.

Gain. As shown in Section 4.2.3, the ability of Reins-MAC to dynamically establish the slot size based on network density produces substantial gains over a typical TDMA scheme such as `MAX` that selects a single, fixed slot size for the entire network based on the maximum 2-hop density.

For networks of high density, Reins-MAC achieves an average slot size double that of `MAX`, as shown in Figure 4.2. Even larger gains, up to 200%, are seen for less dense networks. In fact, Reins-MAC by default does not change the sequence of pulses. Once a pulse is positioned in the schedule, it is constrained by `PREV` and `NEXT`. The initial position is chosen randomly and, therefore, in more dense scenarios it is more likely to be placed in a locally "poor" location. We study the potential benefits of changing pulse location in Section 4.8.3. In any case, establishing slot sizes based on local constraints produces clear advantages over the fixed slot size `MAX` solution, even when this fixed size is identified optimally.

Interestingly, Reins-MAC also offers gains over the `MEAN` solution. We initially expected Reins-MAC to essentially *find* this size and allocate it. Instead, when considering nodes at a 2-hop distance, a 2-hop neighbor can communicate concurrently to a 1-hop neighbor, as long as they are in different interference sets. Interestingly Reins-MAC identifies these allowed overlaps, resulting in the gains over `MEAN`.

Convergence. Reins-MAC creates a schedule through the iterative update and exchange of pulse information. To accommodate clock drifts and connectivity variability, this process continues for the lifetime of the network, constantly making minor modifications to the schedule. Despite the continuous movement of the pulses inside the schedule, each single node can start using its own transmission slot right after notifying in the coordination slot its intent to pulse. We force a node to wait 3 frame before using a new slot, a value which has demonstrated sufficient in all our simulations.

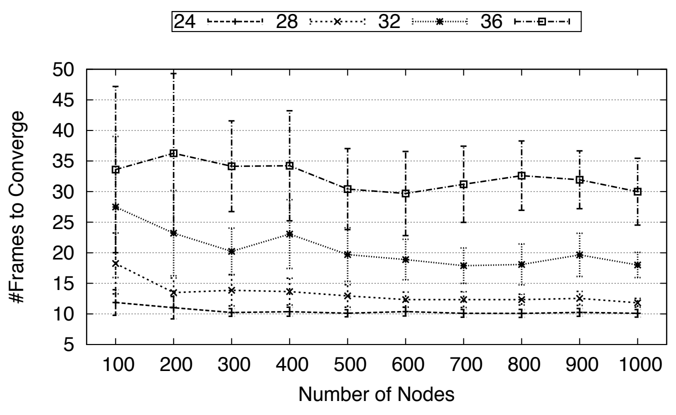


Figure 4.8: Time to establish 95% stable slots with a variation of the beginning of the slot less than 5‰ of the frame length.

To give an index of the time the solution requires to stabilize, we offer two definitions of stability. First, we consider a schedule stable when the location of 95% of the pulses, and therefore of the beginning of the slots, in the network vary less than 0.5%, or 5‰, of the frame length. It gives an indication of the time the scattering function takes to refine the schedule. It is worth remembering that even if the pulses move, the solution presented in Section 4.6.1 both keeps the scheduling correct, and, more importantly, allows the uninterrupted usage of all the slots to transmit application messages. Figure 4.8 shows the results from networks with all nodes started at once, without any initial synchronization provided. According to this metric, networks converge on average in less than 40 rounds, and more dense scenarios with smaller slots converge much faster. Notably, within a given density (a single line in the figure), the time to converge is consistent across networks.

Focusing on the beginning of each slot ignores variations in the slot length arising from changes in the beginning of the slot of the next node in the schedule. Therefore, we consider a more comprehensive stability property, namely when 95% of the slots in the entire network vary less than 5‰ of the frame length. As this metric is affected by more changes in the schedule, Figure 4.9 shows an increase in the converge time. Nevertheless, if we consider that we are starting all the nodes at once and that also the ongoing synchronization process moves the coordination slot in the schedule, these convergence times are quite acceptable. The remainder of this chapter uses this, more strict stability metric.

In both these convergence plots, sparse networks take longer to converge than dense networks. This is due to our definition of convergence that depends on variations w.r.t. the total frame length. E.g., consider that slot sizes in sparse networks are larger than in dense scenarios. With large slots, variations either in terms of the beginning (Figure 4.8) or the size (Figure 4.9) are likely to be large, influencing convergence more. Nevertheless, we maintain these convergence

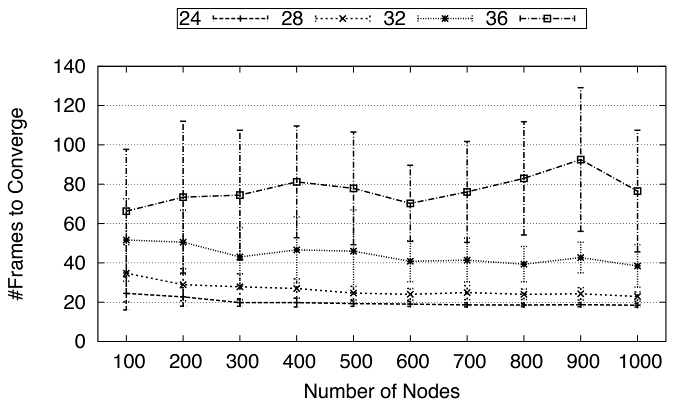


Figure 4.9: Time to establish 95% stable slots with a variation in slot size less than 5‰ of the frame length.

metrics as the behavior of communication on top of Reins-MAC depends on the absolute variations in the slot.

Duty Cycle. Duty cycling with a schedule continuously changing is challenging. In Figure 4.10, the time, in which the radio is active, is shown. In general, decreasing the density reveals an increase in the duty cycle. This behavior is due to the presence of intermediate links causing pulses to be missed. If the pulse information is old, the node must stay awake waiting for the current pulse, which will happen certainly after, as described in Section 4.6.1. However, if the pulse is not received again due to a link loss, the node stays awake until the pulse of the node following in the schedule. With a decreased density, the distance in time between consecutive pulses is bigger and therefore the energy wasted is more. In any case, the duty cycle assesses between 7% and 9%, with 2.5% duty cycle spent in the coordination slot. In achieving these results, the frame length as a clear impact; lower duty cycles can be achieved by increasing the period of the schedule.

As discussed in Section 4.6.1, nodes wake up before the sender pulses, to prevent intermediate links from affecting the ability of receivers to follow the pulsing node. As shown in Figure 4.11, the higher the density, the more energy is wasted while waiting for such a pulse. As density increases, it also increases the number of receivers and the number of pulses each node must wake up to receive. The result is an higher energy cost, about 2% of the duty cycle.

Topology Change. While the previous discussion addresses the ability of Reins-MAC to support network variations arising from wireless communication, here we consider topology changes caused by the addition of nodes at uniformly random locations. We consider the time to stabilize the schedules after a given percentage of nodes are added to a previously stable solution. Given the consistency of the previously shown results and the clear behavioral differences based on density, for the remainder of the evaluation we focus on networks with a fixed number of

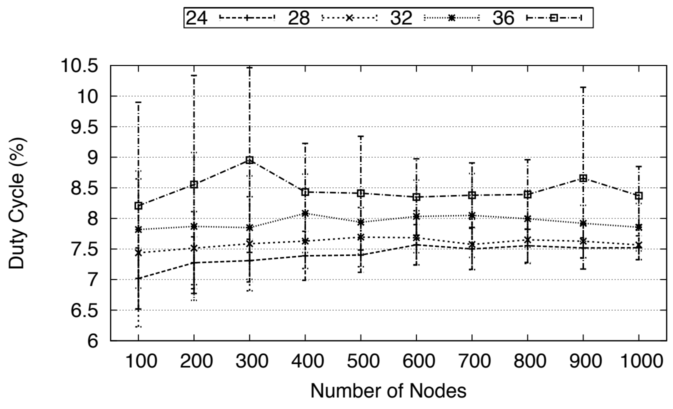


Figure 4.10: Network duty cycle.

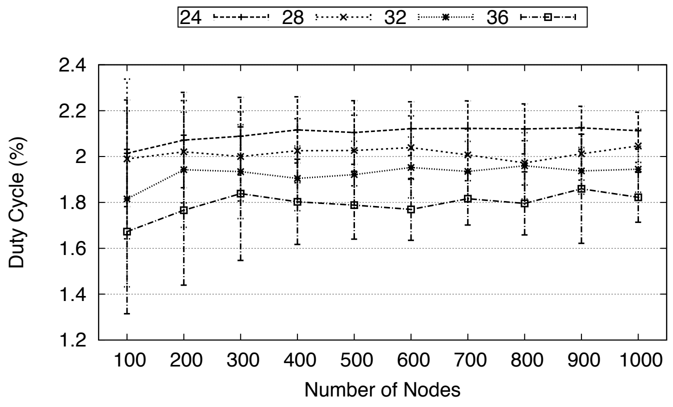


Figure 4.11: Duty cycle wasted by receiver nodes while waiting for the actual pulse.

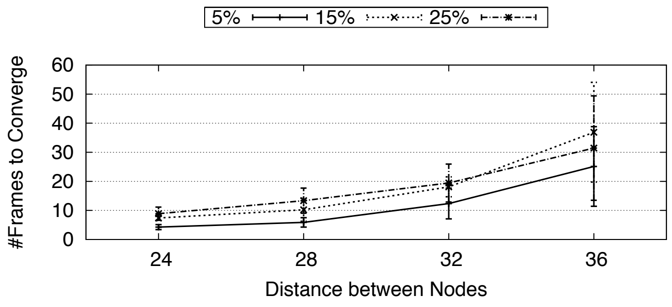


Figure 4.12: Stabilization of the system after node addition.

nodes and varying densities.

From a network composed of 500 nodes placed as in the previous experiments, we randomly select 5%, 15%, and 25% of nodes which are started only once the other nodes in the network have defined a stable schedule. Figure 4.12 shows that the system stabilizes, in all scenarios, within half the rounds required when the whole network is started from scratch (Figure 4.9). As expected, scenarios that add fewer nodes converge slightly faster, as do dense networks.

4.8.3 Reins-MAC support for QoS

We now turn our focus to the ability to support QoS constraints coming from higher layers in the network stack. This section focuses on the contributions at the MAC layer, using simplified high-level requirements and mechanisms.

Schedule reordering. Section 4.7.1 discussed the possibility for a node to change its location in the schedule to meet latency requirements. The same *jumping* mechanism can be used to increase the allocated bandwidth without an explicit reservation (which we show next). By doing so, we demonstrate both the use of jumping to increase system utilization, but also show the general behavior of the jumping technique.

To analyze the benefits of jumping, we study networks of 500 nodes, allowing some nodes to change location in the schedule. Intuitively, the largest bandwidth gain can be achieved by selecting the largest slot in the local schedule. To reduce the probability of neighboring nodes choosing the same slot, we force only one node to jump in each frame.

The top of Figure 4.13 shows the increase in the slot size of nodes applying this jumping strategy (termed: Jumpers). The deviation of the results is quite large due to the high variability of the opportunities to jumpers at the time of changing slot. In fact, it is possible that a jumper has a slot either larger than average or, vice versa, significantly smaller. A more complex jumping strategy can take this into account in selecting which nodes move. Additionally, our metric to identify the slot to jump to does not guarantee that a node will increase its slot size, further affecting the individual results. Nevertheless, the overall trends can be analyzed.

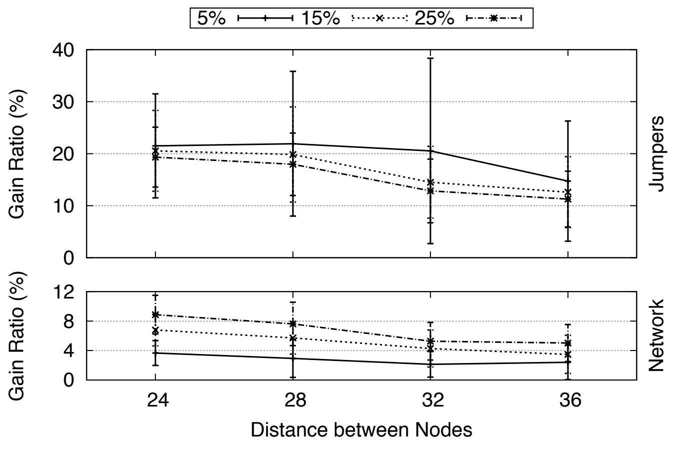


Figure 4.13: Gains achieved by jumping.

In dense networks, jumping is likely to offer larger increases in slot size, around 20%. This indicates that nodes really are being *trapped* in poor, small locations in their schedule. In practice, this means that if bandwidth needs are not being met, a node may opt to jump to a new location. In sparse networks, nodes already have quite large slots, therefore the gains achievable by jumping are proportionally smaller. While clearly most of the jumpers gain bandwidth, one may be concerned of the negative effect on other nodes. However, for all scenarios studied, the average, network-wide slot size increases as shown in the bottom of Figure 4.13. Some of this gain is attributed to the larger slots of the jumping nodes, but one must also consider that every jumping node leaves behind an empty slot. REINS-MAC adjusts the schedule to allow the neighboring nodes to absorb this slot, generally allowing gains.

Bandwidth reservation. To evaluate bandwidth reservation, we build a path with a random walk. Each node makes a reservation and forwards the request to one neighbor not yet in the path. On reception of the request, the node reserves three quarters of the distance between PREV and NEXT. As the scattering function brings the pulse in the middle between those pulses, the reservation is equivalent to one and a half the currently owned slot.

Figure 4.14 shows the average time for a node to achieve a slot at least as big as the reservation. On average, the allocation takes a little more than 10 frames. This depends on the mechanism described in Section 4.6.1. The node keeps the beginning of the slot at the latest pulse in its own pulsing history. By setting the pulsing history to 10, the slot can be effectively used only once the pulse references active before the reservation are pruned. Interestingly, once those references are removed from the history, the slot is already big enough to support the request as both PREV and NEXT move as soon as the request is submitted. In some cases, NEXT moves fast enough to support the reservation in fewer rounds.

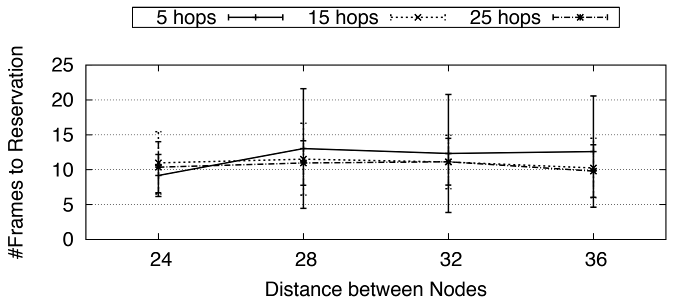


Figure 4.14: Average time to obtain the requested slot size.

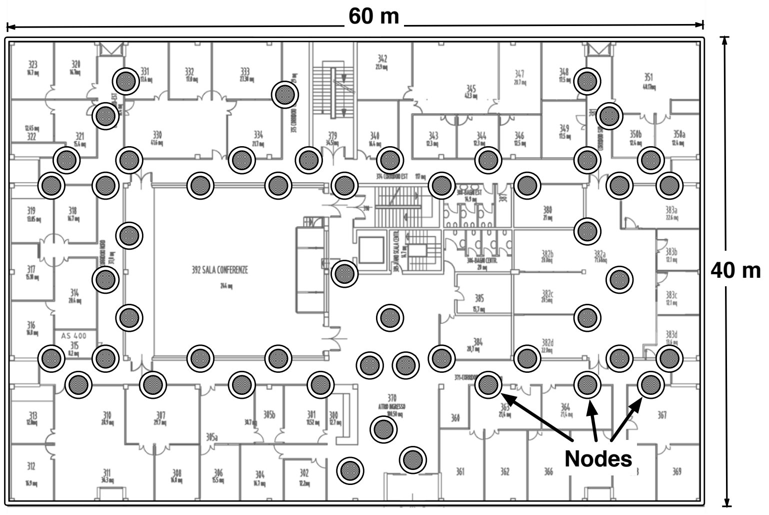


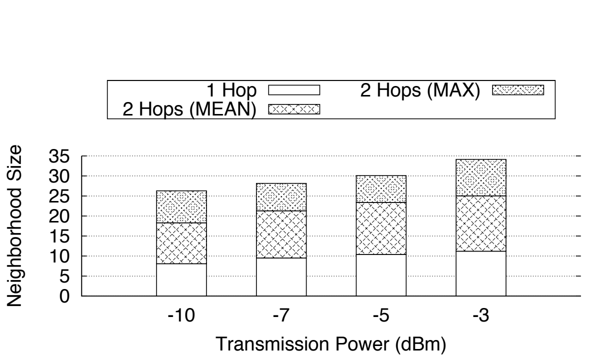
Figure 4.15: Map of the testbed.

4.8.4 Implementation

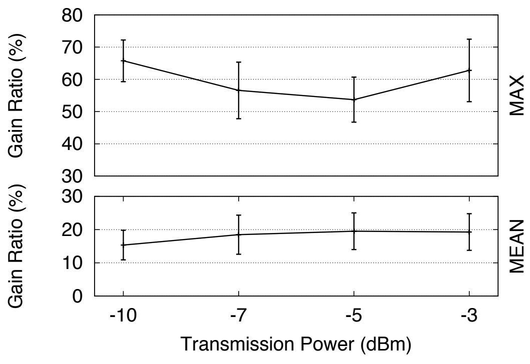
To verify the feasibility of the presented solution, we discuss the implementation of Reins-MAC on real hardware and the evaluation of its performance in a testbed.

Description of the code. We implemented Reins-MAC on top of TinyOS v2.1.1 [77]. The code requires around 12 KB of ROM and 1.5 KB of RAM, most of which devoted to store and maintain the neighborhood information. In our experiments we defined the neighborhood table to contain 40 records. Despite our choice to develop the solution specifically on the TelosB [66] platform, the one available in our testbed, we avoided any platform specific optimization. The only exception is the message timestamping at transmission and reception. The code, available for download at `*.sf.net`, is therefore easily portable to different hardware platforms and room is available for more optimizations.

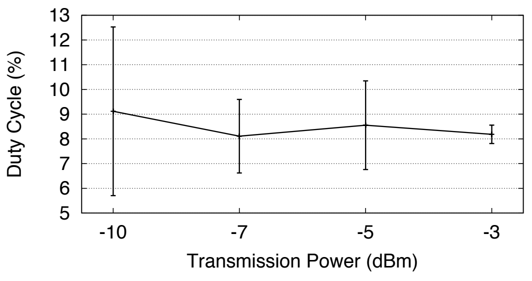
Experiments setup. At our institution, we installed 50 TelosB nodes in an area of about



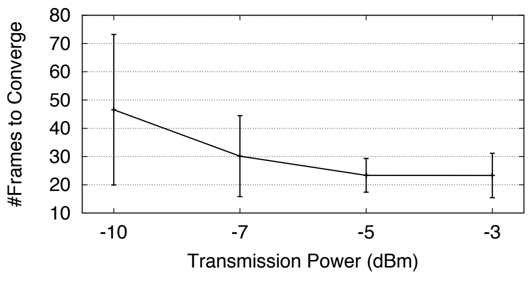
(a) Neighborhood.



(b) Gain.



(c) Duty Cycle.



(d) Convergence.

Figure 4.16: Results from experiments run in the testbed with stable networks.

60x40 m^2. The devices are located as depicted in Figure 4.15 underneath the corridors on poles that lift the tiles from the concrete floor. Each node is connected through an infrastructure of USB hubs and routers to a PC from which the testbed can be controlled.

The network density, using standard transmission power, is higher than the one we tested in simulations. Most of the results are influenced by edge effects. Therefore, we used lower transmission powers and vary them to test different densities. However, as shown in Figure 4.16(a), significant differences in the number of direct neighbors start to appear only using very low, unreliable, transmission power. Moreover, differently from the simulation setup, the network density is more homogeneous, with a smaller difference between the average and the maximum 2 hop neighborhood size.

Evaluation. The results, obtained by running 20 executions varying the time of node boot to force different scheduling, are shown in Figure 4.16 for stable networks along different transmission powers. The gain with respect to `MAX` and `MEAN` is lower than the one seen in simulations, as the network is smaller and more homogeneous. Nonetheless, the gain with respect to `MAX` is always more than 50%. The duty cycle assesses between 8% and 9%, with more variability for networks with more unreliable links. In terms of convergence time for the highest transmission powers tested, the results are in line with the simulations; however, decreasing the transmission

Operation	Metric	Avg	Stddev
Add (10%)	Convergence (frames)	11.45	6.2
Add (20%)	Convergence (frames)	15.21	5.1
Jump (10%)	Gain Jumpers (%)	13.43	12.5
	Gain Network (%)	2.67	3.8
Jump (20%)	Gain Jumpers (%)	12.75	11.9
	Gain Network (%)	4.99	3.41
Guard (5 hops)	Reservation (frames)	4.65	2.2
Guard (10 hops)	Reservation (frames)	4.45	2.1

Table 4.1: Results of testbed experiments with node addition, schedule reordering and bandwidth reservation.

power results in more unreliable links, which ultimately increases the number of rounds required to achieve a stable schedule, as shown in the case of -10 dBm.

In Table 4.1, we show results for experiments using -5 dBm transmission power, with addition of nodes, schedule reordering via jumping, and bandwidth reservation. The node addition requires about 5 rounds on average more than simulations to stabilize. As the nodes are not uniformly displaced in the area, the addition of nodes in key locations can connect previously disjoint neighborhoods, with a longer convergence time required to merge the schedules. The results obtained with schedule reordering follow the trend of the simulations; the reduced gain is due to the small and homogeneous neighborhood, with limited chances to find significantly better positions. Finally, reserving bandwidth takes less time because the slots are big enough to allow `NEXT` to move further away from the requester, in fewer rounds.

4.9 Is Reins-MAC really different?

We classify Reins-MAC as a TDMA communication scheduling protocol. As such, it clearly contrasts commonly employed CSMA-based contention solutions in which the channel is allocated only when data must be sent. Prominent examples are B-MAC [65] and X-MAC [4] where the carrier is periodically sensed for incoming communications and different forms of preambles are used to reserve the channel before transmission. Despite the simplicity that makes them suitable in practice, they cannot provide deterministic guarantees either on channel availability or on collision avoidance.

In TDMA approaches, instead, collisions are prevented by construction. This comes at the cost of building the communication schedule for the entire network, either in a centralized, as in PEDAMACS [23] or LiteTDMA[72], or distributed fashion. Examples of the latter are LMAC [81], TRAMA [68], and CRANKSHAFT [29], where the slot owner is elected depending on a priority index resolving contending requests. All these solutions rigidly define the reservation scheme, require explicit ownership mechanisms, and limit adaptation to variable communication requirements.

Alternative solutions are based on the pulse scattering algorithm introduced in DESYNC [19] and Wake-Up Scattering [25]. In [18] and [58], the DESYNC algorithm is extended to solve the hidden terminal problem, obtaining the same solution we describe in Section 4.5.1. The works present only a preliminary discussion and evaluation of the solution. Moreover, they are limited to building the communication schedule, without providing radio duty cycling. Reins-MAC, instead, not only enables nodes to sleep but also evolves the basic solution to make changes to the schedule possible without interrupting ongoing communication, ultimately, providing the application with full control of the resource allocation.

Once a TDMA schedule is defined, hybrid approaches are also possible. DRAND [71], the TDMA distributed scheduling protocol used in ZMAC [70], achieves pseudo-optimal collision free slot assignments. On top of such schedule, a CSMA policy is defined to hand over the slot from the actual owner to neighbors, increasing channel reuse. Notably, Reins-MAC does not prevent the definition of specific policies to handle the communication inside each slot, making the definition of a hybrid behavior possible.

Next we discuss the TDMA recognized limitations as presented by the ZMAC authors, describing how Reins-MAC, while retaining a pure TDMA nature, faces them.

Building an Efficient Schedule. Computing a collision-free communication schedule maximizing bandwidth utilization is complex. Therefore, usually topological information is collected centrally where a schedule for the whole network is computed. Data traffic patterns can also be taken into account. Similarly, in the IEEE 802.15.4 standard [34], nodes organize in Personal Area Networks (PANs). In each PAN, the coordinator controls the access to the communication medium, assigning time slots to nodes upon request. These techniques usually result in limited scalability.

With distributed scheduling, the communication frame is discretized in slots of equal size. The nodes notify their intention to use one or more slots together with the neighborhood information. In LMAC, the first node notifying interest in a slot becomes its owner; in TRAMA and CRANKSHAFT, a system-wide election function identifies a slot leader among multiple contenders. The efficiency of these solutions is bounded by the properties of such functions, restricting their applicability to different applications and topologies. Otherwise, generic solutions require high computational complexity, as in DRAND. In any case, the fixed slot and frame sizes confine the optimality of the solution to the feasible schedules. Selecting the right parameters fitting all the possible system states is crucial.

Reins-MAC defines a suboptimal schedule. Each node selects its local position in the schedule. This choice does not result in a predefined amount of bandwidth, and the final achieved slot length is a consequence of the movements of the surrounding pulses. This constraint does not guarantee optimal channel reuse. However, the fixed scheduling ordering can be modified by relocating pulses. The efficiency of the mechanism employed is grounded in the anarchic control of the pulse, which is automatically accounted for by the nodes in the same interference set. Properties of the schedule can be changed by the application, which is ultimately responsible for efficient communication resource assignment.

Clock Synchronization. TDMA techniques usually assume the execution of a synchronization protocol, such as RBS [22], FTSP [51], or TPSN [24]. Depending on the synchronization requirements, such protocols are run frequently enough to guarantee slots alignment, at a considerably high cost. In contrast, Reins-MAC aligns the schedule implicitly, by using the reference provided by the surrounding pulses. The definition of a coordination slot, used only to notify changes to the schedule, exploits a synchronization mechanism that naturally integrates in the protocol. The cost of the procedure, a field in the pulsing message, is almost negligible and explicit in the scheduling mechanism. Moreover, the merging of the synchronization pulse with the one used for scattering communication solves the limitations of PCO synchronization mechanisms [83], e.g., the collision of simultaneous pulsing.

Low Utilization with Low Contention. In case of low contention, generic TDMA solutions can result in low utilization due to the assignment of a slot to each node in the network. DRAND relies on CSMA to reuse an unused slot. In LMAC, TRAMA, and CRANKSHAFT, nodes can renounce to their own slots, making these available to others. Similarly, Reins-MAC can be extended to allow nodes to leave the schedule, avoiding unused resource allocation. In addition, Reins-MAC lets a node define the slot size it uses with the duration property; this mechanism allows the application to guide the utilization of the resources depending on the current needs.

High Delays. TDMA fixes the position of a slot and repeats the same frame, enabling the sender to communicate only in defined times. In between owned slots, no information can be sent, forcing increased delays in communication. The application cannot dictate any latency requirement, as the underlying scheduling mechanisms do not support changes on demand. This limitation is solved in Reins-MAC, enabling the application to define position and duration of a slot.

Radio Interference outside Communication Range. One of the advantages of TDMA is the ability to schedule a collision-free communication. Unfortunately, the communication range can be different from the interference one [86], affecting the identification of the interference sets. This problem is not explicitly faced in any of the previously mentioned protocols, if not by either the extension of the schedule horizon to span 3 hops or the usage of CSMA in hybrid approaches. Reins-MAC, instead, tries to identify error in the definition of the interference sets by detecting anomalous message losses and signaling this condition to the sender.

Time Varying Topologies. In common TDMA solutions, the strict framing needs atomic actions for slot assignment. The reevaluation of the entire schedule is required to face changes.

Time varying topology are common in WSNs. In LMAC, TRAMA, and CRANKSHAFT, those modifications require the reevaluation of the prioritization indexes, if possible. In DRAND the whole network needs to be rescheduled. In REINS-MAC, instead, the resources owned by a node no longer in the interference set are automatically acquired by the ones surrounding that node; in a similar way, a node entering the interference set steals resources to make room for itself in the schedule. Temporary inconsistencies are possible but are automatically fixed locally by the protocol itself.

4.10 Identifying the Limitations

REINS-MAC does not pretend to be the panacea for all the problems involving communication. In this section, we discuss its limitations, some of which originate from its TDMA nature, whereas others are intrinsic to the introduced scheduling mechanism itself. Their identification highlights the applicability of the solution, the tradeoffs, and the possible future directions of research.

Time and Message Complexity. The scheduling mechanism is a continuous process that lasts the lifetime of the network. This enables the high flexibility previously discussed, which allows the adaptation to both changing topologies and variable application requirements. Therefore, there is no stable convergence of the scheduling and it is unclear how to analyze the time complexity of the solution in real scenarios.

For what concerns the complexity in terms of messages, to support responsive adaptation to changes, each node must keep notifying its current view of the schedule at the beginning of the transmission slot, as well as listen to the others' schedule information. The amount of information exchanged at each round clearly depends on the number of elements in the schedule that need to be notified, which is ultimately dependent on the size of the direct neighbors. The more dense is the network, the more messages each node must either send or receive at the beginning of a slot (e.g., in our current implementation 10 neighbors can be notified in a message of 100 bytes, in addition to the information about the transmitter itself).

State Maintenance. The information about the whole schedule must be maintained in memory. It essentially requires to keep and update data for each and every node in the 2-hop neighborhood. The minimum amount of information needed is the tuple ⟨*source identifier, distance, duration, pulsing phase, pulsing history flags*⟩, which in our implementation requires 12 bytes. If the employed OS does not support dynamic memory allocation, a predefined maximum number of records must be stored in memory. It results in waste of memory and limitation on the actual adaptability of the solution to the current scenario as constrained by the worst case identified a priori.

Fairness. A recognized property of common TDMA solutions is fairness. This is guaranteed by construction as all the slots have the same size and, therefore, each node is allocated the same amount of communication resources. In Reins-MAC, instead, the nodes own transmission slots of different sizes depending on the local density and the position in the schedule. This clearly produces an unfair scheduling. The solution, however, does not force any node to remain trapped in the current state. In fact, the ability to change the slot and reserve some amount of bandwidth can change the balance of the resource allocation. In the specific case of fairness, if it is required by the application, each node can simply be asked to reserve a minimum amount of bandwidth, to guarantee a minimum level of fairness.

One Parameter: Frame Length. Despite Reins-MAC removes one of the fundamental parameter of common TDMA solutions, i.e., the slot size, the length of the schedule frame still needs to be identified a priori. This parameter contribute to define the actual transmission latency, the supported traffic load and the resulting duty cycle. E.g., in the case of very low traffic and no latency requirements, a long frame would dilute the protocol overhead over a long period of time, decreasing the resulting duty cycle. This change would, however, also reduce the speed at which the solution can react to changes to the schedule. With Reins-MAC, these tradeoffs must be still taken into account in the tuning of the solution before deployment.

Effectiveness Depends on the Traffic. The traffic workload is one of the important factors defining the effectiveness of the achieved solution. The difference between CSMA and TDMA is clear: the lower the traffic, the more impact has the constant overhead of scheduled mechanisms. CSMA is very efficient when the contention on the channel is low; as soon as the traffic increases, the channel becomes too busy to handle all the traffic, with an increasing number of messages lost in collisions and more energy spent in coordinating the access to the channel. In the case of TDMA solutions, the overhead of building a collision-free schedule is constant over time. As the traffic increases, the probability of collisions does not vary, whereas the cost of handling communication decreases. In fact, the scheduling requires the same amount of overhead to handle an increasing number of application messages. However, if no communication is ongoing, the schedule must be maintained. With this respect, setting a proper frame length is crucial.

Dependency on the network characteristics. As Reins-MAC is a fully distributed protocol, it is affected by the ability of spreading in the network information required to maintain the schedule. If we consider partitioned networks, each individual entity builds an independent schedule. This holds in particular for the position of the coordination slot. If disjoint networks become able to communicate, e.g., due to addition of nodes or changes in the communication range caused by the environment, they may not be able to align the coordination slot. It is enough to ask the nodes connecting the two networks to stay awake so to reconcile the, otherwise independent, schedules. As of now, the protocol faces this problem with the same mechanism used to handle interference ranges. The detection of abnormal message losses can trigger both an extended overhearing period and eventually a relocation of the transmission slots.

Anarchy and flexibility transfer to the application. Reins-MAC moves the burden of controlling the allocation of communication resources to the higher layers. Both CSMA and TDMA solutions available in the literature either seize the channel in an application agnostic fashion [56, 81] or merge the application needs inside the protocol [67, 21]. In the former case, the application cannot control the resources allocated; in the latter, instead, the solution can be applied only to specific application profiles. The flexibility and anarchy of REINS-MAC, instead, asks the application to rein in the resource allocation and adjust the schedule, as it naturally emerges from individual choices, to better fit the overall network goals.

Only the application has the knowledge required to define what is the best resource allocation according to the service it must provide. In this direction, REINS-MAC pushes further the cross-layer architecture typical of WSNs, without constraining a priori the application. However, the simple send and receive interface provided by standard MAC protocols hides the wider range of possibilities offered by REINS-MAC. Furthermore, the addition of commands to monitor the schedule and apply changes to it increases the complexity of the application. To simplify it, an intermediate layer between the MAC and the application with a proper interface can simplify the development of system services, enabling the application to steer the protocol anarchy, ultimately leading to the full exploitation of the protocol potentials.

4.11 Concluding Remarks

Although the domain of MAC protocols for WSNs seems to be saturated with works covering nearly every letter in the alphabet [38], here we identified a clear direction that has not been explored: simultaneously offering both flexibility and guarantees. The evaluation of REINS-MAC clearly demonstrated its ability to outperform traditional TDMA approaches in terms of slot size allocation and flexibility to varying application needs. The determinism offered by REINS-MAC makes possible the careful design of the network, according to the application requirements. The explicit allocation of communication empowers the higher layers with the ability to steer the communication resources, to ultimately improve the resulting service quality provided to the final user.

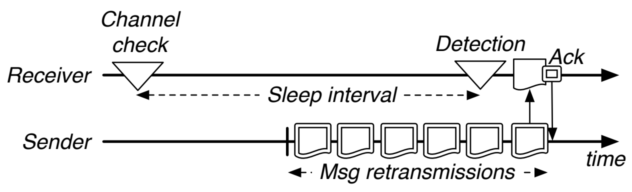
Chapter 5

Designing System Services with Communication Guarantees in Hand

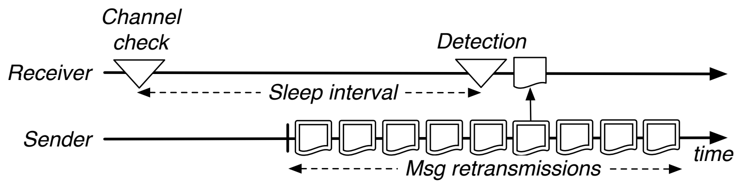
The availability of Reins-MAC opened new possibilities, not at our disposal at the time of designing and building the systems described in Chapters 2 and 3. As already mentioned in our motivation to define a new MAC protocol, the heterogeneous traffic requirements of the deployment in Torre Aquila and the timeliness ones in the closed loop control of light in road tunnels delineated two case studies for Reins-MAC. In this chapter, we reconsider the design of the system services employed in such monitoring infrastructures and evaluate the impact of Reins-MAC on their definition.

5.1 Medium Access Control: Enabling Communication

One of the main resources available in a WSN is, as the term itself states, wireless communication, which enables the technology itself. Unfortunately, the wireless medium is shared among all the nodes and concurrent communications can result in failures, unless properly scheduled. Communication is inherently broadcast, and messages sent simultaneously by different senders can collide at the receiver side even if the transmitters are not in each other's communication range. Finally, the radio is one of the main sources contributing to energy consumption and maintaining the radio active waiting for possible transmissions becomes a considerable energy expenditure. To solve the problem, several MAC protocols have been defined in the literature [39]. Among all the alternatives, B-MAC [65] and X-MAC [4] (or BoX-MAC [56] in their TinyOS combined implementation) are the most used solutions in real world deployments. In this section, after a brief description of their functioning, we summarize Reins-MAC, the protocol we introduced in Chapter 4 able to define a flexible communication schedule and provide guarantees.



(a) Unicast Transmission.



(b) Broadcast Transmission.

Figure 5.1: Low Power Listening functionalities.

5.1.1 Low Power Listening: Random Access to Communication

A low-power listening (LPL) [56] asynchronous MAC is the default medium access control available in TinyOS. Based on a Carrier Sense Multiple Access (CSMA) mechanism with clear channel assessments (CCA), LPL makes each node wake up periodically to check if there is any ongoing communication possibly of interest. Defined as *sleep interval SI* the period between consecutive receive checks, every *SI* a node turns on the radio and performs a CCA. If the channel is busy, the node keeps the radio active for the time required to receive the transmitted packet; otherwise, it goes to sleep, turning off the radio. The transmitter, once a message is ready to be sent, performs a CCA to assess that no other transmitter in communication range is using the channel. Then, if the channel is free, it continuously retransmits the message for the entire *SI* of the expected receiver.

As depicted in Figure 5.1, LPL differentiates between unicast and broadcast transmissions. In the former case, the receiver is asked to acknowledge the reception of the message. By leaving between message retransmissions the time for the acknowledgement to be sent, the sender can detect the successful delivery of the message and therefore stop the transmission. To facilitate the relay of trail of messages, the receiver stays awake for a small period after each reception to save the otherwise required preamble. If instead a broadcast transmission is required, no acknowledgement mechanism is used and the retransmission of the message concludes after the *SI* period of the receivers. In fact, all the nodes in communication range are expected to wake up in such an interval and check for ongoing transmissions.

The only interface provided by LPL to the application allows the configuration of the local *SI* and the remote one. The application can set those intervals during the network lifetime to better adapt the sleep interval to the ongoing traffic, as the two are correlated in defining the effective duty cycle of the system. The simplicity and robustness of the mechanism make it very attractive to be used in real world deployments, as we did both in Torre Aquila and in road tunnels. However, its random CSMA nature causes the impossibility to guarantee channel availability at any point in time. Moreover, its scope restricted to the 1-hop neighborhood is unable to prevent collisions at the receiver side, as it does not solve the hidden terminal problem.

5.1.2 Reins-MAC: Flexible Communication Scheduling with Guarantees

Opposite from CSMA approaches, periodic Time-Division Multiple-Access (TDMA) techniques schedule communication allocating dedicated resources to each node in the network. The determinism and the guarantees gained by using these approaches are usually overcome by the reduced flexibility, scalability, robustness, as well as the increased algorithmic complexity. In Chapter 4 we introduced Reins-MAC, a new TDMA protocol that arguably solves the limitations typical of this category of solutions, making it an appealing alternative to currently employed CSMA approaches.

As Reins-MAC defines a periodic schedule with time slots allocated to senders, each node is guaranteed the availability of dedicated communication resources. In the definition of the scheduling, each node has full control of the beginning of its own slot, which is not required to be aligned to any predefined grid, as instead happens in common TDMA protocols. Moreover, the individuals can apply changes and reserve available resources without any active negotiation. As discussed in Section 4.7, this mechanism offers anarchic support for both reordering of the scheduling to control latency and guarding of the slots for bandwidth allocation.

The form of resource reservation supported by Reins-MAC allows the solution designer to exploit the provided determinism in the definition of the higher layers protocols, as well as in the overall system design. Moreover, the protocol only adjusts the schedule to accommodate both topology changes and requests to apply modifications, without constraining a priori the resource allocation. Therefore, the network protocols can coordinate the exploitation of the communication schedule, as it best matches the application requirements.

5.2 Different Usage of Communication

Reins-MAC is placed right above the physical communication layer, providing deterministic access to communication to all the higher layer components interacting remotely. Moreover, the protocol provides an interface enabling each component to request changes to the resource allocation. Both the scheduled availability of communication and the possibility to steer resource allocation impact on the design of system services. In this section, we describe the general differences in using communication as fostered by the types of access to the medium we have experience with, as implemented by LPL and Reins-MAC. In Section 5.3, we discuss concrete examples of this influence on the design of network protocols.

From Pull to Push. Given that LPL tries to allocate communication resources immediately after a message is created by the application, a pull approach is favored in the implementation of a request-reply message exchange. The request is sent as soon as possible after it is submitted by the application to the MAC; similarly, the replier processes the query when it arrives and the answer, shortly submitted by the application to the communication layer, is delivered right after. As communication resources are allocated only on demand, the programmer is promoted to pull information on the basis of explicit needs.

In Reins-MAC, the timings of the communication are driven by the periodicity of the defined schedule. For this reason, once the application provides a message, it must wait for the beginning of the time allocated to it for transmission. This latency between the creation of the data and its delivery to the destination is further amplified in the case of a query-reply pattern. In this case, the interaction may require up to two entire scheduling periods before completion. Moreover, the resources are reserved even if no exchange of application information is ongoing. As a result, the programmer is encouraged to push the information that may be of interest in the neighborhood, as the resources are already allocated and the latency would be minimized.

From Sequential to Parallel. Following from the previous discussion, as the round trip time of a request-reply is short in CSMA approaches, the programmer has clear advantages in waiting for an answer to a query before submitting a new operation. This reduces the amount of state the application must keep. In TDMA techniques, instead, the latency in exchanging information and the allocation of time slots, in which multiple messages can be transmitted at once, favors the parallelization of requests, at the cost of a larger amount of information to maintain at the application level.

Reliability Mechanisms. LPL, by construction, does not avoid the possibility of collisions at the destination, as possibly caused by the hidden terminal problem. Furthermore, LPL requires a trail of messages to be sent by the transmitter for the entire sleep interval, keeping the channel busy for an extended amount of time, increasing the probability of a hidden terminal to concurrently transmit, ultimately causing collisions. As reliability is one of the most important requirements of a monitoring infrastructure based on WSNs, the programmer devotes a lot of effort to implement dedicated mechanisms. The simple and most common solution is to use acknowledgments sent by the destination at the reception of the information. Being the acknowledgment an additional message sent on the channel, it can further contribute to cause collisions.

Being a TDMA, instead, Reins-MAC solves the problems of collisions by construction. It confines the probability of successful delivery to the physical characteristics of the link along which the packet is sent. Moreover, as the receiver of a message can not reply until its scheduled time for transmission, it becomes more efficient to use negative acknowledgments. In fact, in addition to the possible reception of multiple messages from the same sender at once, the losses are likely going to be fewer than in the case of LPL, as collision is avoided. Finally, the information on the successful receptions sent by the destination will not affect the reliability of other ongoing transmissions as they are forced to be scattered in time.

Broadcast vs Unicast Communication. The technique used by LPL to coordinate communication requires the sender to transmit for the whole sleep interval of the receivers, in the case of broadcast communication. This solution keeps the channel busy for long times, spending a lot of energy and favoring collisions. A TDMA approach, instead, sends one single packet to deliver a message to the whole neighborhood, exploiting the broadcast nature of the communication medium. Moreover, in a TDMA, a broadcast or unicast communication has the same probability to fail, which is independent from the ongoing activities in the network.

Communication Requirements. As access to the medium is random and it does not avoid collisions, LPL is unable to support any guarantee, which forces the programmer to explicitly define solutions at the higher layers. Common TDMA techniques already allocate communication resources to each single node in the network. Therefore, the network protocols can rely on the deterministic availability of reserved time slots in which collision-free interactions can be performed. Instead, the unique flexibility of the scheduling implemented in Reins-MAC allows the network protocols not only to passively exploit the schedule as allocated by the MAC itself, but also to actively impose changes. In this way, the current application needs can explicitly guide the distribution of communication resources, which are then allocated directly by Reins-MAC.

5.3 Impact on Network Protocols

The successful experience in the two deployments described in Chapter 2 and 3 provides our background in building system services. As mentioned in the motivations to the definition of Reins-MAC, we expect the usage of the protocol to have a considerable impact in such scenarios. In this section, we focus our attention on reconsidering the design of the network protocols, as a natural continuation of the discussion presented in the previous section. The analysis is divided based on the fundamental building blocks our group developed in the Torre Aquila and road tunnel deployments.

5.3.1 Network Flooding

The first functionality we consider is the dissemination of information from one single point to each node in the network. In our deployments, this basic technique is used to refresh the routing paths. As a message is received, the parent is reevaluated and, if any change is applied, the information is further broadcasted in the network.

LPL perspective. The flooding can be based on a simple broadcast. Through the usage of sequence numbers set at the initiator, messages already forwarded can be recognized and discarded. However, as discussed in Section 5.2, the broadcast operation is likely to cause collisions. Moreover, if we consider that multiple nodes can receive a broadcast at about the same time, the synchronization in the re-broadcast would further increase the probability of collisions. To limit the effect of such synchronization, small random backoffs are used.

Through the Reins-MAC **Lens.** The same mechanisms of the case of LPL, based on broadcast and sequence numbers, is enough to implement the flooding. Interestingly, the developer does not need to implement any backoff, as communication is desynchronized by construction.

5.3.2 Link Quality Estimators

In order to collect data at one or multiple sources, each node must select a single node to which deliver the information. This is usually accomplished by choosing among the neighbors the node with the highest probability to deliver the messages to the final intended destination. The selection is based on link quality estimators. In our deployments, we used the LQI information provided at the reception of a message, however, other techniques are also possible, e.g., ETX [17].

LPL perspective. A quality estimator, unless dependent solely on application traffic, requires additional communication. The messages needed to compute the metric are scheduled as any other and, therefore, they risk colliding. The collision can affect the selection of the parent, making it dependent on the concurrency between the metric evaluation and the application traffic. For example, we can consider that ETX uses messages to identify the expected number of retransmissions required to communicate with a neighbor. If the beacons used to compute the metric collide with the application traffic, the determined metric is artificially affected.

Through the Reins-MAC **Lens.** As the communication is scheduled, the evaluation of a link is independent from the actual ongoing message exchange. Moreover, information on the neighborhood is continuously collected to update the schedule. Both the information already available inside Reins-MAC as well as the continuous pulsing mechanism can be exploited in the computation of the metric and in the spreading of information.

5.3.3 Hop-by-Hop Reliable Data Delivery

Once the collection tree is built and a parent has been selected, data can be funneled towards the collection point. Given the importance of the data for the end user, the reliable delivery is a key requirement. Differently from end-to-end reliability, in our deployments we exploited hop-by-hop reliability to localize the effort of recovering from losses.

LPL perspective. Each message transmission is required to be acknowledged. Both messages and acks can fail, in which case the message gets retransmitted. Given the possibility of false acknowledgments, in both our deployments we used a mechanism to allow the next hop to recover a missed packet. A cache is kept at the sender and sequence numbers are used to find holes in the sequence of messages at the receiver side. If a hole is found, a pull approach is used so that the missed information is recovered by the receiver, asking the sender to search in its cache for the specified missing messages. This scheme requires new data to be received by the destination in order for recovery to happen.

Through the Reins-MAC **Lens.** All the messages are sent inside the sending slot and cached locally. Per-message acknowledgments are not possible at transmission time, which would break the collision-free schedule. Therefore, the receiver in his slot notifies the senders about the last message seen by each of them and the holes in the sequence of received messages, effectively using negative acknowledges. The senders then retransmit the messages during their own slots. Transmission failures are suddenly identified and fixed. However, as both transmission and retransmission require in the worst case an entire frame before being executed, the messages waiting to be delivered to the next hop must be stored in buffers. Given the need to buffer possibly multiple messages, data aggregation sounds as a natural approach to investigate.

5.3.4 Flow Control

In the Torre Aquila deployment, the heterogeneity of requirements asked for the definition of specific solutions able to control the amount of data injected by the application into the network. In particular, accelerometers produce a big amount of data in very short time, which are then made available to the services in charge of delivering those data to the base station, possibly via a multi-hop forwarding.

LPL perspective. We assume that a data transmission failure, a missed acknowledge in the scheme previously defined, is correlated to a congestion on the channel. After this event, the sub-tree is notified about the congestion state with a notification sent in broadcast, forwarded only by the nodes receiving such notification from their own parent in the tree. Alternatively a parent in congestion state can wait for the child to transmit data, effectively forwarding a notification only if needed. On the sources, a simple timer with a constant backoff can be used for low-rate traffic in order to delay transmission to the time when congestion is expected to be solved.

For high-rate, bursty traffic, we used a mechanism that controls the application rate with a variable timeout. Such timeout is controlled depending on the notifications of congestion. If no congestion is heard, the timeout decreases, increasing the allowed application rate, until a defined threshold is reached. When a congestion is notified, the timeout is restarted from an initial value, suddenly increasing the time between sending of consecutive messages and reducing the application rate. Different LPL values can also be used to increase the speed at which data are sent along specific paths. Finally, this solution uses several parameters, whose setting is complex and requires evaluation directly in the field.

Through the Reins-MAC **Lens.** The source limits the application rate to the reserved slot duration (not to the actual slot size). The application rate is then controlled by changing the reserved slot size. At any point in time in the lifetime of the application, a source changes the duration of the slot to the one supporting the application rate or to the maximum it can reserve. The parent modifies its slot to last at least as much as the sum of the slot duration of each child, considering also the local application needs. If the bigger slot that can be allocated is smaller than the demand, a message is sent to the children asking each one to reduce the flow by an amount proportional to the data flow.

5.3.5 Latency Control

The deployment in road tunnels, being part of a control loop, imposed timeliness requirements. The maximum latency of information being forwarded along a routing tree from the sources to the sink needs to be controlled in order to provide the control algorithm with fresh data each time it is executed.

LPL perspective. As discussed in Chapter 3, controlling latency can be supported only through a careful parameter setting. Despite LPL tries to send a message as soon as possible and no delay is required by construction before the transmission can start, the jitter between consecutive data from the same source can greatly vary. Moreover, congestion may easily arise if sampling is synchronized on the sources, forcing delays in the delivery.

Through the Reins-MAC **Lens.** Reins-MAC offers the possibility to reorder the slot assignment. A simple policy, similar to what described in [46], can require each node to move its slot before the parent, adjusting the slot size to fit the traffic flowing from the children. The resulting latency is, therefore, driven by the length of the frame. Interestingly, an alternative solution, to control the resulting latency, can ask the base station to increase its own slot to the point of minimizing the time elapsing in between sources and forwarder slots. Finally, the possible synchronicity of the sampling does not contribute to affect the network performance.

5.3.6 Reliable Dissemination

When a configuration needs to be spread in the network, we need to guarantee eventual delivery of the provided information. In the previously discussed refresh of the routing tree, a node may skip a rebuilding round as it would happen in the case in which the concurrent flooding of neighbors collide at that node. Instead, when reliable dissemination is employed, we require each device to either receive the information at the time of the flooding or to have the possibility to obtain it from a neighbor later.

LPL perspective. Each information to disseminate is identified by a sequence number. Each node broadcast, as in the case of simple flooding, a message with a newer sequence number and discards the ones received with an older identifier. The identifier of the last information received can be used to determine the dissemination state in which a node is. If a neighbor has an higher state, it means that it received some newer information not yet available on the local node, and vice versa in the case in which the node has a lower state.

By piggybacking the state identifier on the outgoing messages, each node can spread in the neighborhood the information about its local state. This mechanism of course reduces the amount of payload available for application data and its efficiency is clearly dependent on the remote interactions issued by either the application or the other services. An alternative is a periodic beaconing, which, in any case, can not be guaranteed to collide neither with other beacons nor with application traffic.

Once a mismatch between the local and some remote state is realized, two policies to update the node with missing information are possible: ask a node to push the information to the node with a lower state, or make the node lacking behind pull data from a node with the higher state. In our deployments, we made use of the latter. In particular in the case of road tunnels, pushing information can result in a lot of collisions due to the high density.

Through the Reins-MAC **Lens.** Reins-MAC already notifies local information by pulsing, corresponding to one or more messages sent in broadcast at the beginning of the slot. The local state id can easily be piggybacked on such pulse. The data can still be initially broadcasted as in the flooding. Moreover, Reins-MAC keeps an updated consistent view of the neighborhood, therefore the network flooding can avoid the broadcast of data already received by all the neighbors.

To face a difference in the state of a neighbor, a push approach is the most natural and easy to implement, given that collisions are avoided by construction. This solution still runs the risk that multiple pushes can happen before the node with a lower state notifies its updated state in its slot. However, at the cost of sending more information, this solution reduces the implementation burden of handling a query-reply interaction and the latency that this would cause in case the operation needs to be submitted multiple times.

5.3.7 Time Synchronization

A common service required in deployments is the synchronization of the clocks on the nodes. Given its additional cost, its employment is connected to strong requirements. The analysis of data coming from the vibration sensors deployed in Torre Aquila required synchronized sampling. In the adaptive lighting application, instead, the absence of requirements on the relative sampling time at the sources suggested to omit the usage of this service. However, we can foresee possible useful usages, e.g., the simultaneous application of system configurations.

LPL perspective. Given the possible latency introduced by LPL in the exchange of time references, the sleep interval is usually reduced to zero when the synchronization procedure is run. After disabling LPL, a protocol like FTSP[51] is executed, which synchronizes the clocks to a reference. The synchronization needs to be run frequently enough to overcome clock drift; however, the specific employed frequency is usually based only on the experience of the network designer. Furthermore, while the synchronization procedure is running, the ongoing application can be affected by the additional communication introduced by the synchronization protocol.

Through the REINS-MAC **Lens.** REINS-MAC already provides pulse synchronization. This is not enough per se for time synchronization. As defined in [83], the mechanism we ported into REINS-MAC offers *synchronicity*: the agreement among the nodes in the network on a common period and a reference phase. This is due to the absence of a need for an absolute reference, which aligns both period and phase to a global clock. However, once a reference node is defined, it can easily provide the time anchor for each synchronization pulse, which can then be forwarded in the network as information integrated into nodes pulses.

5.4 Impact on Communication Abstractions

In both our deployments, we implemented the system services on top of the communication abstraction provided by TeenyLIME, which replaces the OS-level communication interface with operations on a data space shared among 1-hop neighbors. This abstraction demonstrated itself very useful in reducing the programming effort and the resulting binary size. Moreover, it fostered the decouple of the application and system service components, enabling the reuse of functionalities implemented for Torre Aquila first, in the road tunnel scenario next. After describing TeenyLIME, we discuss how the usage of REINS-MAC impacts also the abstraction and the use of it made by the higher layers.

5.4.1 Introduction to TeenyLIME

TeenyLIME defines a communication abstraction to share data among nodes in direct communication. Each node is provided with a local tuple space populated with data in forms of tuples, i.e., an ordered sequence of typed fields. The tuple space is made available both locally and remotely to the neighbors, through a set of operations at disposal of software components built atop TeenyLIME to modify and query the state of the tuple space, as well as get notifications of changes to such a state. The operations make use of patterns to filter the tuples currently stored in the data space; the pattern defines constraints on each tuple field based on its actual type and value.

The interface of TeenyLIME offers operations to insert (**out**) a tuple in the tuple space, and read (**rd**) or remove (**in**) a single data matching the provided pattern already available at the time of the query. As multiple tuples can match against a single pattern, **rdg** and **ing** can be used to retrieve the whole set of results present. Finally, the construct of the *reaction* is provided to signal a notification at the insertion of information of interest. To differentiate between local and remote interactions, each operation has a scope that can be LOCAL, NEIGHBORHOOD, to identify all the nodes in communication range, or the identifier of a specific node.

TeenyLIME also offers a dedicated mechanism to provide information about a node to its neighbors: the *neighbor tuple*. This special tuple is a description of the node, defined and provided by the application, which is piggybacked on each message TeenyLIME exchanges with neighbors. Consequently, at the time of message reception, TeenyLIME extracts the neighbor tuple and makes it available in the tuple space as any other data. After a pre-defined timeout, the information expires and then it is removed from the tuple space.

Finally, the programmer can ask to execute remote operations reliably. In this case, instead of simply delivering the request in a best effort manner, the middleware requires a notification from the receiver and, in the case it does not receive it, it retries the operation for a maximum given number of times. At the end of the operation, the software component that submitted the operation is informed about the resulting state of the operation, i.e., if the request and the corresponding answer were successfully delivered or not.

5.4.2 Neighborhood View

TeenyLIME autonomously makes available tuples describing the neighbors in the tuple space and takes care of both updating and removing them after a period.

LPL Perspective. Without any explicit interaction, no exchange of messages happens among the nodes. For this reason, the only implementation choice to spread the neighbor tuple in a cost effective manner is to piggyback on already planned transmissions. Therefore, in addition to the fact that nodes not communicating remain invisible to the neighbors, no consistency on the view of the neighborhood is provided and the part of the exchanged packets available to data is reduced by the presence of the neighbor tuple.

The programmer is, therefore, in charge of defining a specific component that explicitly submits remote operations to update the neighbor tuples in the neighborhood. Alternatively, the components must rely on each other communication patterns, according to which the neighbor tuple will be updated. The result is a clear dependency among application components and the timings with which they interact remotely, ultimately hampering their decoupling.

Through the REINS-MAC **Lens.** In order to build the schedule and update it, REINS-MAC makes each node pulse periodically, already exchanging information with the neighbors. By exploiting this background communication, it is possible to provide each node with a consistent and updated view of the whole neighborhood. In this way, TeenyLIME can make full use of message to fill it with one bigger tuple or multiple tuples. As a consequence, not only the programmer can make full use of the size of the communicated packets, but it can also rely on a uniform updating mechanism of the informations related to neighbors, which is independent from the application components.

5.4.3 Remote Queries

Whereas interactions among local components are clearly independent from the usage of the wireless medium, querying a remote tuple space requires both the requester and the replier to access the shared medium. In the case of random CSMA access, the interaction completes in very short time. With a periodic scheduling of communication, as provided by TDMA approaches, the operations require longer times and their results can explicitly depend on the scheduling.

LPL Perspective. By accessing the medium as soon as a message is ready to be sent, communication in LPL is localized with computation, as near as possible. Therefore, as soon as the application submits an operation, TeenyLime tries to forward the request to the destination and the same is done right after the reply is computed at the destination side. In this way, the round trip required to conduct the operation ends potentially shortly after its submission, in the worst case after twice the sleep interval.

Moreover, the requester does not have any advantage in submitting more requests at the same time, as the communication required by subsequent operations would likely collide with the reply of the previous ones. Therefore, in the development of the application, the programmer prefers to wait the completion of previous operations before continuing the processing. In addition, the effectively immediate reply to the operation further limits the probability that other concurrent remote operations involve the same requester and replier.

Through the Reins-MAC lens. Given the scheduled access implemented by Reins-MAC, once an operation is submitted by the application, the query must wait the beginning of the local transmission slot, similarly the waiting is forced at the replier. In the worst case, the operation must wait for an entire frame before a chance of completion. If any remote operation depends on the result of a previous one, it must wait possibly long before having a chance to continue in the processing, further delaying the end of the computation.

As each node is provided with a continuous slot, multiple messages can be sent all at once. However, operations dependent on each other would require to wait for some result in order to continue in the processing. The result is a limitation in the number of messages sent in each slot, ultimately wasting the allocated communication resources. Alternatively, the application should handle the parallelization of the requests, their state, and their interdependency, until the result is returned.

More importantly, the execution of remote operations would be forced to be interleaved according to the communication scheduling, resulting in a clear dependency of the operation results from the order in the schedule. For example, we can consider a network composed by three nodes, A, B, and C, which follow each other in the alphabetical order in the communication schedule. Assuming that A and B wants to query C's tuple space, the query of A would be forced to be queued on C, followed by the one coming from B. In the case in which A and B are contending to obtain and remove some data from C, A would clearly always success to the detriment of B. This would not happen with LPL, as the access to the channel is random and therefore all the neighbors have equal chance to obtain access to the same data.

Finally, the aforementioned impact implies changes at the internal implementation of TeenyLime. In particular, to handle the interleaving and the delays between request and reply two different alternatives are possible: either the result is computed at the reception of the request or right before producing the reply. Despite this does not affect the previously discussed example, we can consider a local process running on C that submits a local operation to remove the same data in between others' requests and corresponding replies. Depending on the selected implementation choice, different results can be obtained.

5.5 Concluding Remarks

In this chapter, we reconsidered the system services used in the Torre Aquila and road tunnel deployments. We demonstrated the impact of the underlying MAC on their design. This is partially a consequence of the inherent differences between TDMA and CSMA techniques in supporting access to the medium. However, this effect is also caused by the unique flexibility of Reins-MAC and the interface it provides to the higher layers. Furthermore, the repercussions of our MAC protocol extend to the definition and usage of communication abstractions, which are supposed to hide the details of the underlying communication mechanisms. This first step demonstrated the ability of Reins-MAC to provide a concrete alternative to currently employed solutions.

Part IV

Observing Communication

Chapter 6

Motes in the Jungle: Lessons Learned from a Short-term WSN Deployment in the Ecuador Cloud Forest

The design of a system and the definition of the services and algorithms employed in it are bound to the deployment scenario. The characteristics of connectivity are the main factor affecting the performance of an implemented solution in the field. Given that assessing the specifics of a target environment is crucial and complex, a preliminary pilot deployment is required. In this process, the final user is usually involved only to define the requirements not to actively handle the technology. In this last chapter, we report about a short-term deployment, undertaken directly by biologists, which took place in the cloud forest of the North-Western slopes of Ecuadorian Andes during March 29–April 3, 2010.[1]

6.1 Scenario, Motivation and Contribution

The work described here is part of a larger research effort targeting the monitoring of biodiversity in community-based primary cloud forest reserves in this Andean region. Indeed, this area is at the confluence of two of the world's hottest biological hotspots: the Chocó-Darién Western Ecuadorian and the Tropical Andes. Available checklists of vertebrates likely miss most reptile and mammal species, including medium-to-large ones. The knowledge about these species' use

[1]The content of this chapter is a joint work with Matteo Chini, Amy L. Murphy, Gian Pietro Picco, Francesca Cagnacci, and Bryony Tolhurst, published in "Motes in the Jungle: Lessons Learned from a Short-term WSN Deployment in the Ecuador Cloud Forest", 4th Workshop on Real-World Wireless Sensor Networks (RealWSN'10), Colombo (Sri Lanka), December 2010 [7].

of space and community interactions is essential to ascertain their susceptibility to environmental changes and guide conservation measures. Available information is extremely sparse and based on discontinuous observations and occasional surveys. Direct observation of animals is not a robust method, due to the very dense vegetation, while traditional indirect methods, such as capture-mark-recapture or radio-tracking are extremely effort-demanding as these areas are secluded. Recent advancements in wildlife studies, e.g., the use of GPS devices, are expensive and therefore applicable to a small number of species and sample size. WSNs provide a new, exciting option in such challenging environmental conditions, especially for long-term monitoring. Advantages include the need for only a single capture (to fit the node) and the possibility to study a large sample thanks to the relatively low equipment and deployment cost. However, an essential step in seizing this opportunity is the evaluation of the node performance in the target environment.

The envisioned WSN application will encompass nodes permanently deployed in the environment at known locations as well as attached with collars to the animals themselves. We intend to use motes functionally equivalent to Moteiv's TMote Sky [66], arguably the most popular platform today. However, the 2.4 GHz band used by the CC2420 radio chip on these motes is known to be highly sensitive to foliage and water—essential ingredients of a cloud forest. Therefore, the primary motivation behind the study described here was to assess the connectivity characteristics of the target environment to determine the feasibility of our WSN architecture and guide its design.

Related work. A few real-world deployments focus on forests [80], but with characteristics different from ours. Despite the importance of understanding the connectivity of the environment targeted by a WSN, this information is rarely reported in the literature. Instead, the problem is usually tackled with studies targeting either static [74] or mobile [53] scenarios. All the reported works, however, leverage the possibility to progressively refine the investigation based on the findings. Our need to define a priori the entire experimentation pushed us towards a more general methodology, something still not available in the literature. To design our study we leveraged our prior expertise in comparing the network characteristics of a tunnel against the vineyard environment [57]. However, the differences in the application scenario, involving mobile nodes, and the inability to access the experiment site demanded a significant revision of our techniques.

Challenges. The deployment itself presented non-trivial logistical difficulties due to the geographical distance and the harshness of our target environment. Things were further complicated by the fact that the WSN experiments were "piggybacked" on the biologist's trip to Ecuador for other research purposes.

As a consequence, we faced rather unusual requirements. In the literature, similar experiments are typically run by the WSN developers, often in rather controlled environments. Instead, in our case the experiments had to be run by the biologists, and *in isolation.* Remote WSN configuration was not an option, due to the absence of data connectivity from the experiment location—the jungle. Similarly, a multi-phase deployment, where the output of one experiment guides the setup of the next, was also not an option due to the distance between the experiment location and the closest Internet access, and to the duration of the experiments. The latter was limited by the biologist's already-established trip schedule, further reduced by the inevitable lost baggage.

Simply put, this meant that our hw/sw WSN setup had to work out of the box for the entire duration of the experimental campaign, and had to be simple enough to be operated by someone without expertise with this technology.

Contributions and findings. The details about our cloud forest experiments are provided in Section 6.3. The main contributions of the work described in this chapter are the following:

I. *Low-power wireless in the jungle environment.* In Section 6.4 we analyze the gathered data. The depth of the analysis is somewhat limited by the aforementioned logistic problems, as we did not have a second chance to investigate the source of unexpected behaviors. However, we are not aware of other studies investigating low-power wireless communication in an environment similar to ours and therefore, even with these limitations, we believe our study can be of value for the research community. Moreover, some of our findings are somewhat surprising. For instance, we expected links to be rather short and unreliable, due to foliage, water, and humidity. Instead, our data show that 30-meter links are common, and in some cases reliable communication occurs up to 40 m.

II. *Mobile nodes as a connectivity exploration tool.* The inclusion of experiments with mobile nodes was initially motivated by the animal-borne nodes in our envisioned application. We expected to draw the bulk of our considerations from stationary-only experiments. Instead, mobile nodes played a much more relevant role in our study. On one hand, the stationary-only experiments did not deliver the amount of data we expected. The connectivity patterns were not known in advance, and a multi-phase deployment was not an option, as already discussed. Mobile experiments provided a data set complementing the stationary ones. On the other hand, with hindsight, the use of mobile nodes is an effective way to explore connectivity, regardless of mobility requirements. Intuitively, a broadcasting node moving through a single, well-designed path yields a wealth of information, more varied and fine-grained w.r.t. stationary-only experiments, even considering the interference introduced by the person executing the experiments. This enables a more precise "connectivity map" of the environment, that can be used for instance to guide node placement. We believe the use of mobile nodes can become an essential element of studies aimed at characterizing connectivity in WSN environments.

III. *When WSN developers are not in charge.* Our experiments were run by someone other than the WSN developers because of opportunity. There may be other reasons, e.g., the necessity to require authorizations or safety concerns related to the target deployment area. In any case, for WSN to become truly pervasive, end-users must be empowered with the ability to deploy their own system. The lessons we learned, distilled in Section 6.5, can be regarded as a contribution towards this goal.

6.2 Deployment Scenario

Location. The community-based reserve of Junin, in the Intag region of the Imbabura province in Ecuador (0°16'19.09"N; 78°39'28.92"W) is between 1,200 and 2,800 m above sea level of the North-Western slopes of the Ecuadorian Andes. Significant portions of these mountain areas are primary cloud tropical forests, almost permanently cloudy and foggy. According to the United Nation's World Conservation Center, cloud forests comprise only 2.5% of the world's tropical forests, and approximately 25% are found in the Andean region. Therefore, they are considered at the top of the list of threatened ecosystems. The climate is tropical, and the flora and fauna incredibly rich, with about 400 species of birds and 50 known mammal species (including 20 carnivores), many probably still unchecked or even unknown. The small human community of about 50 people is 20 km from the closest village, and a 7 hour dirt-road drive from the closest town. The vegetation is made by relatively scattered mature trees, constituting the canopy, and a dense undergrowth of shrubs and epiphites. During the rainy season (November-May), when we ran our experiments, it rains every day for nearly the entire day.

WSN Equipment. Our experiments used 18 TMote Sky nodes, equipped with the ChipCon 2420 IEEE 802.15.4-compliant, 2.4 GHz radio and on-board inverted-F micro-strip omnidirectional antenna. The choice of this popular platform is motivated both by our intended use of a similar platform in our own wildlife application, and to enable comparison with similar experiments in different environments reported in the literature. Alternate hardware would significantly modify the results, e.g., an external antenna would likely dramatically increase the observed connectivity. Moreover, these motes are provided with an external flash memory, enabling storage of the experiment data.

As stationary motes were intended to be attached to trees in a very humid environment, under heavy rain, we used IP65 water-proof boxes with a transparent cover, enabling the sampling of the light as requested by the biologists. Inside each box we glued a USB female connector to easily anchor and replace the node as needed. Each box also contained a battery holder with two size D batteries and desiccant bags to protect the node against humidity. The packaging is shown in Figure 6.1 in the same orientation as it was attached to the trees. In contrast, the mobile node was simply a TMote Sky powered by 2 AA batteries, wrapped in a plastic bag.

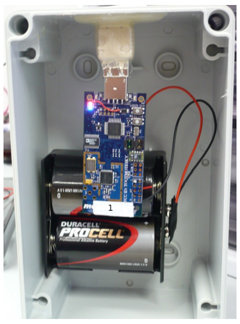


Figure 6.1: Packaging.

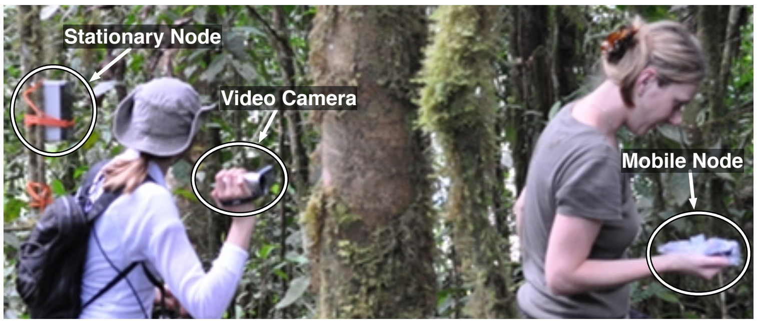


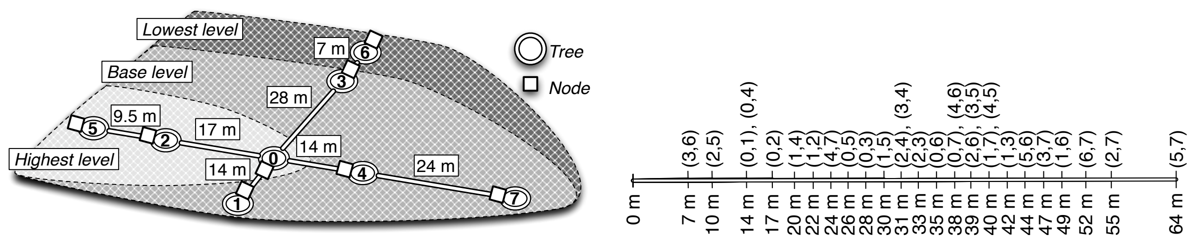
Figure 6.2: In the jungle with mobile nodes.

6.3 Experiment Design

The WSN was composed of 8 nodes, placed in a cross configuration, as shown in Figure 6.3(a). The placement was determined as part of the stationary experiments, described next. Node 0 served as the experiment coordinator, broadcasting a message indicating the start time and configuration of each experiment. All communication took place on channel 18. Since no computer was available in-field, we used the motes' LEDs to visualize the node functionality. For example, toggling the yellow LED indicated message transmission, while toggling together the other two LEDs indicated message reception. At node boot time, a visual code for the battery voltage was shown to advise for battery replacement in case of values below 2.7 V, the minimum required to write to the flash memory. To start an experiment, the biologist pressed the user button.

The software was built on TinyOS and without any MAC protocol, given our goal of characterizing physical connectivity. Packet collision was avoided by an appropriate transmission schedule sent at the beginning of each experiment by node 0. For each experiment, and for each link $i \to j$, we recorded in the flash the following metrics:

- *Packet Delivery Ratio* ($PDR_{i \to j}$), the number of packets received at node j over the total number of packets sent by node i;
- *Received Signal Strength Indicator* ($RSSI_{i \to j}$), the signal strength of the packets transmitted by i, as observed by the radio of j;



(a) Node placement. (b) Link Distances.

Figure 6.3: Deployment of stationary nodes; each color corresponds to about 1 m difference.

- *Link Quality Indicator* ($LQI_{i \to j}$), the correlation index between the symbol received at j, sent by i, and the one to which it is mapped after radio soft decoding.

6.3.1 Preliminary Tests

Goals. Given the lack of reported experiences in scenarios similar to ours, the primary goal of these tests was to determine the communication range, to properly place nodes in the next experiments. These experiments also investigated different power transmission levels as well as the impact of direct tree obstruction.

Implementation. The experiments exploited only node 0 and 3 in Figure 6.3(a). We implemented two experiments, one to determine the range of communication, and the other to investigate the effect of signal power and tree obstruction. In the former, each node sent 600 messages with an inter message interval (IMI) of 2 s. All messages were sent with -1 dBm transmission power. The LED visual feedback was used to guide the identification of the maximal communication range. In the latter experiment, each node sent a sequence of 3000 messages with a 2 s IMI, interleaving sending between the involved nodes. These messages are logically divided into 5 tests of 600 messages each, 3 at -1 dBm, commonly used in WSN deployments, and 2 at -8 dBm, to investigate the effect of reduced power. For each 600-message set we stored the aggregated average RSSI ($\overline{RSSI}$), average LQI ($\overline{LQI}$) and PDR values over all received messages.

Deployment. In all experiments node 0 was attached to a tree at 1 m height, while node 3 was placed on a chair. In the first experiment, the two nodes were in line of sight (LoS) and the biologist gradually moved the chair away from the tree while monitoring the LEDs for determining a safe communication range, which she established at 28 m. The second experiment with different power levels was run a first time with nodes in line of sight, and then again with node 0 directly behind the tree, creating a link obstruction.

6.3.2 Tests with Stationary Nodes

Goals. The purpose of these tests was to investigate connectivity among nodes at different distances, over a long time interval, and at different node heights.

Implementation. These experiments used the nodes as in Figure 6.3(a) and, as in the preliminary tests, relied on node 0 for disseminating the start time and transmission schedule. In each experiment, each node sent 215 messages with an IMI of 8 s, resulting in an interval of 1 s between nodes adjacent in the transmission schedule. The experiments were batched and ran for an entire day, interleaving 23 experiments at -1 dBm with 22 experiments at -8 dBm. Before this batch, a 1-hour *setup* experiment (with LEDs enabled) was performed, to verify connectivity and thus node placement. At the end, each node computed and stored the overall PDR, $\overline{RSSI}$, and $\overline{LQI}$ w.r.t. all other nodes.

Deployment. Node 0 and 3 were left in place after the preliminary tests. During the *setup* experiment, all the others were moved one by one away from node 0 in small steps. Based on high-level instructions, the LEDs blinking, and the communication range of 28 m determined in the preliminary tests, the biologists determined the final placement shown in Figure 6.3(a), yielding the set of distances covered as shown in Figure 6.3(b). The experiments were executed twice for a total of 2 days.

Our original idea was to deploy the nodes in a flat area, placing them first at ground level, then at 1 m from the ground, and finally at various, possibly higher heights. The rationale was to determine node placement in the least favorable connectivity conditions, close to the ground. Unfortunately, due to the delayed arrival on site (caused by lost luggage), the biologists decided to eliminate the first experiment. Moreover, due to the available terrain, highly irregular and on a sort of hill as shown in Figure 6.3(a), the second and third deployments were reversed. Therefore, the deployment was setup in the connectivity conditions *most* favorable, which affected the subsequent experiments. Indeed, undergrowth interfered significantly during the second test, making its results unusable. Also, node 2 failed to start some tests and its data has been excluded.

6.3.3 Tests with Stationary and Mobile Nodes

Goals. These experiments were initially motivated by our wildlife application, combining fixed and animal-borne nodes. When interpreting the results, however, we realized the importance of these tests in enabling exploration of connectivity at many more distances w.r.t. the static deployment, yielding more spatial continuity to data points.

Implementation. In these experiments, node 0 was carried by the biologist, who moved throughout the deployment area. Stationary nodes only listened, while node 0 broadcast messages at -1 dBm for 15 min, with an IMI of 500 ms, yielding 1,800 messages per experiment. Unlike stationary experiments, which recorded only one aggregate value for each link, in the mobile tests statistics about each individual message were recorded. This allowed us to treat each message separately, by considering the distance between the mobile node and each stationary node at the moment it was sent. Offline data correlation across nodes was enabled by timestamping the message at the sender, and saving this along with the RSSI and LQI values at the receiver. During experiments the biologist moved freely, her path recorded by a video camera carried by a second team member (Figure 6.2), allowing us to visualize the movements and correlate the timings.

Deployment. The placement of stationary nodes was the same as in Section 6.3.2, but the nodes were physically replaced as their (pre-loaded) software was different. The nodes were placed at 1 m from the ground. The mobile node was either held in the biologist hands (as in Figure 6.2) with the antenna parallel to her shoulders and the board facing the sky or carried

Link	**TX power**	*PDR*		$\overline{RSSI}$		$\overline{LQI}$	
		LoS	**Tree**	**LoS**	**Tree**	**LoS**	**Tree**
$0 \to 3$	-1 dBm	86.7%	79.5%	-87 dBm	-91 dBm	99	90
$3 \to 0$	-1 dBm	84.4%	69.7%	-88 dBm	-92 dBm	98	88
$0 \to 3$	-8 dBm	24.2%	1.3%	-92 dBm	-93 dBm	80	77
$3 \to 0$	-8 dBm	11.8%	0.5%	-92 dBm	-94 dBm	77	75

Table 6.1: Results from the preliminary tests.

chest height inside a pouch, unfortunately with undefined orientation. First, the biologist stood near a stationary node (node 2) and made simple movements of approximately 1 m amplitude along the horizontal plane at the node height and along the tree, approaching the node from four directions—front, back, right, and left. Then, the biologist moved back and forth between node 1 and 3, then between 2 and 5. Although these experiments focused on movement between a subset of the available nodes, all nodes in the network recorded message reception, thus we gathered a large amount of data. Finally, the biologist composed a path visiting all stationary nodes. Each path was repeated 4 times. In total, these experiments produced 116,448 data points. We excluded the data collected by node 7 as we verified that its short-range reception was abnormal.

6.4 A Mote's Life In the Jungle

This section presents our experimental results. Due to the previously described limitations, analysis is limited, and further insights into the observed behaviors require additional dedicated tests. Nevertheless, valuable information about communication in the jungle environment is presented.

6.4.1 Preliminary Tests

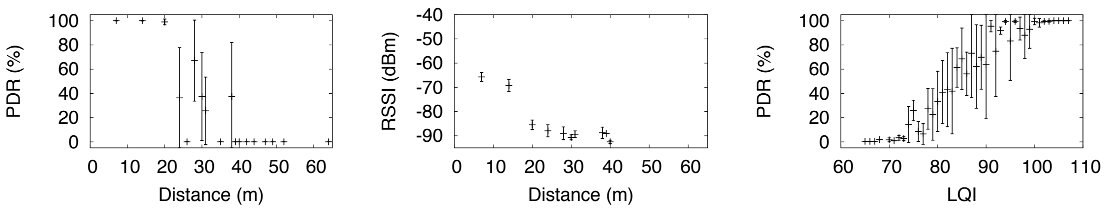
The results of the tests on transmission power and tree influence are shown in Table 6.1. As discussed in Section 6.3.1, these involved only node 0 and 3. At -1 dBm, both *PDR* and $\overline{LQI}$ are high. This is expected as these results are at the distance of 28 m the biologists chose as the border of good connectivity. Interestingly, our initial guess for a safe communication distance was much lower, around 10-15 m, given the presence of thick vegetation and high humidity. $\overline{RSSI}$ is low but, given the absence of radio interference in the forest, it does not significantly affect *PDR*. The presence of a tree right in front of a node may cause link asymmetries. With nodes in line of sight, the *PDR* difference between the two link directions is only 2%, but with the tree in between this increases to 10%, indicating a weaker link when communication originates near the tree. $\overline{RSSI}$ and $\overline{LQI}$ do not show marked asymmetries, although they decrease when the tree obstructs the link. With lower transmission power, *PDR* is non-negligible but more heavily influenced by the tree. The low $\overline{LQI}$ is consistent with the next experiments showing that 28 m is well outside the good-connectivity range at −8 dBm.

6.4.2 Tests with Stationary Nodes

Long-Distance, High-Quality Links. We expected the dense jungle foliage to significantly limit communication. Instead, Figure 6.4(a) shows that communication is almost perfect up to 20 m, although the high *PDR* at 19.8 m occurs with a relatively low signal strength (Figure 6.4(b)). Further, although the 38 m link falls well beyond the region with perfect communication, analysis over time (Figure 6.5) shows that this link was also perfect for more than half of the experiment duration. While this is clearly an anomaly of the setup, it clearly demonstrates that connectivity in the jungle is much different than expected. At −8 dBm, the area with perfect links is only slightly reduced to 14 m.

Fluctuations and Asymmetries of Mid-Range Links. Figure 6.4(a) and 6.4(b)) show that links with mid-range distances of 20–40 m have highly-variable quality and low RSSI. The *PDR* large standard deviation is best viewed over time in Figure 6.5, where each point describes the result of one 30-min experiment for a given link. From the detail on the right-hand side of the figure, one can see that the variability is unpredictable. For example, around hour 15 some links improve while others decline. Further, some links such as $(3, 0)$ show transient asymmetries. Weather could be the culprit, and indeed it rained during the majority of these tests. Although one would expect a global decay of link quality, it is possible that humidity, rain, and pools of collected water affect communication in local, unpredictable ways, although we do not have direct observations confirming this. In any case, mid-range links clearly cannot guarantee connectivity, but they can certainly be exploited transiently by adaptive routing algorithms.

Long-Range Interference with Reduced Power. At −8 dBm, links outside the perfect communication range disappear for long periods of time (Figure 6.6). While these links are basically unusable, they can cause long-range interference. For example, Figure 6.6(b) shows



(a) *PDR* vs Distance. (b) $\overline{RSSI}$ vs Distance. (c) *PDR* vs $\overline{LQI}$.

Figure 6.4: Average and standard deviation of the results from stationary tests with power −1 dBm.

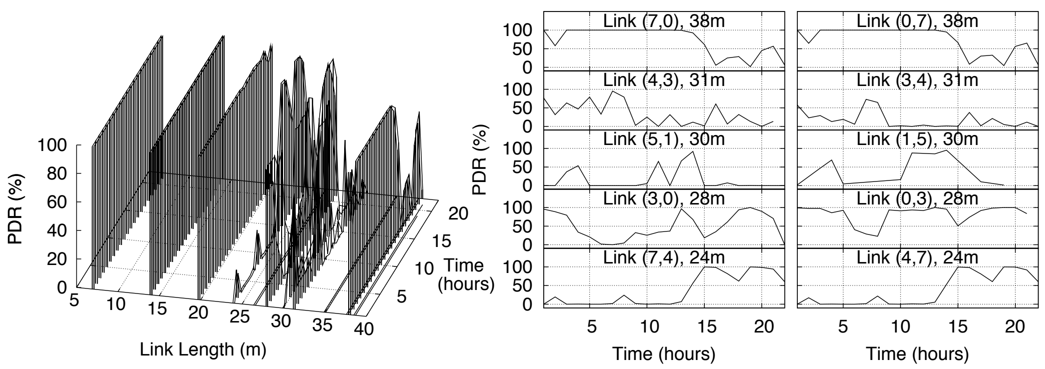
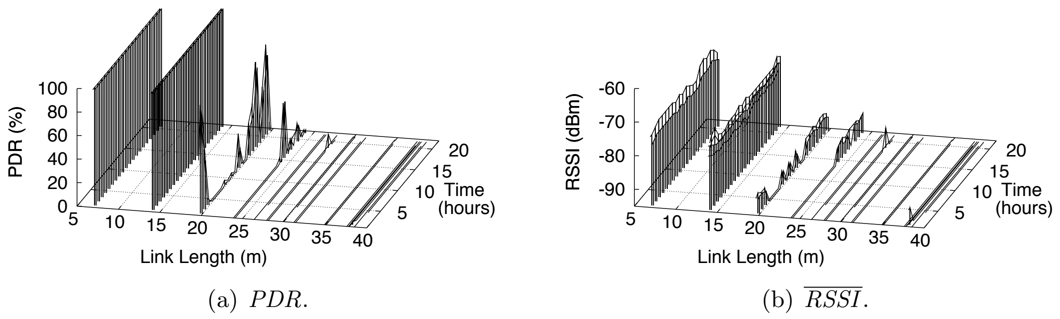


Figure 6.5: *PDR* over time with power −1 dBm from stationary tests.



(a) *PDR*.

(b) $\overline{RSSI}$.

Figure 6.6: Results over time with power −8 dBm from stationary tests.

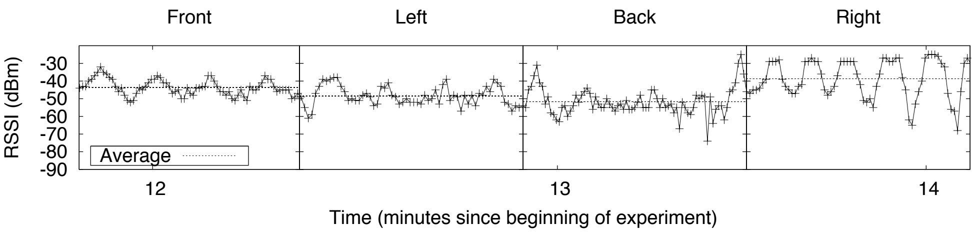


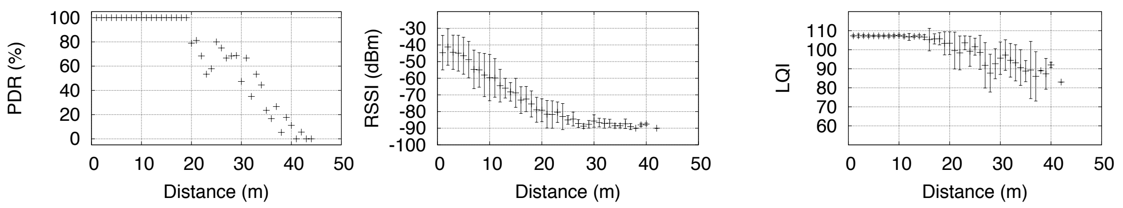
Figure 6.7: Node 0 approaching node 2, attached to a tree, from different directions.

messages received with very low RSSI even at 40 m. Although these distant transmissions rarely succeed, they could easily disrupt overlapping shorter-range ones.

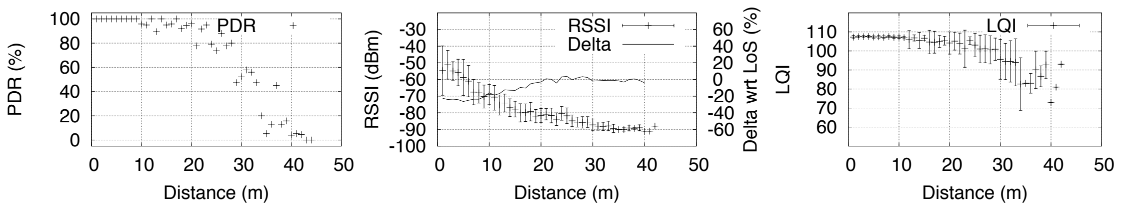
6.4.3 Tests with Stationary and Mobile Nodes

"Omnidirectional" Antenna. Figure 6.7 shows the effect of a node approaching a second one fixed to a tree, as described in Section 6.3.3. Based on the biologist's 1-meter horizontal movements, the different shapes of the *Front*, *Left*, and *Back* curves clearly show the well-known fact that the used antenna is not perfectly isotropic. Interestingly, the flat tops in *Right* do not correspond to a movement pause, rather to the "saturation" of RSSI for very short links. Tree obstruction is clearly evident in the *Back* curve.

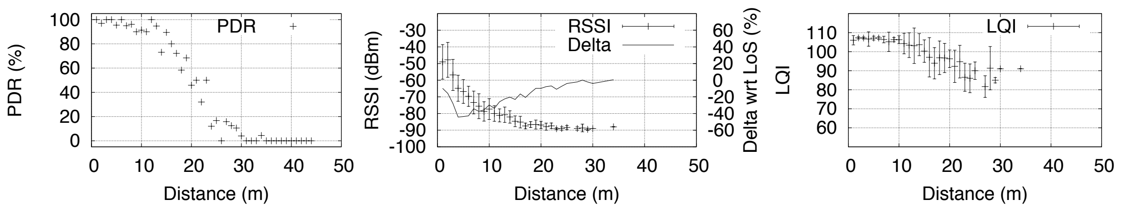
Influence of Body, Tree, and Ground. In Figure 6.8 the biologist, holding the mobile node in front of her chest, looped four times around nodes 1 and 3. We decomposed the data trace to distinguish the possible obstructions. For example, when walking from 1 to 3, the tree obstructed communication received at 3 (Figure 6.8(b)), and the body obstructed receptions



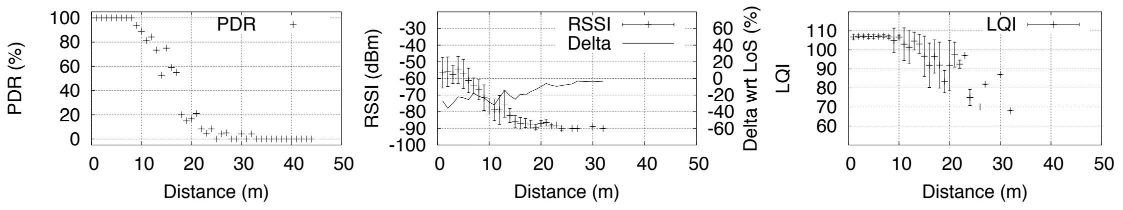
(a) Line-of-sight at 1 m height.



(b) Tree obstruction.



(c) Body obstruction.



(d) Line-of-sight at ground level.

Figure 6.8: Effect of tree, body, and ground on communication. The line in the RSSI plots shows the delta in percent w.r.t. the line-of-sight shown in (a).

at 1 (Figure 6.8(c)). As a reference, we chose the line-of-sight case: reception at 1 when walking from 3 to 1 (Figure 6.8(a)). The same experiment was run with the mobile node held a few centimeters from the ground (Figure 6.8(d)).

Trees induce a reduction up to 20% on $\overline{RSSI}$ in short links (< 20 m), while longer links are not affected. The body also reduces $\overline{RSSI}$ in short links, but more significantly, up to 40%. Moreover, the body reduces the maximum communication range by 10 m, as denoted in Figure 6.8(c) by a nearly-zero *PDR* beyond 30 m. As expected, the simultaneous obstruction of tree and body yields a combination of previous results: a shorter communication range and $\overline{RSSI}$ reductions up to 60%. This bears an important implication for our wildlife application, where we need to estimate the distance between animals upon contact: RSSI-based distance approximation schemes may have a significant error, induced by trees, the body of animals, and the direction the animal approaches the tree, as discussed previously.

Placing the sender near the ground produces a different combination of effects. Specifically, the line-of-sight communication range is much shorter than in Figure 6.8(a), but the $\overline{RSSI}$ is affected by at most 20%. As this scenario is the closest to our target deployment with tagged animals, it warrants additional study.

6.4.4 An Evaluation of Mobile Nodes as Connectivity Probes

We take a step back from the data analysis to consider our data collection methodology, specifically, comparing the results of stationary test against those with mobile ones.

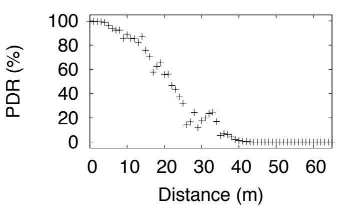
Aggregated Mobile Tests vs. Stationary Tests. Thus far we have looked only at excerpts of the mobile traces, extracting cases with specific characteristics. Here, we aggregate all data points collected over all node movements, with the results shown in Figure 6.9(a)–6.9(c). To plot *PDR*, we calculate the distance between the mobile and each stationary node, then plot the number of messages received over those sent at each distance. $\overline{RSSI}$ and $\overline{LQI}$ are instead shown as the average and standard deviation over all the messages received along links of a specific length. We then compare these data to those collected in the stationary tests of Figure 6.4, by plotting the percentage difference in Figure 6.9(d), only for the points studied in the stationary scenario.

In the mobile scenario, the reduction of $\overline{RSSI}$ on short links (< 15 m) is likely attributable to body interference as observed in Figure 6.8(c). From the *PDR* comparison in Figure 6.9(a), we note that at all distances, the mobile scenario produces *worse* results, meaning that the *PDR* at a given distance is lower in the mobile scenario than in the stationary. To understand the implications, consider that we intend to use the results of this study to plan a future deployment. If we base this deployment only on the results of the mobile study, all stationary nodes in our future deployment would certainly be connected. Instead, if we base our fixed node placement on the stationary results, we would erroneously expect to communicate with mobile nodes carried by animals at the same distance. In other words, the mobile case underestimates the communication potential of stationary nodes while the stationary overestimates communication to mobile nodes.

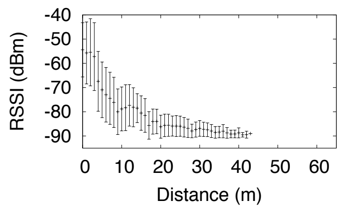
Interestingly, Figure 6.9(c) shows better quality links in the mobile scenario. While this is opposite from the observations of *PDR*, the stationary experiments showed that LQI varied significantly throughout the day. Instead, the mobile experiments were concentrated in less time, and may have taken place in favorable connectivity conditions.

Figure 6.9(e) accounts only for the data recorded in conditions similar to those of the stationary only tests, i.e. removing the body shielding and using the data from Figures 6.8(a) and 6.8(b), namely LoS and tree-only obstruction. For short links (< 20 m), values are in agreement while longer links are hampered by interference from the ground and dense low-level foliage in the mobile scenario. In the stationary tests, nodes were always within LoS, therefore the undergrowth had minimal effect.

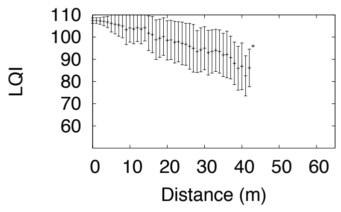
Statistical Relevance of Mobile Tests. The experiments run with a mobile node made it possible to explore the physical space in a continuous fashion, spreading the collected data points over more distances w.r.t. stationary-only tests. To understand the effectiveness of this



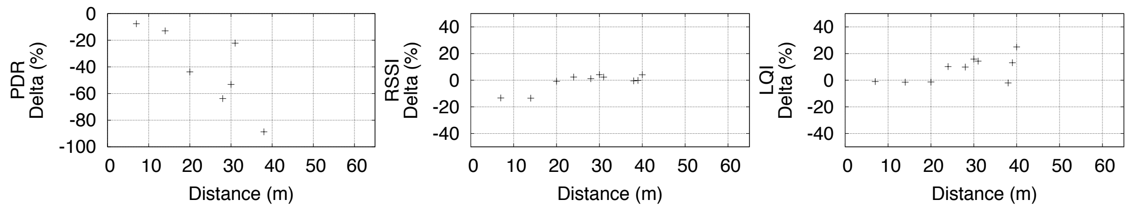
(a) *PDR*.



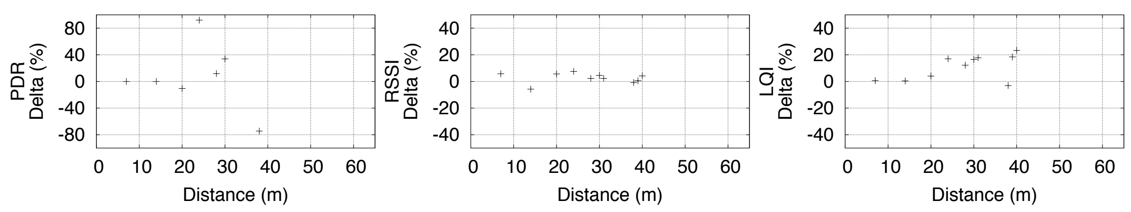
(b) $\overline{RSSI}$.



(c) $\overline{LQI}$.



(d) Comparison of mobile with stationary tests.



(e) Comparison of mobile (no body shielding) with stationary tests.

Figure 6.9: Aggregated results over all 11 mobile experiments. In (d) and (e), the difference in *PDR* for the links longer than 38 m is outside of the chart range.

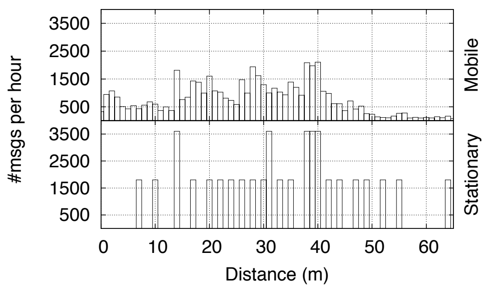


Figure 6.10: Number of messages sent per hour along links of a given distance by stationary and mobile tests.

approach, Figure 6.10 compares the average number of messages received in 1 hour for each distance covered in the mobile case, to the number of messages that the stationary experiment would receive with the same IMI as the mobile nodes, i.e. 500 ms. Recall that to avoid collisions, our stationary experiments used a 1 s IMI. The distribution of the tested distances is naturally biased by the executed movements. Nonetheless, even without a guided motion plan, all distances less than 40 m have been tested by at least 400 messages, i.e., 25% of the messages sent by the stationary tests for each link. The ability of the mobile node to cover so many distances clearly motivates its use as a probe to characterize connectivity.

6.5 Lessons Learned

Our experiments were run in a challenging scenario by biologists without WSN expertise, with limited equipment, and in isolation. Our group have never faced this combination in previous real world deployments, and we learned interesting lessons.

Mobile Nodes: Application Insights or Connectivity Probes? It was the biologists who requested experiments with mobile nodes, to concretely understand what WSNs could offer them. Nevertheless, we learned that the use of mobile nodes, despite the inherent imprecision, is useful for characterizing an unknown environment and guiding the actual deployment. Further work is needed to explore the opportunities of this technique and understand its limitations, e.g., the difficulty to capture long-term variations.

The Role of LEDs. In our study, the node output had to be simple yet informative enough to guide the biologists. Our solution, based on giving a visual clue only about send/receive operations, contributed to the creation of very long links between stationary nodes which in turn contributed to the failure of the second set of stationary experiments, as mentioned in Section 6.4.2. A visual representation of the RSSI values (e.g., represented by a "histogram" using the three LEDs), would have led to shorter links, which would have produced meaningful data even in the second set of experiments.

Testing Blindly. Our experimental campaign involved many decisions taken blindly. We did not have an understanding of the environment based on previous studies. We did not have a well-defined methodology for performing this kind of experiments, and none yet exists in the WSN field. Finally, we could not modify experiments based on intermediate results. We partially reduced the unknowns by breaking down experiments into phases with well-defined outputs. Examples are the preliminary tests (Section 6.3.1) and the 1-hour *setup* phase preceding the stationary tests (Section 6.3.2). These enabled the biologists to take informed decisions autonomously, partially obviating the absence of WSN experts in-field. Nevertheless, this did not avoid incorrect decisions, and could not provide answers for unanticipated questions (e.g., the cause of high time variance of links).

6.6 Concluding Remarks

The work described in this chapter presented an analysis of low power wireless communication in a previously unknown scenario, which manifested unexpected results. After the mission in Ecuador, eager to further investigate similar environments and encouraged by the manifest interest of the biologists, we reproduced the same experiment campaign in the mountains nearby our institution in Trento. Different vegetation and presence of snow were just some of the factors we experienced as key in defining the behavior of communication. This demonstrated that characterizing a scenario, and therefore its impact on the functioning of a WSN, only at deployment time is insufficient. Significant and unanticipated changes can happen during the whole system lifetime. Such influence and variability ask for the definition of comprehensive and effective exploration techniques, comprehensible to non experts. Our experience in Ecuador was a first step in this direction.

Part V

Conclusion

Chapter 7

Conclusion

In this thesis, we described the concrete experience obtained by our group in building real world deployments, fostering the vision of WSNs as a viable technology in diverse scenarios. Despite the decade of progress in the field, building reliable systems with WSNs at their core is a challenging task, still relying on the experts' skills. Some of the limitations reside at the physical layer, for which the human remains the most reliable solution. Handling proper node placement is still a challenge currently solved through informed choices made by experts. Whereas deployment methodology is still an unexplored research subject, mechanisms to control access to the wireless medium are a well investigated topic. Nonetheless, we argue that the literature lacks solutions able to make the vision of WSNs a reality. Works introduced so far either limit their applicability and efficiency or fail to maintain their promises in operational systems. With the objective of reinstating the control of communication in the hands of the resource user, we introduced REINS-MAC. The protocol defines a scheduling mechanism that the common belief considered impractical. Its ability to empower the higher layers in the communication stack with the control of resource allocation, factoring out the coordination needed to support it, promotes both the revision of old solutions and the design of new ones. In addition to the deployments described in this work, REINS-MAC is a promising solution in all the contexts where controlling communication quality is a requirement.

As a concrete example, the reader can consider a scenario where a monitoring infrastructure is used to detect movements of rocks and signal early alarms in case of landslides or avalanches. Networks composed by hundreds of nodes could make use of REINS-MAC to guarantee latency control of warnings, dynamically reconfiguring the resource allocation as the network topology changes with the sliding of the terrain. In order for REINS-MAC to keep the resource allocation consistent, a view of the nodes in communication range is continuously maintained. This information does not only enable the application to reason about communication resource allocation, but also exports knowledge of the current network topology. By exploiting such a view, we can conceive, for example, of automatic deployment of sensors made by unmanned aerial or terrestrial vehicles. Helicopters or quadcopters would place nodes in hazardous scenarios, e.g., car accidents inside road tunnels or catastrophes in civil areas, guaranteeing the basic connectivity properties required to form a continuous and dense monitoring infrastructure. Once deployed, such a network can also provide to the members of the rescue teams an alternative communication channel to exchange information when traditional wired or wireless technologies are either damaged or unavailable. Despite the focus on WSNs, arguably one of the most challenging networking scenarios, Reins-MAC defines generic scheduling mechanisms that apply to multiple contexts. In particular, mobile platforms, e.g., phones as well as the already mentioned unmanned vehicles, are gaining increasing attention as sensing and networking devices. Despite the more powerful platform, their interconnection remains a challenge. This scenario would investigate the limits of the introduced scheduling flexibility as well as the effectiveness of Reins-MAC anarchy in highly dynamic settings.

Part VI

Bibliography

Bibliography

[1] G. Barrenetxea, F. Ingelrest, G. Schaefer, and M. Vetterli. The hitchhiker's guide to successful wireless sensor network deployments. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, SenSys '08, pages 43–56, New York, NY, USA, 2008. ACM.

[2] J. Bonwick. The slab allocator: an object-caching kernel memory allocator. In *Proceedings of the USENIX Summer 1994 Technical Conference on USENIX Summer 1994 Technical Conference - Volume 1*, USTC'94, pages 6–6, Berkeley, CA, USA, 1994. USENIX Association.

[3] A. Boulis. Castalia: A simulator for WSNs. `http://castalia.npc.nicta.com.au/index.php`, 2010.

[4] M. Buettner, G. V. Yee, E. Anderson, and R. Han. X-MAC: a short preamble MAC protocol for duty-cycled wireless sensor networks. In *Proceedings of the 4th international conference on Embedded networked sensor systems*, SenSys '06, pages 307–320, New York, NY, USA, 2006. ACM.

[5] N. Burri, P. von Rickenbach, and R. Wattenhofer. Dozer: ultra-low power data gathering in sensor networks. In *Proceedings of the 6th international Conference on Information processing in sensor networks*, IPSN '07, pages 450–459, New York, NY, USA, 2007. ACM.

[6] U. I. centric Government & Enterprise Next Generation Networks-Vision 2010. `www.u-2010.eu`.

[7] M. Ceriotti, M. Chini, A. L. Murphy, G. P. Picco, F. Cagnacci, and B. Tolhurst. Motes in the jungle: lessons learned from a short-term WSN deployment in the ecuador cloud forest. In *Proceedings of the 4th international workshop on Real-world wireless sensor networks*, REALWSN'10, pages 25–36, Berlin, Heidelberg, 2010. Springer-Verlag.

[8] M. Ceriotti, M. Corra, L. D'Orazio, R. Doriguzzi, D. Facchin, S. Guna, G. P. Jesi, R. Lo Cigno, L. Mottola, A. L. Murphy, M. Pescalli, G. P. Picco, D. Pregnolato, and C. Torghele. Is there light at the ends of the tunnel? wireless sensor networks for adaptive

lighting in road tunnels. In *Proceedings of the 2011 International Conference on Information Processing in Sensor Networks*, IPSN '11, 2011. To Appear.

[9] M. Ceriotti, L. Mottola, G. P. Picco, A. L. Murphy, S. Guna, M. Corra, M. Pozzi, D. Zonta, and P. Zanon. Monitoring heritage buildings with wireless sensor networks: The torre aquila deployment. In *Proceedings of the 2009 International Conference on Information Processing in Sensor Networks*, IPSN '09, pages 277–288, Washington, DC, USA, 2009. IEEE Computer Society.

[10] S. Cheekiralla. Poster abstract: Wireless sensor network-based tunnel monitoring. In *Proceedings of the 1st international workshop on Real-world wireless sensor networks*, REALWSN'05, 2005.

[11] K. Chintalapudi, T. Fu, J. Paek, N. Kothari, S. Rangwala, J. Caffrey, R. Govindan, E. Johnson, and S. Masri. Monitoring civil structures with a wireless sensor network. *IEEE Internet Computing*, 10:26–34, March 2006.

[12] K. Chintalapudi, J. Paek, O. Gnawali, T. S. Fu, K. Dantu, J. Caffrey, R. Govindan, E. Johnson, and S. Masri. Structural damage detection and localization using NETSHM. In *Proceedings of the 5th international conference on Information processing in sensor networks*, IPSN '06, pages 475–482, New York, NY, USA, 2006. ACM.

[13] CIE—International Commission on Illumination. *Guide for the Lighting of Road Tunnels and Underpasses (CIE 88-2004)*, 2004.

[14] CONET Consortium. Roadmap. `http://www.cooperating-objects.eu/roadmap/`, 2009.

[15] P. Costa, G. Coulson, R. Gold, M. Lad, C. Mascolo, L. Mottola, G. P. Picco, T. Sivaharan, N. Weerasinghe, and S. Zachariadis. The RUNES middleware for networked embedded systems and its application in a disaster management scenario. In *Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications*, pages 69–78, Washington, DC, USA, 2007. IEEE Computer Society.

[16] P. Costa, L. Mottola, A. L. Murphy, and G. P. Picco. Programming wireless sensor networks with the TeenyLime middleware. In *Proceedings of the ACM/IFIP/USENIX 2007 International Conference on Middleware*, Middleware '07, pages 429–449, New York, NY, USA, 2007. Springer-Verlag New York, Inc.

[17] D. S. J. De Couto, D. Aguayo, J. Bicket, and R. Morris. A high-throughput path metric for multi-hop wireless routing. *Wireless Networks*, 11:419–434, July 2005.

[18] J. Degesys and R. Nagpal. Towards desynchronization of multi-hop topologies. In *Proceedings of the 2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, pages 129–138, Washington, DC, USA, 2008. IEEE Computer Society.

[19] J. Degesys, I. Rose, A. Patel, and R. Nagpal. DESYNC: self-organizing desynchronization and tdma on wireless sensor networks. In *Proceedings of the 6th international conference on Information processing in sensor networks*, IPSN '07, pages 11–20, New York, NY, USA, 2007. ACM.

[20] V. Dyo, S. A. Ellwood, D. W. Macdonald, A. Markham, C. Mascolo, B. Pásztor, S. Scellato, N. Trigoni, R. Wohlers, and K. Yousef. Evolution and sustainability of a wildlife monitoring sensor network. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, SenSys '10, pages 127–140, New York, NY, USA, 2010. ACM.

[21] A. El-Hoiydi and J.-D. Decotignie. WiseMAC: An ultra low power MAC protocol for multi-hop wireless sensor networks. In *First International Workshop on Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSORS 2004), Lecture Notes in Computer Science, LNCS 3121*, pages 18–31. Springer-Verlag, July 2004.

[22] J. Elson, L. Girod, and D. Estrin. Fine-grained network time synchronization using reference broadcasts. In *Proceedings of the 5th symposium on Operating systems design and implementation*, OSDI '02, pages 147–163, New York, NY, USA, 2002. ACM.

[23] S. C. Ergen and P. Varaiya. PEDAMACS: Power efficient and delay aware medium access protocol for sensor networks. *IEEE Transactions on Mobile Computing*, 5:920–930, July 2006.

[24] S. Ganeriwal, R. Kumar, and M. B. Srivastava. Timing-sync protocol for sensor networks. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, SenSys '03, pages 138–149, New York, NY, USA, 2003. ACM.

[25] A. Giusti, A. L. Murphy, and G. P. Picco. Decentralized scattering of wake-up times in wireless sensor networks. In *Proceedings of the 4th European conference on Wireless sensor networks*, EWSN'07, pages 245–260, Berlin, Heidelberg, 2007. Springer-Verlag.

[26] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection tree protocol. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, SenSys '09, pages 1–14, New York, NY, USA, 2009. ACM.

[27] O. Gnawali, K.-Y. Jang, J. Paek, M. Vieira, R. Govindan, B. Greenstein, A. Joki, D. Estrin, and E. Kohler. The tenet architecture for tiered sensor networks. In *Proceedings of the 4th international conference on Embedded networked sensor systems*, SenSys '06, pages 153–166, New York, NY, USA, 2006. ACM.

[28] Gumstix. `www.gumstix.com`.

[29] G. P. Halkes and K. G. Langendoen. Crankshaft: an energy-efficient mac-protocol for dense wireless sensor networks. In *Proceedings of the 4th European conference on Wireless sensor networks*, EWSN'07, pages 228–244, Berlin, Heidelberg, 2007. Springer-Verlag.

[30] Q. Han, A. P. Jayasumana, T. H. Illangasekare, and T. Sakaki. A wireless sensor network based closed-loop system for subsurface contaminant plume monitoring. In *Proceedings of the 22nd IEEE International Symposium on Parallel and Distributed Processing*, IPDPS '08, pages 1–5. IEEE, 2008.

[31] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. *SIGPLAN Notices*, 35:93–104, November 2000.

[32] W. X. Hu, 2010. Duracell Corp - Private correspondence.

[33] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[34] Institute of Electrical and Electronics Engineers. Inc. IEEE Standard 802 for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements - Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (LR-WPANs), September 2006.

[35] B. Karaoglu, T. Numanoglu, and W. Heinzelman. Adaptation of TDMA parameters based on network conditions. In *Proceedings of the 2009 IEEE Conference on Wireless Communications & Networking Conference*, WCNC'09, pages 1830–1835, Piscataway, NJ, USA, 2009. IEEE Press.

[36] S. Kim, S. Pakzad, D. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon. Health monitoring of civil infrastructures using wireless sensor networks. In *Proceedings of the 6th international conference on Information processing in sensor networks*, IPSN '07, pages 254–263, New York, NY, USA, 2007. ACM.

[37] Y. Kim, R. G. Evans, and W. M. Iversen. Evaluation of closed-loop site-specific irrigation with wireless sensor network. *Journal of Irrigation anf Drainage Engineering*, 135(1):25–31, 2009.

[38] K. Langendoen. The mac alphabet soup. `http://www.st.ewi.tudelft.nl/~koen/MACsoup/`.

[39] K. Langendoen. Medium access control in wireless sensor networks. In H. Wu and Y. Pan, editors, *Medium Access Control in Wireless Networks*, pages 535–560. Nova Science Publishers, Inc., May 2008.

[40] K. Langendoen, A. Baggio, and O. Visser. Murphy loves potatoes: experiences from a pilot sensor network deployment in precision agriculture. In *Proceedings of the 20th international Conference on Parallel and distributed processing*, IPDPS'06, pages 174–174, Washington, DC, USA, 2006. IEEE Computer Society.

[41] P. Levis, N. Patel, D. Culler, and S. Shenker. Trickle: a self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation - Volume 1*, pages 2–2, Berkeley, CA, USA, 2004. USENIX Association.

[42] M. Li and Y. Liu. Underground coal mine monitoring with wireless sensor networks. *ACM Transations on Sensor Network*, 5:10:1–10:29, April 2009.

[43] Q. Li and D. Rus. Global clock synchronization in sensor networks. *IEEE Transactions on Computers*, 55:214–226, February 2006.

[44] K. Lin and P. Levis. Data discovery and dissemination with DIP. In *Proceedings of the 7th international conference on Information processing in sensor networks*, IPSN '08, pages 433–444, Washington, DC, USA, 2008. IEEE Computer Society.

[45] W. Liu, J. C. Principe, and S. Haykin. *Kernel Adaptive Filtering: A Comprehensive Introduction*. Wiley Publishing, 1st edition, 2010.

[46] G. Lu, B. Krishnamachari, and C. Raghavendra. An adaptive energy-efficient and low-latency MAC for data gathering in wireless sensor networks. In *Proceeding of the 18th Parallel and Distributed Processing Symposium*, page 224, April 2004.

[47] J. P. Lynch and K. J. Loh. A summary review of wireless sensors and sensor networks for structural health monitoring. *Shock and Vibration Digest*, Mar 2006.

[48] J. P. Lynch, A. Sundararajan, K. H. Law, A. S. Kiremidjian, and E. Carryer. Power-efficient data management for a wireless structural monitoring system. In *Proceedings of the* 4^{th} *International Wrkshp. on Structural Health Monitoring*, 2003.

[49] J. P. Lynch, Y. Wang, R. A. Swartz, K. C. Lu, and C. H. Loh. Implementation of a closed-loop structural control system using wireless sensor networks. *Structural Control and Health Monitoring*, 15(4):518–539, 2008.

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

[51] M. Maróti, B. Kusy, G. Simon, and A. Lédeczi. The flooding time synchronization protocol. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, SenSys '04, pages 39–49, New York, NY, USA, 2004. ACM.

[52] D3S Research Group. `teenylime.sourceforge.net`.

[53] E. Miluzzo, X. Zheng, K. Fodor, and A. T. Campbell. Radio characterization of 802.15.4 and its impact on the design of mobile sensor networks. In *Proceedings of the 5th European conference on Wireless sensor networks*, EWSN'08, pages 171–188, Berlin, Heidelberg, 2008. Springer-Verlag.

[54] R. E. Mirollo and S. H. Strogatz. Synchronization of pulse-coupled biological oscillators. *SIAM Journal on Applied Mathematics*, 50(6):1645–1662, 1990.

[55] J.-M. Molina-Garcia-Pardo, M. Lienard, and P. Degauque. Propagation in tunnels: experimental investigations and channel modeling in a wide frequency band for MIMO applications. *EURASIP J. Wirel. Commun. Netw.*, 2009:7:1–7:9, January 2009.

[56] D. Moss and P. Levis. BoX-MACs: Exploiting physical and link layer boundaries in low-power networking. Technical Report SING-08-00, Stanford University, 2008.

[57] L. Mottola, G. P. Picco, M. Ceriotti, c. Gună, and A. L. Murphy. Not all wireless sensor networks are created equal: A comparative study on tunnels. *ACM Transactions on Sensor Networks*, 7:15:1–15:33, September 2010.

[58] C. Mühlberger and R. Kolla. Extended desynchronization for multi-hop topologies. Technical Report 460, Institut für Informatik, Universität Würzburg, Institut fr Informatik, 2009.

[59] R. Musaloiu-E., C.-J. M. Liang, and A. Terzis. Koala: Ultra-low power data retrieval in wireless sensor networks. In *Proceedings of the 7th international Conference on Information processing in sensor networks*, IPSN '08, pages 421–432, Washington, DC, USA, 2008. IEEE Computer Society.

[60] Octopus Home Page. `http://csserver.ucd.ie/~rjurdak/Octopus.htm`.

[61] C. Park, K. Lahiri, and A. Raghunathan. Battery discharge characteristics of wireless sensor nodes: an experimental analysis. In *Proceedings of the 2nd International Conference on Sensor and Ad-hoc Communications and Networks*, SECON '05, pages 430 – 440, September 2005.

[62] H. Park, J. Burke, and M. B. Srivastava. Design and implementation of a wireless sensor network for intelligent light control. In *Proceedings of the 6th international conference on*

Information processing in sensor networks, IPSN '07, pages 370–379, New York, NY, USA, 2007. ACM.

[63] S. Park, A. Savvides, and M. Srivastava. Battery capacity measurement and analysis using lithium coin cell battery. In *Proceedings of the 2001 international symposium on Low power electronics and design*, ISLPED '01, pages 382–387, New York, NY, USA, 2001. ACM.

[64] C. S. Peskin. *Mathematical aspects of heart physiology.* Courant Institute of Mathematical Sciences, New York University, New York, NY, USA, 1975.

[65] J. Polastre, J. Hill, and D. Culler. Versatile low power media access for wireless sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, SenSys '04, pages 95–107, New York, NY, USA, 2004. ACM.

[66] J. Polastre, R. Szewczyk, and D. Culler. Telos: enabling ultra-low power wireless research. In *Proceedings of the 4th international symposium on Information processing in sensor networks*, IPSN '05, Piscataway, NJ, USA, 2005. IEEE Press.

[67] V. Rajendran, J. Garcia-Luna-Aceves, and K. Obraczka. Energy-efficient, application-aware medium access for sensor networks. In *2nd IEEE Conference on Mobile Ad-hoc and Sensor Systems (MASS 2005)*, Washington, DC, Nov. 2005.

[68] V. Rajendran, K. Obraczka, and J. J. Garcia-Luna-Aceves. Energy-efficient collision-free medium access control for wireless sensor networks. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, SenSys '03, pages 181–192, New York, NY, USA, 2003. ACM.

[69] B. Raman and K. Chebrolu. Censor networks: a critique of "sensor networks" from a systems perspective. *SIGCOMM Computing Communincation Review*, 38:75–78, July 2008.

[70] I. Rhee, A. Warrier, M. Aia, and J. Min. Z-MAC: a hybrid MAC for wireless sensor networks. In *Proceedings of the 3rd international conference on Embedded networked sensor systems*, SenSys '05, pages 90–101, New York, NY, USA, 2005. ACM.

[71] I. Rhee, A. Warrier, J. Min, and L. Xu. DRAND: distributed randomized TDMA scheduling for wireless ad-hoc networks. In *Proceedings of the 7th ACM international symposium on Mobile ad hoc networking and computing*, MobiHoc '06, pages 190–201, New York, NY, USA, 2006. ACM.

[72] 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 5th international Conference on Embedded networked sensor systems*, SenSys '07, pages 103–116, New York, NY, USA, 2007. ACM.

[73] V. Singhvi, A. Krause, C. Guestrin, J. H. Garrett, Jr., and H. S. Matthews. Intelligent light control using sensor networks. In *Proceedings of the 3rd international conference on Embedded networked sensor systems*, SenSys '05, pages 218–229, New York, NY, USA, 2005. ACM.

[74] K. Srinivasan, P. Dutta, A. Tavakoli, and P. Levis. An empirical study of low-power wireless. *ACM Transactions on Sensor Networks*, 6:16:1–16:49, March 2010.

[75] I. Stoianov, L. Nachman, S. Madden, and T. Tokmouline. PIPENET: a wireless sensor network for pipeline monitoring. In *Proceedings of the 6th international conference on Information processing in sensor networks*, IPSN '07, pages 264–273, New York, NY, USA, 2007. ACM.

[76] Z. Sun and I. Akyildiz. Channel modeling of wireless networks in tunnels. In *Proceedings of the International Global Telecommunications Conference*, GLOBECOM '08, pages 1 –5, December 2008.

[77] TinyOS Official Source Tree. `www.tinyos.net`.

[78] TinyOS. TEP 105 - Low Power Listening. `www.tinyos.net`. Accessed on 1/2010.

[79] TinyOS. TEP 119 - Collection. `www.tinyos.net`. Accessed on 1/2010.

[80] 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 3rd international conference on Embedded networked sensor systems*, SenSys '05, pages 51–63, New York, NY, USA, 2005. ACM.

[81] L. van Hoesel and P. Havinga. A lightweight medium access protocol (LMAC) for wireless sensor networks: Reducing preamble transmissions and transceiver state switches. In *1st International Workshop on Networked Sensing Systems (INSS, Tokio, Japan*, pages 205–208, Tokio, Japan, 2004. Society of Instrument and Control Engineers (SICE).

[82] G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh. Fidelity and yield in a volcano monitoring sensor network. In *Proceedings of the 7th symposium on Operating systems design and implementation*, OSDI '06, pages 381–396, Berkeley, CA, USA, 2006. USENIX Association.

[83] G. Werner-Allen, G. Tewari, A. Patel, M. Welsh, and R. Nagpal. Firefly-inspired sensor network synchronicity with realistic radio effects. In *Proceedings of the 3rd international conference on Embedded networked sensor systems*, SenSys '05, pages 142–153, New York, NY, USA, 2005. ACM.

[84] A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, SenSys '03, pages 14–27, New York, NY, USA, 2003. ACM.

[85] N. Xu, S. Rangwala, K. K. Chintalapudi, D. Ganesan, A. Broad, R. Govindan, and D. Estrin. A wireless sensor network for structural monitoring. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, SenSys '04, pages 13–24, New York, NY, USA, 2004. ACM.

[86] G. Zhou, T. He, S. Krishnamurthy, and J. A. Stankovic. Impact of radio irregularity on wireless sensor networks. In *Proceedings of the 2nd international conference on Mobile systems, applications, and services*, MobiSys '04, pages 125–138, New York, NY, USA, 2004. ACM.

[87] D. Zonta, M. Pozzi, and P. Zanon. Manging the historical heritage using distributed technologies. *International Journal of Architectural Heritage*, 2(3):200–225, 2008.

[88] D. Zonta, M. Pozzi, P. Zanon, G. A. Anese, and A. Busetto. Real-time probabilisitc health monitoring of the portogruaro civic tower. In *Proceedings of the* 6^{th} *International Conference on Structural Analysis of Historical Constructions*, SAHC '08, pages 723–731, 2008.

Part VII

Publications

Publications

International Journals

Authors Luca Mottola, Gian Pietro Picco, Matteo Ceriotti, Ştefan Gună, and Amy L. Murphy

Title Not All Wireless Sensor Networks Are Created Equal: A Comparative Study On Tunnels

Journal *ACM Transactions on Sensor Networks*, Volume 7, Issue 2, August 2010

Abstract Wireless sensor networks (WSNs) are envisioned for a number of application scenarios. Nevertheless, the few in-the-field experiences typically focus on the features of a specific system, and rarely report about the characteristics of the target environment, especially w.r.t. the behavior and performance of low-power wireless communication. The TRITon project, funded by our local administration, aims to improve safety and reduce maintenance costs of road tunnels, using a WSN-based control infrastructure. The access to real tunnels within TRITon gives us the opportunity to experimentally assess the peculiarities of this environment, hitherto not investigated in the WSN field. We report about three deployments: *i)* an operational road tunnel, enabling us to assess the impact of vehicular traffic; *ii)* a non-operational tunnel, providing insights into analogous scenarios (e.g., underground mines) without vehicles; *iii)* a vineyard, serving as a baseline representative of the existing literature. Our setup, replicated in each deployment, uses mainstream WSN hardware, and popular MAC and routing protocols. We analyze and compare the deployments w.r.t. reliability, stability, and asymmetry of links, the accuracy of link quality estimators, and the impact of these aspects on MAC and routing layers. Our analysis shows that a number of criteria commonly used in the design of WSN protocols do not hold in tunnels. Therefore, our results are useful for designing networking solutions operating efficiently in similar environments.

Authors Daniele Zonta, Huayong Wu, Matteo Pozzi, Paolo Zanon, Matteo Ceriotti, Luca Mottola, Gian Pietro Picco, Amy L. Murphy, Ştefan Gună, and Michele Corrá

Title Wireless Sensor Networks for Permanent Health Monitoring of Historic Constructions

Journal *International Journal on Smart Structures and Systems (SPIE), Special Issue on Wireless Sensor Advances and Applications for Civil Infrastructure Monitoring*, Volume 6, Issue 5-6, June 2010

Abstract This paper describes the application of a wireless sensor network to a 31 meter-tall medieval tower located in the city of Trento, Italy. The effort is motivated by preservation of the integrity of a set of frescoes decorating the room on the second floor, representing one of most important International Gothic artworks in Europe. The specific application demanded development of customized hardware and software. The wireless module selected as the core platform allows reliable wireless communication at low cost with a long service life. Sensors include accelerometers, deformation gauges, and thermometers. A multi-hop data collection protocol was applied in the software to improve the system's flexibility and scalability. The system has been operating since September 2008, and in recent months the data loss ratio was estimated as less than 0.01%. The data acquired so far are in agreement with the prediction resulting a priori from the 3-dimensional FEM. Based on these data a Bayesian updating procedure is employed to real-time estimate the probability of abnormal condition states. This first period of operation demonstrated the stability and reliability of the system, and its ability to recognize any possible occurrence of abnormal conditions that could jeopardize the integrity of the frescos.

International Conferences and Workshops

Authors Matteo Ceriotti, Michele Corrá, Leandro D'Orazio, Roberto Doriguzzi, Daniele Facchin, Ştefan Gună, Gian Paolo Jesi, Renato Lo Cigno, Luca Mottola, Amy L. Murphy, Massimo Pescalli, Gian Pietro Picco, Denis Pregnolato, and Carloalberto Torghele

Title Is There Light at the Ends of the Tunnel? Wireless Sensor Networks for Adaptive Lighting in Road Tunnels

Venue 10th *ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN'11, SPOTS track)*, Chicago (IL, USA), April 2011 (*Best Paper Award*)

Abstract Existing deployments of wireless sensor networks (WSNs) are often conceived as stand-alone monitoring tools. In this paper, we report instead on a deployment where the WSN is a key component of a closed-loop control system for adaptive lighting in operational road tunnels. WSN nodes along the tunnel walls report light readings to a control station, which closes the loop by setting the intensity of lamps to match a legislated curve. The ability to match dynamically the lighting levels to the actual environmental conditions improves the tunnel safety and reduces its power consumption. The use of WSNs in a closed-loop system, combined with the real-world, harsh setting of operational road tunnels, induce tighter requirements on the quality and timeliness of sensed data, as well as on the reliability and lifetime of the network. In this work, we test to what extent mainstream WSN technology meets these challenges, using a dedicated design that however relies on well-established techniques. The paper describes the hw/sw architecture we devised by focusing on the WSN component, and analyzes its performance through experiments in a real, operational tunnel.

Authors Matteo Ceriotti, Matteo Chini, Amy L. Murphy, Gian Pietro Picco, Francesca Cagnacci, and Bryony Tolhurst

Title Motes in the Jungle: Lessons Learned from a Short-term WSN Deployment in the Ecuador Cloud Forest

Venue 4th *Workshop on Real-World Wireless Sensor Networks (RealWSN'10)*, Colombo (Sri Lanka), December 2010

Abstract We study the characteristics of the communication links of a wireless sensor network in a tropical cloud forest in Ecuador, in the context of a wildlife monitoring application. Thick vegetation and high humidity are in principle a challenge for the IEEE 802.15.4 radio we employed. We performed experiments with stationary-only nodes as well as in combination with mobile ones. Due to logistics, all the experiments were performed in isolation by the biologists on our team. In addition to discussing the characteristics of links in this previously unstudied environment, we also discuss the lessons we learned from operating under peculiar constraints in a peculiar deployment scenario.

Authors Matteo Ceriotti, Luca Mottola, Gian Pietro Picco, Amy L. Murphy, Ştefan Gună, Michele Corrá, Matteo Pozzi, Daniele Zonta, and Paolo Zanon

Title Monitoring Heritage Buildings with Wireless Sensor Networks: The Torre Aquila Deployment

Venue 8th *ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN'09, SPOTS track)*, San Francisco (CA, USA), April 2009 (*Best Paper Award*)

Abstract Wireless sensor networks are untethered infrastructures that are easy to deploy and have limited visual impact—a key asset in monitoring heritage buildings of artistic interest. This paper describes one such system deployed in Torre Aquila, a medieval tower in Trento (Italy). Our contributions range from the hardware to the graphical front-end. Customized hardware deals efficiently with high-volume vibration data, and specially-designed sensors acquire the building's deformation. Dedicated software services provide: *i)* data collection, to efficiently reconcile the diverse data rates and reliability needs of heterogeneous sensors;

ii) data dissemination, to spread configuration changes and enable remote tasking; *iii)* time synchronization, with low memory demands. Unlike most deployments, built directly on the operating system, our entire software layer sits atop our TeenyLime middleware. Based on 4 months of operation, we show that our system is an effective tool for assessing the tower's stability, as it delivers data reliably (with loss ratios $< 0.01\%$) and has an estimated lifetime beyond one year.

Authors Matteo Ceriotti, Amy L. Murphy, and Gian Pietro Picco

Title Data Sharing vs. Message Passing: Synergy or Incompatibility? An Implementation-Driven Case Study

Venue 23rd *Annual ACM Symposium on Applied Computing (SAC'08)*, Fortaleza, Brazil, March 2008

Abstract One reasonable categorization of coordination models is into *data sharing* or *message passing*, based on whether the information necessary to coordination is persistently stored and shared, or instead is only transiently available during communication. Generally speaking, approaches based on data sharing are more expressive and provide full decoupling in space and time. The alternative approach requires the simultaneous presence of the coordinated parties, but is typically more scalable. Prominent examples are, respectively, tuple spaces and publish-subscribe. An open research question is whether it is possible to exploit in synergy the best of these two approaches, e.g. by implementing the more complex data sharing coordination on top of the more lightweight message passing one. In this paper, we seek an answer to this question in a pragmatic way: we analyze an implementation of the Lime tuple space middleware on top of REDS, an open source publish-subscribe system. Our implementation-driven style of investigation forces us to face details that do not surface when reasoning in the abstract about the nature and expressiveness of the models. We report about lessons we learned in this experience, and propose an extension to the publish-subscribe model that, albeit useful per se, constitutes a more effective foundation for data sharing coordination models.

Posters and Demos

Authors Matteo Ceriotti, and Amy L. Murphy

Title Demo Abstract: A MAC Contest between LPL (the Champion) and Reins-MAC (the Challenger, an Anarchic TDMA Scheduler Providing QoS)

Venue 8th *ACM International Conference on Embedded Networked Sensor Systems (SENSYS)*, Zurich, Switzerland, November 2010

Authors Matteo Ceriotti, and Amy L. Murphy

Title Poster Abstract: Introducing an adaptive MAC layer to support Quality of Service in WSN

Venue 5th *European conference on Wireless Sensor Networks (EWSN'08)*, Bologna, Italy, January 2008

Non-refereed Publications

Authors Daniele Zonta, Matteo Pozzi, Huayong Wu, Paolo Zanon, Matteo Ceriotti, Luca Mottola, Gian Pietro Picco, Amy L. Murphy, and Ştefan Gună

Title Real-Time Health Monitoring of Historic Buildings with Wireless Sensor Networks

Venue 7th *International Workshop on Structural Health Monitoring (IWSHM'09)*, Stanford (CA, USA), September 2009.

Authors Matteo Ceriotti, Roberto Doriguzzi, Ştefan Gună, Renato Lo Cigno, Luca Mottola, Amy L. Murphy, Matteo Nardelli, Gian Pietro Picco, and Carloalberto Torghele

Title Demo Abstract: Adaptive Lighting in Road Tunnels Using Wireless Sensor Networks

Venue 1st *European TinyOS Technology Exchange (ETTX'09)*, Cork (Ireland), February 2009.