RELYonIT
dependability for the internet of things

Research by Experimentation
for Dependability on the
Internet of Things

SEVENTH FRAMEWORK
PROGRAMME

D-4.2 – Prototype of Testbeds with Realistic Environmental Effects

Grant Agreement no: 317826
www.relyonit.eu

Date: October 31, 2013

Author(s) and affiliation: Carlo Alberto Boano (TUG), Felix Jonathan Oppermann (TUG), Kay Römer (TUG), James Brown (ULANC), Utz Roedig (ULANC), Chamath Keppitiyagama (SICS), and Thiemo Voigt (SICS).

Work package/task: WP4
Document status: Final
Dissemination level: Public
Keywords: JamLab, TempLab, Testbed, Wireless Sensor Networks.

Abstract This document presents the design and implementation of two wireless sensor network testbed extensions that enable the repeatable generation of controlled environmental conditions. First, we present TempLab, an extension for wireless sensor network testbeds that allows to control the on-board temperature of sensor nodes and to study the effects of temperature variations on the network performance in a precise and repeatable fashion. Second, we present JamLab, a low-cost infrastructure to augment existing sensornet testbeds with accurate interference generation while limiting the overhead to a simple software upload. These testbed extensions play a crucial role for the investigation of protocol performance, as they allow to rerun experiments under identical environmental conditions.

Disclaimer

The information in this document is proprietary to the following RELYonIT consortium members: Graz University of Technology, SICS Swedish ICT, Technische Universiteit Delft, University of Lancaster, Worldsensing, Acciona Infraestructuras S.A.

The information in this document is provided "as is", and no guarantee or warranty is given that the information is fit for any particular purpose. The user uses the information at his sole risk and liability. The above referenced consortium members shall have no liability for damages of any kind including without limitation direct, special, indirect, or consequential damages that may result from the use of these materials subject to any liability which is mandatory due to applicable law.

Contents

List of Figures

List of Tables

Executive Summary

To understand how the environment affects the performance and the operation of IoT protocols it is fundamental to be able to rerun experiments under identical environmental conditions. This deliverable, created in the context of Task 4.1 (*Testbeds with Realistic Environmental Effects*) of WP4 (*Experimentation and Applications*), presents the design and implementation of two IoT testbed extensions that enable the repeatable playback of environmental conditions.

Within RELYonIT, we consider as primary environmental factors temperature and radio interference, and this deliverable presents the design and implementation of:

1. **TempLab**: a low-cost extension for sensornet testbeds that allows to study the impact of temperature on wireless sensor hardware and protocols;

2. **JamLab**: a low-cost extension for sensornet testbeds that allows the creation of realistic and repeatable interference patterns.

These testbed extensions allow to rerun experiments under identical environmental conditions and hence play a crucial role for the investigation of protocol performance. TempLab can accurately reproduce temperature traces recorded in outdoor environments with an average error of only 0.1°C, and can also be used with specific test patterns (e.g., a series of cold and warm periods) that vary temperature with a specified frequency, allowing quick debugging of protocol behaviour. TempLab plays an important role in examining and quantifying the effects of temperature variations on sensornet applications and protocols, as it can reveal system limitations that would not have been visible when experimenting with existing testbed installations. Our preliminary experiments using TempLab have indeed revealed that high temperature can drastically change the topology of a network and lead to network partitions, reduce significantly the performance of MAC protocols, as well as increase the processing delay in the network.

JamLab is a low-cost extension for sensornet testbeds we have developed in earlier work that allows the creation of realistic and repeatable interference patterns. JamLab provides simple models to emulate the interference patterns generated by several devices, and a playback capability to regenerate recorded interference patterns. As it does not require additional hardware and the overhead is limited to a simple software upload, JamLab can be used to generate controllable interference to test the performance of protocols, as successfully shown in [12], [23], and [25]. We have extended JamLab and introduced an automatic testbed configuration that allows an optimal selection of the jammers within the network. Previously, the configuration of the testbed had to be carried out manually, a time-consuming task that did not guarantee the creation of optimal jammer configurations.

1. Introduction

The IoT and specifically also the FIRE research community traditionally relies on testbed facilities to evaluate and tune newly developed methods, protocols, and applications under realistic conditions in a cost-effective way. A large number of publicly available testbeds have been developed in the last decade, where registered users can typically upload the specifications of an experiment and collect traces directly via a web interface. Examples are MoteLab [49], one of the first open-source wireless sensor networks testbeds to be developed (and still one of the largest, with its 190 nodes deployed over 3 floors), Kansei [24], (210 sensor nodes with a gateway station attached to each of the sensor nodes) Indriya [20], (127 TelosB nodes deployed at the National University of Singapore), TWIST [27], (200 heterogeneous nodes across several floors in a building in the campus of the Technical University of Berlin, Germany) and NetEye [29]. (130 TelosB mote deployed at Wayne State University, MI, USA).

The accuracy of a testbed experiment, however, largely depends on how accurately environmental effects can be reproduced. Recent efforts have looked at extending existing infrastructures with the emulation of environmental effects such as the mobility of nodes [28], [37]. In ViMobiO [37], Puccinelli and Giordano implemented a virtual mobility overlay to reproduce movement patterns of nodes during experimental evaluation. In the CONET testbed [17], a swarm of five Pioneer 3-AT autonomous robots communicates with a static wireless sensor network [2], [3]. Similarly, in Mobile Emulab [28], several robots carrying Mica2 nodes can move in a $25m^2$ space.

Within RELYonIT, the primary environmental factors considered are temperature and radio interference, which have not received significant attention in the community even though they can dramatically affect the performance of wireless sensor networks. Slipp et al. [41] have developed a testbed facility to replicate the harsh RF conditions of an offshore oil platform by means of a VSG-based EMI generator. In the context of the CREW project [19], several testbed facilities were augmented with state-of-the-art cognitive systems to allow research on advanced spectrum sensing and cognitive networking strategies. However, replicating these approaches in FIRE facilities can be very costly, as it requires expensive equipment to generate realistic interference patterns.

Concerning temperature, instead, we are not aware of testbed infrastructures that allow to vary the on-board temperature of wireless sensor nodes in a repeatable fashion. Industry makes heavy use of temperature chambers during device verification processes (e.g., to calibrate sensors and transceivers [16]), but such solutions are not suitable due to their high cost and because they target individual components and not *a network* of nodes, which is necessary to disclose limitations at the communication level.

In this deliverable, we presents the design and implementation of TempLab [14] and JamLab [11], low-cost extensions for sensornet testbeds that allow to study the impact of temperature and interference on the performance of wireless sensor networks. These testbed extensions allow to rerun experiments under identical environmental conditions and hence play a crucial

role for the investigation of protocol performance in WP1 [13, 15, 51] and WP2 [12].

Chapter 2 describes the design and implementation of TempLab, an extension for wireless sensor network testbeds that allows to control the on-board temperature of sensor nodes and to study the effects of temperature variations on the network performance in a precise and repeatable fashion. Chapter 3 describes the design and implementation of JamLab, a low-cost flexible testbed infrastructure that allows the repeatable generation of a wide range of interference patterns. We conclude the deliverable and draw our conclusions in Chapter 4.

2. TempLab: a Testbed to Study the Impact of Temperature on Wireless Sensor Networks

Research and industrial deployments have shown that the operations of wireless sensor networks are largely affected by the on-board temperature of sensor nodes. Temperature variations may significantly affect, among others, clock drift [39], battery capacity and discharge [34], as well as the quality of wireless links [5].

Depending on the packaging and deployment location, the electronics of wireless sensor nodes may experience a substantial temperature variation. Sunshine may easily heat a packaged sensor node up to 70 degrees Celsius – especially if the packaging absorbs infra-red (IR) radiation [10], and long term outdoor deployments have shown that the on-board temperature can vary by as much as 35°C in one hour and 56°C over a day [13]. This variation is sufficient to cause a frequency offset of more than 100 ppm on the crystal oscillator frequency [33], which can affect the rendezvous process of synchronous duty-cycled MAC protocols. Such temperature change is also enough to reduce the received signal strength between two sensor nodes by more than 6 dB [13], which can change the packet reception rate (PRR) of the link from 100% to 0%. Hence, a deep analysis of how temperature affects the operation of sensor networks is necessary to inform the design of robust and dependable applications.

Analytical models or simulation tools that can accurately predict the impact of temperature are hard to obtain due to the complexity of the involved physical processes. Similarly, setting up a pilot deployment of a sensor network to evaluate the impact of temperature can be very complex and time-consuming. On the one hand, meteorological conditions cannot be controlled, making it impossible to ensure repeatability across several experiments. On the other hand, temperature profiles that can be tested are highly specific to the deployment location and to the time of the year in which the experiment is carried out (i.e., to get a flavour of seasonal temperature variations, the pilot deployment should last several months). What is needed to overcome these limitations is an experimental facility that allows researchers and system designers to mimic the temperature variations normally found in outdoor deployments in a fast and simple way.

Traditional testbed facilities used to evaluate protocols and applications under realistic conditions in a cost effective manner such as MoteLab [49], TWIST [27], Kansei [24], and NetEye [29], do not allow the evaluation of temperature effects. To date, a low-cost flexible testbed infrastructure that allows the repeatable generation of predefined temperature patterns across a sensor network still does not exist. Industry makes heavy use of temperature chambers during device verification processes (e.g., to calibrate sensors and transceivers [16]), but such solutions are not suitable due to their high cost and because they target individual components and not a network of nodes, which is necessary to disclose limitations at the communication level. We

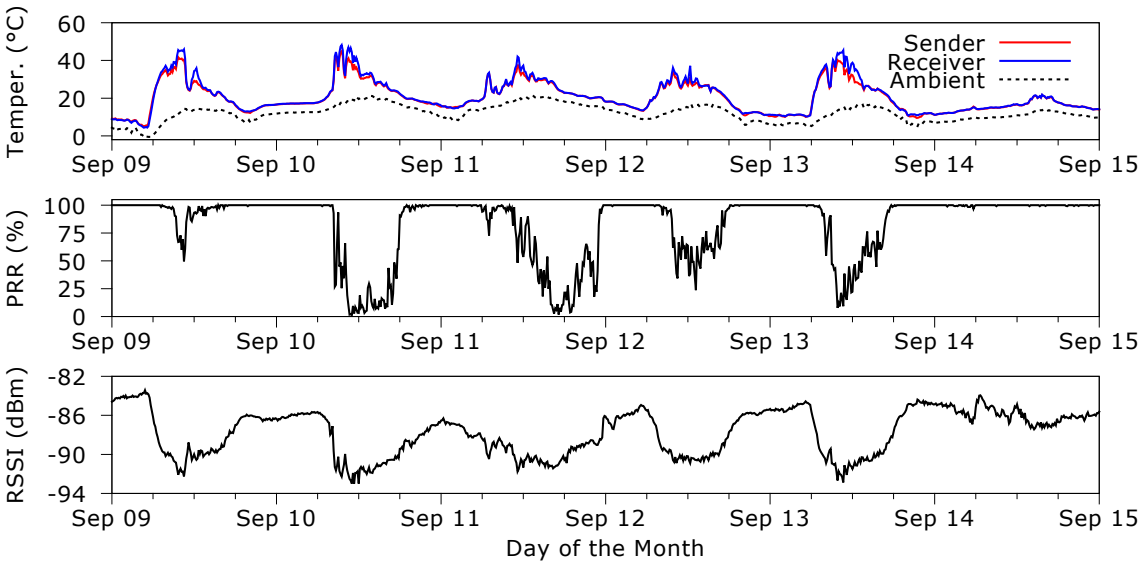


Figure 2.1.: Temperature has a strong impact on link quality in outdoor deployments. Even the normal temperature fluctuations during a day can render a good link useless [48].

aim to close this gap and design tools to make a testbed capable of reproducing real-world temperature profiles.

Augmenting a testbed with the ability to reproduce temperature profiles is not a trivial task. Firstly, we need to recreate in a faithful manner the temperature variations that each node would experience in a real-world deployment over time. Secondly, these temperature profiles must be applied in such a way that no other property of the setup besides temperature is altered. Thirdly, the temperature profiles reproduced in the testbed need to be repeatable in order to allow a systematic quantification of the impact of temperature, and should ideally emulate daily or seasonal changes within a few hours, allowing fast prototyping and experimentation. All these goals should be met while minimizing costs and efforts, so to have a widely applicable solution.

In this chapter we present TempLab, an extension for wireless sensor networks testbeds that allows the on-board temperature of sensor nodes to be varied in a fine-grained and repeatable fashion. We describe testbed components, methods for implementing different temperature profiles, and evaluate TempLab to show that it can accurately reproduce temperature dynamics found in outdoor environments with fine granularity.

The next section motivates the need for a testbed solution to evaluate the impact of temperature on wireless sensor networks. Sect. 2.2 describes the requirements of such a testbed infrastructure. We describe the architecture and implementation of TempLab in Sect. 2.3 and 2.4. We finally investigate TempLab's performance in Sect. 2.5, showing that temperature dynamics found in typical deployments can be accurately reproduced.

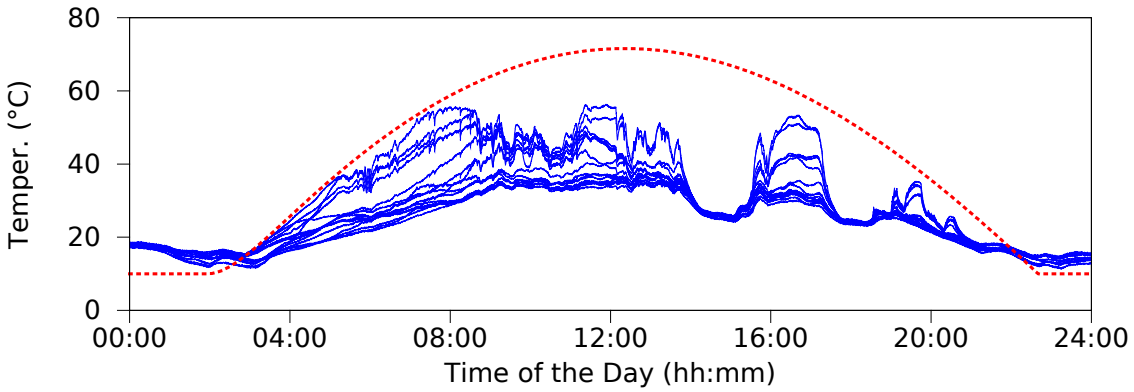


Figure 2.2.: Temperature profiles over the course of a day of 16 nodes deployed in an outdoor setting (blue curves), and maximum temperature profile obtained with the model-based instantiation (red dashed curve).

2.1. Temperature Matters

Temperature affects the operations of the most basic elements in *all* electric and electronic circuits: from resistors and capacitors to clocks and transistors. Due to this impact, assessing the effect of temperature on *individual* devices is usual practice in industry, and most electronic devices are given an operational range. Temperature also matters at the *network* level, but the effect of temperature on *inter-device* operation is far less understood.

A few studies have started to evaluate the effect of temperature on network operations. Bannister et al. [5] showed that temperature has a significant impact on link quality, and Boano et al. [13] validated these claims with a more systematic study. The most powerful case highlighting the importance of temperature at the network level is probably given by Wennerström et al. [48], who report insights from a long-term study showcasing the impact of meteorological conditions on the quality of 802.15.4 links. Fig. 2.1, based on traces recorded during Wennerström's outdoor deployment, shows the on-board temperature of a transmitter and receiver pair, and the packet reception rate of their link: even the normal temperature fluctuations occurring during a day can transform a perfect link (100% PRR) into an almost useless one.

However, besides these initial studies, temperature has not received (at the network level) the same level of attention that it received at the device level, but it definitely should. Temperature introduces a sort of *dynamic heterogeneity* across the network: two nodes with the same parameters, but with different on-board temperatures, will perform differently. It is important to analyse this temperature-based heterogeneity, because even nodes that are physically close can have vastly different temperature profiles.

In Wennerström's deployment [48], indeed, all the nodes are within each-other's transmission range, and experience highly different temperatures. Fig. 2.2 depicts the on-board temperature of sixteen of these nodes over the course of a summer day [48], and Fig. 2.3 depicts the temperature density function for two of them. One node is much "hotter" than the other, and this hot node will have a shorter transmission coverage [5], [13], a larger clock drift [39], whereas

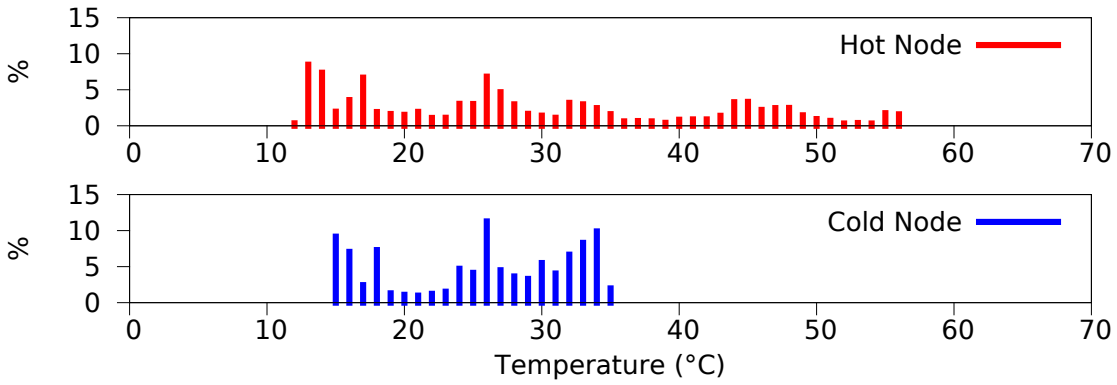


Figure 2.3.: The temperature profile of nodes can be highly different even if nodes are in proximity of each other. This difference can affect the overall performance of the network.

the lifetime of the cold node will be much shorter [34].

How do all these temperature effects, and others that are yet uncovered, affect the operation of network protocols?

To evaluate these effects, we need to provide the sensor network community with a simple, yet accurate, low-cost testbed infrastructure enabling the study of the effects of temperature variations on the network performance in a precise and repeatable fashion.

2.2. Requirements

Such a testbed solution should essentially have the ability to control the on-board temperature of wireless sensor nodes. However, in order to accurately reproduce the temperature dynamics that can be found in typical deployments, it is not simply enough to choose off-the-shelf heating and cooling elements and connect them to the testbed. The choice of the hardware, as well as the design of the infrastructure should meet a number of requirements that we describe below.

Large temperature range Ideally, the testbed would be able to reproduce temperature patterns covering the complete operating range of sensor nodes. For example, in the case of the off-the-shelf TelosB platform, this would imply to heat sensor nodes up to 85°C, but also to cool them down to −45°C. While it is perhaps not necessary to cool all the way down to −45°C, it is important to reach temperatures below 0°C to reproduce the conditions that can be found in a real deployment during the coldest times of the year.

Fine-grained temperature control As shown in Fig. 2.2, the temperature of a node deployed outdoors can continuously vary depending on the presence of sunshine and obstacles (e.g., clouds or buildings). These effects cause *continuum* gradients of temperature, i.e., the jumps of temperature are not sudden and discrete, but smooth. Since our goal is to recreate temperature traces in the most faithful manner, the testbed infrastructure should be able to precisely tune the on-board temperature of a sensor node with a high resolution.

Fast temperature variations In a real deployment, temperature can change quickly due to meteorological effects such as wind, rain, and snow, as well as due to the presence of clouds or sunshine. In the deployment shown in Fig. 2.2, for example, a node that receives the first sun-rays at the beginning of the day increases its temperature as much as 1.98°C/minute. An important requirement for the infrastructure that we want to build is hence the ability of reproducing these variations as fast at they occur in the real-world. This requirement has a strong effect on how accurately temperature dynamics can be reproduced.

Time scaling It is often desirable to compress the time scale of an experiment to save evaluation time (as long as such time compression does not depend on the rate of the temperature change, but only on the absolute temperature values). One may want to time-lapse the recreation of real-world traces and playback, for instance, in a few hours the profile of a full day. This poses stronger requirements on the ability of the testbed to quickly heat up and cool down nodes.

Per-node temperature control As observed in Fig. 2.2, the profile of each node can be highly different. Hence, placing all the nodes into a single chamber would not be realistic because all nodes would follow the same temperature profile. Temperature must be controlled individually on each node.

Unaltered system behaviour The extension of the existing infrastructure should ideally not alter the behaviour of the system in any way, as this may lead to unwanted (and unexpected) system failures. For example, the use of metal casings should be restrained, as RF propagation should be minimally affected. Similarly, the use of I/O ports of a sensor node to control heating or cooling devices has to be avoided if this would affect the operations of the system.

Scalability Although it may not be necessary to augment all nodes of an existing infrastructure with temperature control, it should be ideally possible to extend an entire testbed. Commonly used testbeds such as MoteLab [49], TWIST [27], and NetEye [29] have typically up to 200 nodes, and our testbed solution should be able to scale at these levels.

Low cost All the above requirements have to be satisfied while minimizing the cost of the solution, in order to make it applicable on a large-scale.

In the next section, we present the general architecture of TempLab, our low-cost extension of testbed facilities capable of reproducing real-world temperature profiles with fine granularity, and describe the hardware and software components that we use in our implementation.

2.3. Architecture

In order to study the effects that temperature variations have on the operation of wireless sensor networks and their protocols, the infrastructure needs to be able to reproduce specific temperature profiles on several nodes. This requires (i) temperature profiles to be reproduced, (ii) actuators to control the on-board temperature of each sensor node, and (iii) a controller that uses the actuators to instantiate the desired profiles.

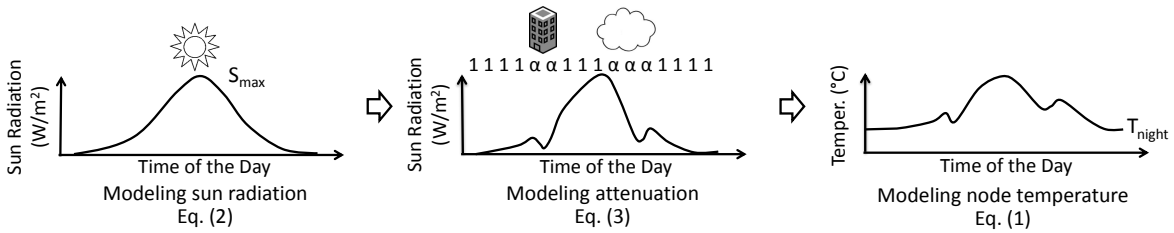


Figure 2.4.: Model-based temperature profile generation.

Temperature Profiles

In order to support a wide range of experimentation techniques, TempLab can generate temperature profiles using three different approaches.

Firstly, one can re-play temperature traces collected in-situ at a given deployment site, such as those in Fig. 2.2. Such *trace-based* temperature profile instantiation can accurately reflect the temperature variations over time with fine granularity if long-term measurements from one or more nodes are available. However, traces are not always at one's disposal, or they may be incomplete: trace-based profiles can be used only if one or more sensor nodes deployed previously actually collected temperature data with a frequency sufficiently high to capture the dynamics of temperature changes.

A second possibility is, therefore, to use a *model-based* temperature profile to have an estimation about the temperature dynamics at a certain location without the need of traces collected in-situ. A model-based approach uses models to estimate the temperature profile of objects using basic environmental information such as the maximum solar radiation and the minimum temperature during a day (that is readily available from satellites and meteorological stations). The temperature model is derived from the work carried out in Task 1.1 [51] and Task 1.2 [15], and is illustrated in Fig. 2.4.

A third possibility is to use TempLab to vary the temperature of sensor nodes using specific *test patterns*. For example, a user may not be interested in recreating a specific profile and needs instead only to verify whether a high temperature variation has an impact on the operation of a given protocol. In this case, TempLab can be fed with on-off patterns (e.g., a series of cold and warm periods) or jig-saw patterns that vary temperature with a specified frequency, allowing a quick debugging of protocols' behaviour.

Actuators

To heat-up and cool-down the on-board temperature of sensor nodes, one or more actuators are required for each node. Actuation can be applied *out-of-band* or *in-band*. Out-of-band means that the sensor node is not involved in the control of its temperature, i.e., additional processing hardware is needed. In-band methods, instead, make use of the sensor node to vary its on-board temperature, e.g., by using its I/O pins to control heating or cooling devices. Although in-band methods have the advantage of avoiding extra-hardware (and reduce testbed costs), they may alter the system behaviour and violate the corresponding requirement.

Therefore, we design TempLab following an *out-of-band* approach based on infra-red heating

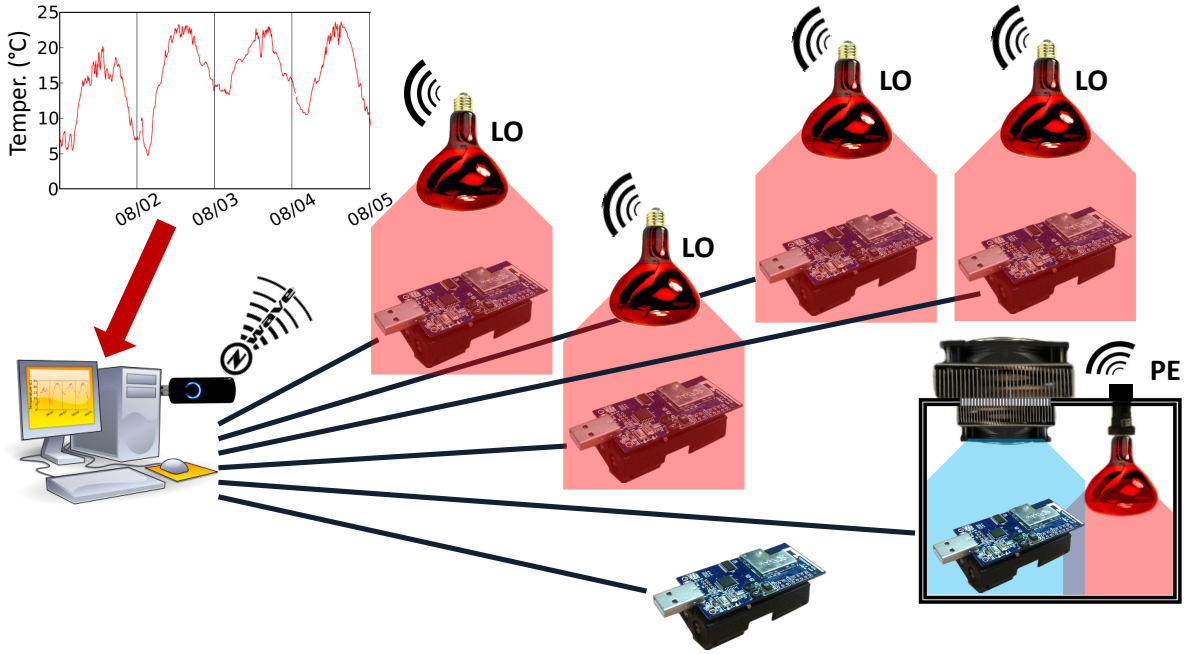


Figure 2.5.: Sketch of TempLab's architecture.

lamps and cooling enclosures that allow to vary the on-board temperature of wireless sensor nodes in the range [-5, +80] °C. TempLab can have two types of nodes with different capabilities as shown in Fig. 2.5: *LO* and *PE* nodes. *LO nodes*, which stands for lamps-only nodes, are heating-only devices that have the capability of warming the sensor nodes between room temperature and their maximum operating range. They are based on IR heating lamps and they do not have any capability to cool-down the nodes below room temperature. *PE nodes*, which stands for Peltier enclosure nodes, are hard temperature-isolating Polystyrene enclosures with an embedded IR heating lamp and an air-to-air Peltier module to heat-up and cool-down the inner temperature of the casing. To control the intensity of the IR lamps and the operations of the Peltier module, we borrow existing home automation solutions and use *wireless dimmers* to vary the intensity of the lamps and *on-off wireless switches* to control the Peltier modules embedded in the enclosure. To make sure that the temperature control system does not interfere with the existing testbed communication, we select home automation solutions working on a ISM frequency band that is different from the one used by the sensor nodes.

This approach can easily scale to large testbeds as PE and LO nodes only need to be plugged into wall power and require no further cabling. Furthermore, home automation solutions such as Z-Wave allow to connect up to 256 wireless dimmers in a multi-hop fashion, and can can in principle scale to large buildings with many devices. If a very large number of nodes need to be supported, it is possible to partition the control network and use several controllers.

Controller

To instantiate a temperature profile and control heat lamps and Peltier modules, TempLab uses different controllers running on a centralized testbed gateway computer.

Open-loop controller The simplest one is an open-loop controller that varies the intensity of the light bulbs in LO and PE nodes according to a pre-computed calibration function[1]. This is possible if the impact of each dimming level on the on-board temperature of a node is known based on a previous calibration. In this case, the open-loop controller can instantiate a given profile without further processing. The key advantage of this approach is hence that no sensors are needed to measure the actual temperature of the motes during the experiment. For an accurate replay of temperature dynamics, however, the surrounding environment as found during calibration would need to remain constant, as the controller would not account for external factors influencing temperature such as open windows or sun shining in the room hosting the testbed.

Closed-loop controller To precisely regenerate trace- or model-based temperature profiles, TempLab uses a closed-loop proportional-integral (PI) controller that tries to minimize the difference between the desired temperature profile and the on-board temperature of the sensor node of interest. The controller should hence receive a periodic feedback with frequency F_U about the on-board temperature of the sensor node in order to minimize the error with respect to the desired temperature profile. The reading of the on-board node temperatures can be carried out either *out-of-band* through the use of an external device or *in-band* using the sensor node itself to measure the temperature and forward it to the controller. As most off-the-shelf wireless sensor nodes carry on-board a temperature sensor, it is very tempting to use an in-band approach to provide up-to-date temperature measurements without adding extra-costs. However, it has to be ensured that system behaviour is not altered. TempLab uses an in-band approach using the USB back-channel to periodically convey temperature readings to the controller. This task is carried out using a low-priority routine executing only when the processor is idle.

During our experiments, we have observed that common USB serial connections used in testbeds for data logging and node programming may be unable to cope with very fast temperature fluctuations, as they result in de-synchronization of the USB sender and receiver. In the presence of such variations, the USB serial port looses synchronization with the mote and the characters forwarded to the USB back-channel become temporarily unreadable, as shown in Fig. 2.6. Since standard nodes do not handle this issue autonomously, TempLab either re-initializes the USB port or piggybacks the temperature readings onto regular data packets. In this way, other nodes that do not suffer from this issue can report the temperature to the controller over the USB back-channel.

[1] For PE nodes, one can vary the intensity of the heat lamps while the Peltier module is constantly active. As we show in Sect. 2.5, the IR lamp can change the temperature much quicker than the Peltier module, and a constantly active Peltier module does not slow down the heating from the IR lamp significantly.

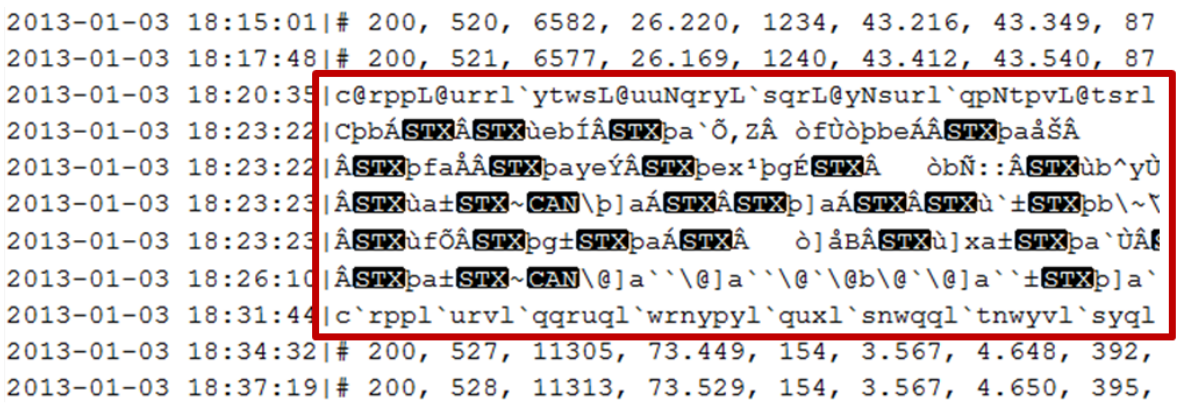


Figure 2.6.: Temporarily unreadable USB serial output in the presence of sudden thermal variations.

2.4. Implementation

We now describe the hardware and software components that we used to extend our local university testbed based on Maxfor MTM-CM5000MSP nodes (TelosB replicas).

2.4.1. Hardware

In our implementation, we use Philips E27 infra-red 100W light bulbs that can be remotely dimmed using the Z-Wave wireless home automation standard. The latter operates on the 868 MHz ISM band, and hence does not interfere with the communications between the wireless sensor nodes (that use the 2.4 GHz ISM band)[2]. To vary the intensity of the light bulbs, we used Vesternet EVR_AD1422 Z-Wave Everspring wireless dimmers, which provide 100 dimming levels.

LO nodes are only controllable using dimmers. PE nodes have the capability of going below room temperature thanks to enclosures made of hard Polystyrene foam embedding, in addition to the IR heating bulb, an ATA-050-24 Peltier air-to-air assembly module by Custom Thermoelectric. The latter allows on-board temperatures of -5°C when operated at room temperature, and can be controlled through Vesternet EVR_AN1572 Z-Wave Everspring on-off wireless switches. The Polystyrene hard foam isolating box has a minimal impact to the radio propagation of sensor nodes and supports temperatures up to +85°C. The overall hardware cost is 65 € for each LO node, and 293 € for a PE node.

2.4.2. Software

Actuators We control the Z-Wave network with a C++ program that uses the Open Z-Wave stack to vary the intensity of dimmers and duty cycle the Peltier modules. Commands to the

[2]We have also implemented a TempLab version that uses the LightwaveRF standard operating on the 433 MHz ISM band, in case the sensor nodes in the testbed operate on the 868 MHz ISM band. In the remainder of the deliverable we refer to the Z-Wave implementation.

control network are sent through the Aeon Labs Series 2 USB Controller deployed within the testbed facility.

Controller Each node runs Contiki, and contains a low-priority process that periodically measures temperature using the on-board SHT11 sensor, and communicates the readings over the USB back-channel. This can be also easily implemented in TinyOS or other operating systems, since it needs only basic building blocks such as reading and outputting temperature. To select the sampling frequency F_U, i.e., how often should the controller receive feedback about the on-board node temperature, we use the fastest temperature variation observed in the outdoor deployment shown in Fig. 2.2, and compare it to the accuracy of the on-board temperature sensors. In our case, the nodes carry SHT11 sensors that have an accuracy of 0.4°C. According to the profiles shown in Fig. 2.2, such a variation can be reached within 12 seconds.

The PI controller is implemented as a standalone multi-threaded C++ application executing on the testbed gateway that receives as input a file with two columns: the first one contains the time of the day, the second one describes the on-board temperature that the node should have at that time. The controller is agnostic to the type of trace (whether derived empirically or from a model): as long as the file adheres to the two column format, it will (try to) recreate such temperatures based on this information and the feedback signals from the motes. In case the user chooses to time-lapse the experiment, the controller skips rows accordingly, e.g., for a 2x speed, the controller skips every other line. We found experimentally that an optimal configuration of the controller is P=2 and I=0.01 to achieve fast and self-stabilizing control.

The controller allows users to manually assign the available traces to the temperature-controlled nodes in the network. If a non-implementable mapping is created, e.g., when mapping a trace containing negative temperatures to a LO node, the controller will signal an error.

2.5. Evaluation

In this section, we carry out an experimental evaluation of the capabilities of our TempLab implementation. First, we investigate the performance of TempLab in terms of implementable temperature profile dynamics and highlight the limitations on how fast nodes can be heated or cooled. Thereafter, we show that temperature dynamics found in typical deployments can be accurately reproduced despite the low-cost infrastructure, even when compressing the time scale of an experiment to save evaluation time.

2.5.1. Heating and Cooling Limits

To verify how fast LO and PE nodes can be heated and cooled, we carry out an experiment in which we let the closed-loop PI controller heat the nodes to 80°C. The initial temperature is room temperature for LO nodes and 0°C for PE nodes, respectively. After reaching a stable temperature, the controller cools the nodes down to their original value.

LO nodes Fig. 2.7 shows that LO nodes can heat from room temperature (26°C) to 80°C in less than 5 minutes, with an average heating slope of 11.3°C/minute. As LO nodes do not have cooling capabilities, their cooling is rather slow: they need only 7 minutes to decrease from 80°C to 35°C, but they require the same time to decrease from 35°C to 30°C, and 20 more minutes to get back to 26°C.

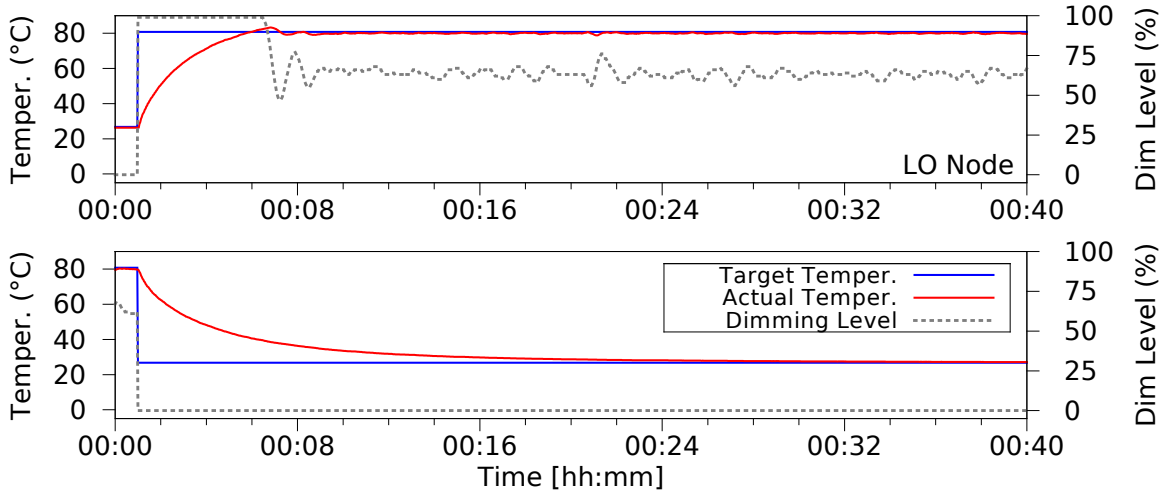


Figure 2.7.: Limits in the speed of heating and cooling for LO nodes.

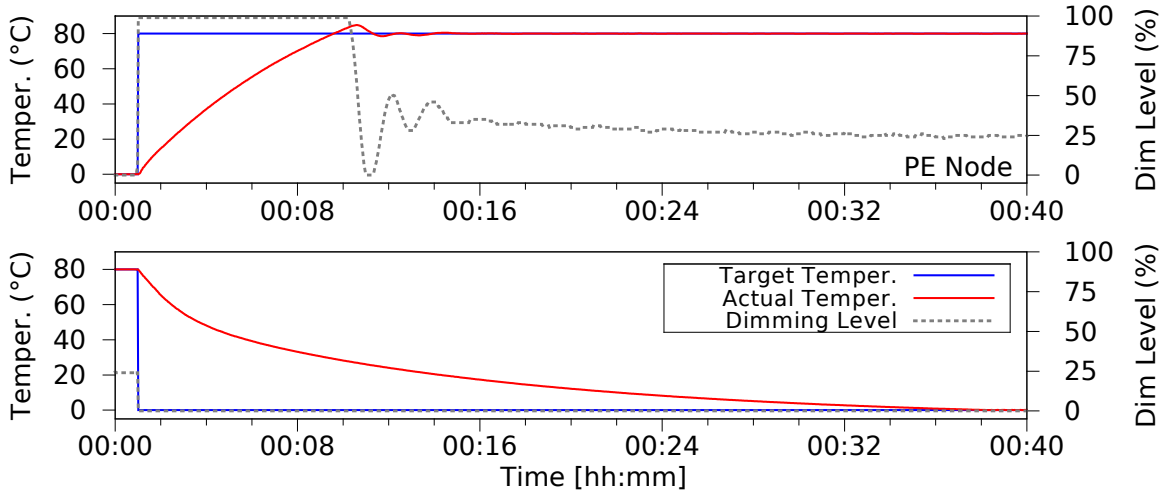


Figure 2.8.: Limits in the speed of heating and cooling for PE nodes.

PE nodes Fig. 2.8 shows that PE nodes can heat from 0°C to 80°C in less than 9 minutes, with an average heating slope of 9.3°C/minute. PE nodes are obviously much more efficient in cooling than LO nodes: they need only 6 minutes to decrease from 80°C to 35°C, and 10 minutes to decrease to ambient temperature (26°C). Overall, they can vary the temperature from 80°C to 0°C in less than 35 minutes.

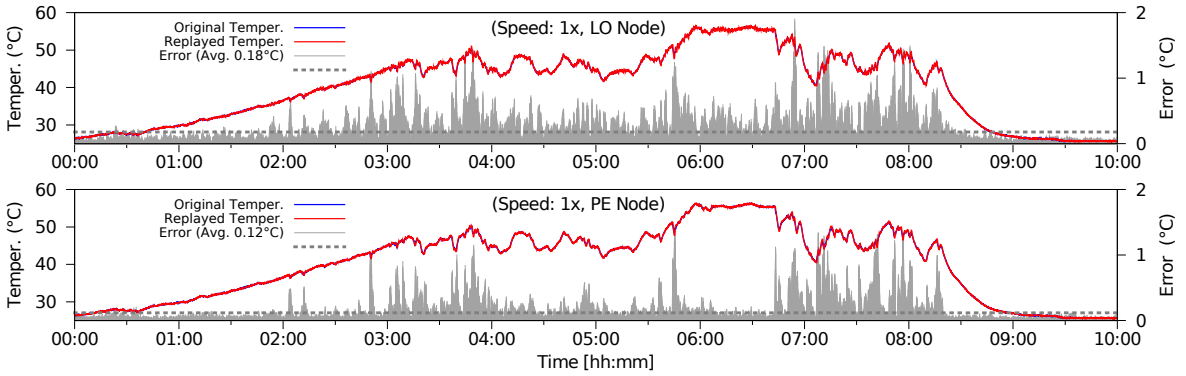


Figure 2.9.: Accuracy of LO and PE nodes in replaying a real-world trace captured during summer.

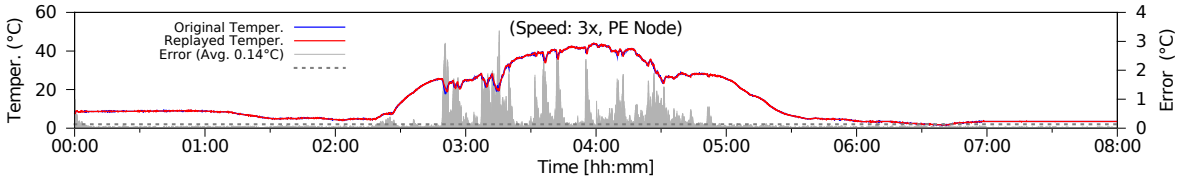


Figure 2.10.: Accuracy of PE nodes in replaying a real-world trace captured during winter.

2.5.2. Regeneration of Traces

We now evaluate TempLab's ability of reproducing a given temperature profile. We compute the accuracy of TempLab by computing how close the instantiated temperature profile P_I follows the given profile to be reproduced P_G. The overall accuracy Q_n of the reproduced temperature profile at node n can be expressed as:

$$Q_n = \frac{1}{T}\int_0^T |\mathrm{P_I(t) - P(t)}|\,\mathrm{d}t \tag{2.1}$$

where T is the duration of the experiment. Besides the requirement to follow a temperature profile over time, it is also important to ensure that the rate of temperature changes is reflected accurately. At no point in time the instantiated temperature curve at a node n should deviate too much from the given temperature profile. The maximum deviation q_n can be expressed as:

$$q_n = \max_t |P_I(t) - P(t)| \tag{2.2}$$

The smaller the value of Q_n, the better the instantiation of the temperature profile, whereas the smaller q_n, the better the dynamics of the temperature change are reflected.

We take as a reference for our evaluation two temperature traces collected in an outdoor deployment in Sweden [48]: one taken during summer (August), and a "colder" one taken in the end of October, when temperature approaches 0°C.

Summer trace Fig. 2.9 shows that both LO and PE nodes can instantiate the desired temperature profile on the sensor nodes with very high accuracy. The average error Q_n equals

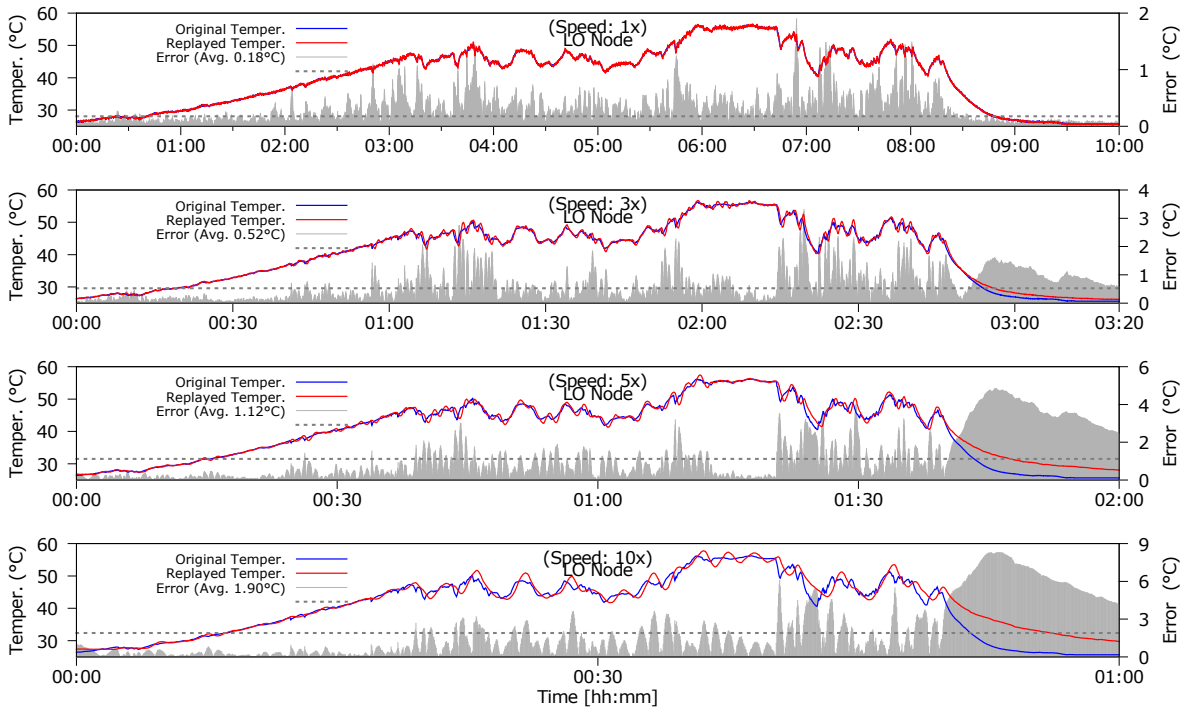


Figure 2.11.: Accuracy of LO nodes when compressing the time-scale of the experiment. The reproduced trace was taken during summer.

0.18°C and 0.12°C, whereas q_n is 1.90°C and 1.43°C for LO and PE nodes, respectively. This is a remarkable accuracy, and shows that despite the use of low-cost components (LO nodes), TempLab can still reproduce with high accuracy real-world temperature profiles above room temperature.

Winter trace During winter time, the sun can quickly raise the temperature in the package hosting the sensor nodes. We replay a trace captured during October 2012 [48], in which the on-board temperature of a node has a significant variation from 45°C during daytime to 0°C in the evening, and see how accurately PE nodes can instantiate this temperature profile on sensor nodes. Fig. 2.10 shows the results: the average error Q_n equals 0.14°C, whereas $q_n = 3.36$°C.

Accuracy of time-lapsed traces The accuracy of the replay shown in Fig. 2.10 is even more remarkable if we consider that we have compressed the original 24-hour trace into 8 hours playback time, i.e., we used a compression factor of 3. We now show the accuracy of LO and PE nodes in the regeneration of traces in which the time has been compressed even further.
Summer trace. Fig. 2.11 shows the accuracy of the regeneration of a summer trace using LO nodes: when instantiating the same trace used in Fig. 2.9, LO nodes show evident limits due to the lack of cooling capabilities. Compared to the error of 0.18°C when regenerating at normal speed, the average error Q_n raises to 1.12°C when the time is compressed by a factor of 5, whereas Q_n is 0.52°C and 1.90°C when replaying a trace compressed with factor 3 and 10,

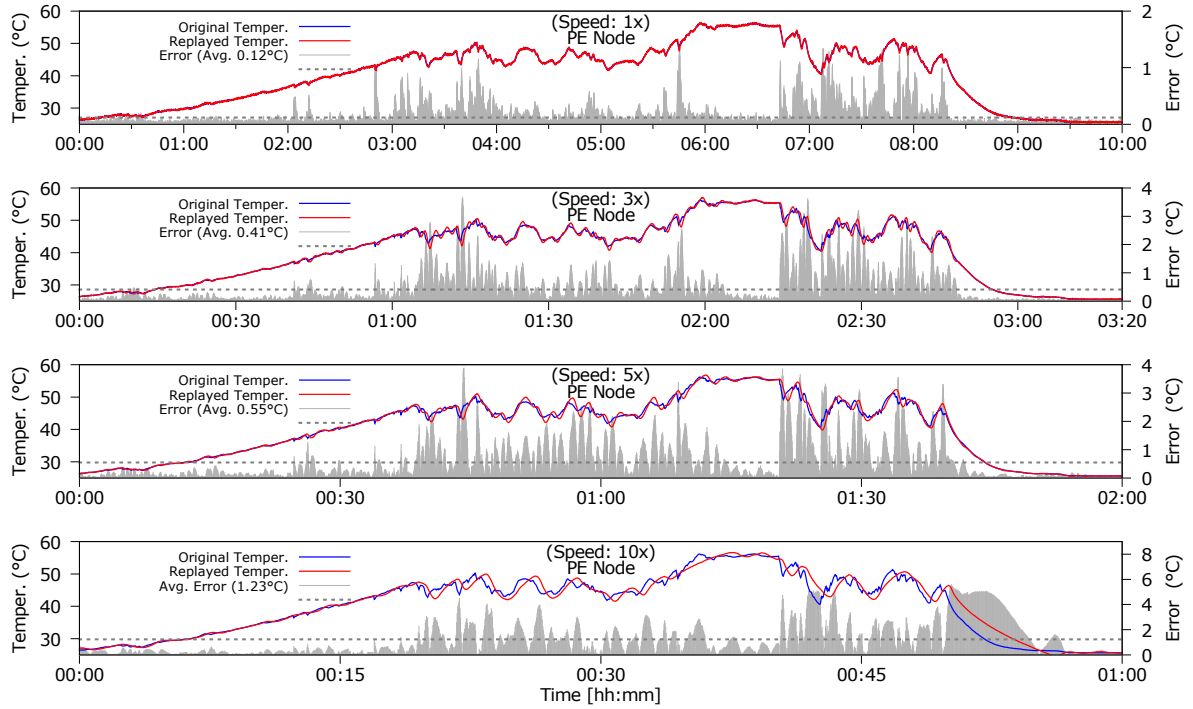


Figure 2.12.: Accuracy of PE nodes when compressing the time-scale of the experiment. The reproduced trace was taken during summer.

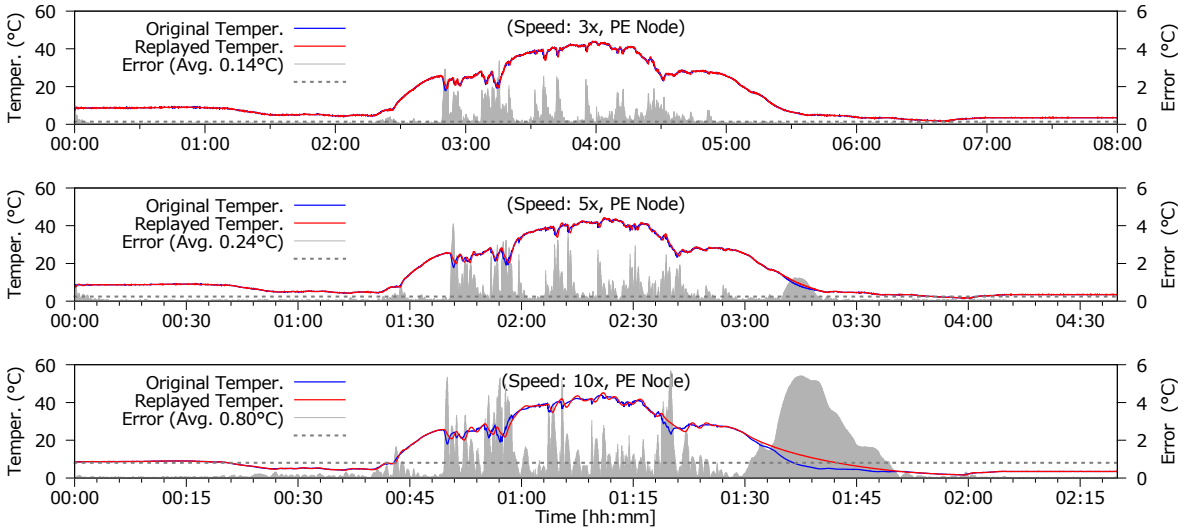


Figure 2.13.: Accuracy of PE nodes when compressing the time-scale of the experiment. The reproduced trace was taken during winter.

respectively. PE nodes, instead, can replay the summer trace with higher accuracy: a trace 3 times faster than the original speed is replayed with an error of $Q_n = 0.41°C$, which corresponds to an error reduction of 25% compared to the LO nodes, as shown in Fig. 2.12. A clear advantage of PE nodes is shown when replaying a trace 5 times faster than the original speed. In this case $Q_n = 0.55°C$ and the error is hence halved compared to the LO nodes ($q_n = 3.84°C$). When compressing time by a factor of 10, however, we can start to observe that the Peltier modules reach their limit, and cannot properly cool down in only 4 minutes what in reality takes 45 minutes. Nevertheless, Q_n is only 1.23°C, and $q_n = 5.57°C$.

Winter trace. Fig. 2.13 shows the accuracy of the regeneration of a winter trace using PE nodes. The average error is $Q_n = 0.14°C$ and $Q_n = 0.24°C$ when replaying the trace 3 and 5 times faster than the original speed, respectively. Also in this case, when compressing time by a factor of 10, we start to observe that the Peltier modules cannot cope with quick temperature variations, as they cannot properly cool down in only 15 minutes what in reality takes more than 2 hours. Although Q_n is only 0.80°C, the maximum deviation q_n can be as high as 5.41°C.

3. JamLab: a Testbed to Study the Impact of Radio Interference on Wireless Sensor Networks

The reliability and robustness of sensornet communications are strongly affected by radio interference. As an increasing number of standardized communication technologies operate in ISM bands, the congestion in the radio spectrum is inflating, and the quality of communications decreases. In safety-critical sensornet applications such as industrial automation and health care, in which the reliability and stability of communications are vital, radio interference represents a major challenge, as it leads to packet loss, high latencies, and reduced energy-efficiency due to retransmissions. This issue is especially serious in the 2.4 GHz ISM band, as wireless sensor networks that operate at such frequencies must compete with the ongoing communications of WLAN, Bluetooth, and other IEEE 802.15.4 devices. Furthermore, sensornet communications in these frequencies can also be affected by several domestic appliances that are source of electromagnetic noise, such as microwave ovens, video-capture devices or baby monitors. This high number of different wireless devices sharing the same frequencies and space raises the need for coexistence and interference mitigation techniques in 802.15.4-based sensor networks, as highlighted by previous studies [35, 40].

In particular, there is a strong need for understanding the performance of existing sensornet protocols under interference, as well as designing novel protocols that can deliver high and stable performance despite changing interference patterns. This, however, requires a proper testbed infrastructure where realistic interference patterns can be easily created in a precise and repeatable way. Unfortunately, existing sensornet testbeds lack such capabilities for interference generation, or they are limited to static WiFi access points randomly placed in the testbed [29], which does not enable the creation of a wide range of interference patterns in a repeatable way. Furthermore, the use of expensive equipment such as VSG-based EMI generators to generate realistic interference patterns as done in [41] is also not optimal. Upgrading existing testbeds with additional heterogeneous devices in order to introduce interference sources is a costly, inflexible, labor-intensive, and placement-dependent operation.

We therefore propose to augment existing sensornet testbeds with JamLab, a low-cost infrastructure for the creation of realistic and repeatable interference patterns. Such an infrastructure supports the recording and playback of interference traces in sensornet testbeds, as well as the customizable generation of typical interference patterns resulting from WiFi, Bluetooth, microwave ovens, or any other device operating in the frequency of interest.

The next section describes the general architecture of JamLab. We then detail how to use common sensor motes to accurately measure interference (Sect. 3.2) and accurately replay it (Sect. 3.3). We then discuss how to configure the testbed in Sect. 3.4 and evaluate JamLab's accuracy in Sect. 3.5.

3.1. JamLab Overview

JamLab is a low-cost extension for existing wireless sensor network testbeds that provides a way to generate *realistic* and *repeatable* interference patterns. The key idea behind JamLab is to use off-the-shelf motes to record and playback interference patterns instead of bringing WiFi access points, microwave ovens, or other equipment to the testbed. The latter approach is not only costly and hard to reproduce exactly by other researchers, but it is even difficult to exactly reproduce a given interference pattern with the same appliance. For example, the sequence and timing of the WiFi frames generated by a file download may differ between repeated trials due to TCP adaptation mechanisms (e.g., timeouts, window sizes). Furthermore, every device used to generate interference in the testbed needs to be programmed remotely. Programming several heterogeneous devices such as WiFi access points or microwave ovens would create a significant overhead, whereas using JamLab the installation overhead is minimal.

Indeed, with JamLab, either a fraction of the existing nodes in a testbed are used to record and playback interference patterns, or a few additional motes are placed in the testbed area. We call those motes used for interference generation *HandyMotes*. The HandyMotes support two modes of operation: *emulation*, where a simplified model is used to generate interference patterns that resemble those generated by a specific appliance (such as a WiFi device or a microwave oven); and *regeneration*, where each HandyMote autonomously samples the actual interference, compresses and stores it locally, and regenerates the recorded patterns later. The latter mode is especially useful to record realistic interference patterns in a crowded shopping center or on a lively street by placing a few HandyMotes to record interference, and bringing them to the testbed to playback the recorded traces there.

One fundamental challenge results from the fact that the maximum RF output power of motes (0 dBm) is typically much smaller than the RF output of other typical interference sources (25 and 60 dBm for WiFi and microwave ovens, respectively). Therefore, a WiFi transmitter or a microwave oven may disturb sensornet communications over much larger distances than a HandyMote can. We address this issue by subdividing the testbed area into cells as depicted in Fig. 3.1, such that a HandyMote placed at the center of the cell can interfere with all testbed motes contained in the cell, but the interference with motes outsides of the cell is minimized. This requires a careful placement or selection of HandyMotes and control of their RF output power. We investigate this issue and propose a procedure for HandyMote placement and power control in Sect. 3.4. Note that there is a trade-off between the realism of the generated interference patterns and the number of HandyMotes: the more cells, the more accurate is the spatial distribution of interference, but the more HandyMotes are required.

Another challenge is that many interference sources emit wideband signals, i.e., they interfere with many 802.15.4 communication channels at the same time. In contrast, a mote can only transmit on a single channel at a time. Fortunately, most existing sensornet protocols use only a single channel. However, there is a trend to use multiple 802.15.4 channels at different nodes to increase robustness and bandwidth. Our approach to deal with this issue is to place multiple HandyMotes in each cell, each one interfering on one 802.15.4 channel. The use of Software Defined Radio (SDR) techniques using USRP devices would provide more accurate jamming signals on a wider bandwidth, but their high cost represents a sizeable limitation. To synchronize the generation of interference patterns within the HandyMotes in one cell and across cells, we need time synchronization, and we propose to use the testbed infrastructure

Figure 3.1.: Testbed augmented with JamLab. Nodes 6, 9, and 23 are selected as HandyMotes, and take care of interference (re)generation in their cell.

(i.e., wired back-channels) to send synchronization signals to the HandyMotes.

Due to the constrained resources of a mote, also the accurate recording and playback of interference represent a challenge. To capture short interference patterns such as those generated by WiFi beacons, we need high sampling rates with low jitter, which requires data compression due to the limited amount of available memory. Our solutions to these problems are described in Sect. 3.3.1. The accurate measurement of the interfering signal strength turned out to be a challenge in itself due to the gain control in the radio, as detailed in Sect. 3.2.

For the playback of recorded interference traces, normal packet transmissions are not appropriate, as this would offer only limited control over the exact timing of the transmitted signals. Therefore, we use special test modes of 802.15.4 radios to generate modulated or unmodulated carrier signals as detailed in Sect. 3.3.2.

JamLab has been designed specifically for the Texas Instruments CC2420 radio [44], and tested on several sensor motes such as Maxfor MTM-CM5000MSP, Crossbow TelosB, and Sentilla JCreate, but the framework can be applied to any sensornet platform. Based on the analysis of the datasheets, the Handymotes should be easily ported to similar radios such as the Ember EM2420 transceiver, and to newer radios such as the CC2520. We develop the

HandyMotes based on Contiki, a lightweight and flexible operating system for tiny networked sensors [22].

3.2. Measuring Interference Accurately Using Motes

Measuring interference accurately on a mote is a key functionality, both for recording and later playback of interference, as well as for acquiring a deep understanding of common interference sources such as WiFi or Bluetooth. We describe in this section the techniques we used in order to let a common sensor mote measure the interference accurately at a sufficiently high sampling rate.

3.2.1. Measuring at High Sampling Rates

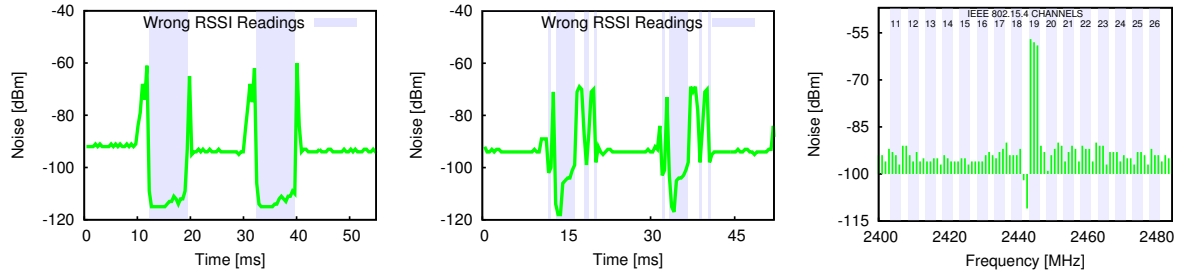
Link quality indicators such as RSSI [42] and LQI [9] provide an indication of the signal strength and quality, but only upon the reception of a packet [4]. The only feasible way to quantify the amount interference is hence the continuous measurement of the RSSI noise floor, i.e., the RSSI in absence of packet transmissions.

We improve existing Contiki tools [18] and develop an application that scans a single predefined IEEE 802.15.4 channel at its middle frequency with a very high sampling rate, and returns the RSSI noise floor readings over time.

A first requirement for this scanner is to achieve a *high sampling rate*, given that we need to detect short transmissions periods. After boosting the CPU speed, optimizing the SPI operations, as well as buffering and compressing the RSSI noise floor readings using Run-Length Encoding (RLE), we reached a maximum sampling rate of approximately 60.5 kHz when sampling a single channel. This sampling rate is not sufficient high to capture all WiFi transmissions, as the maximum speed of 802.11b/g/n standards is 11, 54, and 150 Mbit/s, respectively. The minimum size of a WiFi packet is 38 bytes (ACK and CTS frames), which would make a resolution of 60 kHz sufficient to detect all 802.11b frames, but not all 802.11g/n frames. However, as most WiFi frames are data frames and typically contain higher layer headers, one can sample at 60 kHz frames with TCP/IP headers having a payload size higher than 27 and 227 bytes for 802.11g/n, respectively.

Another requirement is to *accurately* measure the strength of the ongoing interference in the radio spectrum by means of precise RSSI noise floor readings. The CC2420 radio specifies an accuracy of ± 6 dBm, and a linearity of ± 3 dB in the dynamic range $[-100, 0]$ dBm. Such accuracy and linearity has so far been acknowledged by the research community as enough to carry out operations such as Clear Channel Assessment (CCA) and low-power channel sampling for activity recognition [36].

However, our experiments show that the RSSI noise floor readings captured at high sampling rate suffer of a systematic problem in three specific scenarios, namely: (i) when a narrow unmodulated carrier is transmitted, (ii) when microwave ovens are switched on, and (iii) in the presence of Bluetooth transmissions. In these scenarios, the CC2420 radio often returns RSSI values that are significantly below the supported range and the sensitivity threshold, e.g., -110 or -115 dBm. Fig. 3.2 reports examples of such wrong readings, which represent an important problem, since they also impact the correct functioning of CCA in the presence of narrow-band signals, as shown in Fig. 3.2(c). Our investigation also shows that the same problems applies



(a) Active Microwave Oven (b) Bluetooth Transmission (c) Unmodulated Carrier

Figure 3.2.: Examples of wrong RSSI readings: several values are significantly below the sensitivity threshold of -100 dBm due to receiver saturation. This error is caused by an incorrect operation of the AGC loop in presence of narrow-band signals.

to other sensornet platforms employing similar versions of the chip, such as the Ember EM2420 transceiver. We experimentally identified that the problem is due to the saturation of the Intermediate Frequency (IF) amplifier chain: we have observed that maximum gain is used in the Variable Gain Amplifier (VGA) when the incorrect RSSI readings occur.

3.2.2. Avoiding Saturation in RSSI Readings

The reason of this saturation problem can be found in the radio demodulation chain. The CC2420 chip implements part of the IF filtering in analog domain and further filtering is later performed in digital domain. It employs an Automatic Gain Control (AGC) loop to maintain the signal amplitude close to a certain target value that guarantees the correct operation of the Analog-to-Digital Converter (ADC). More specifically, the signal is maintained within the ADC dynamic range, despite large variations in the input signal from the antenna. For this purpose, the AGC loop uses a digital sample of the final IF signal amplitude and adjusts the gain of the VGA stage accordingly.

If a narrowband signal is present near the cut-off frequency of the combined IF chain, the resulting sampled signal amplitude may be remarkably lower than the partially unfiltered one at the ADC, as a consequence of the digital filtering. Since the AGC uses the final value to set the gain of the amplifier chain, there is no guarantee that the ADC is not saturating. In the event of ADC saturation, the receiver is no longer linear and the RSSI values are incorrect.

To linearise the radio response for an arbitrary noise signal and hence avoid wrong RSSI readings, we activate the peak detectors in-between the amplifier stages so that their output is used by the AGC algorithm to compute the required gain. The latter is attained with VGA stages and the system switches in and out fixed gain stages as needed. In the CC2420, the peak detectors are controlled by the *AGCTST1* register, and can be configured as follows:

```
unsigned temp;
CC2420_READ_REG(CC2420_AGCTST1, temp);
CC2420_WRITE_REG(CC2420_AGCTST1,
(temp + (1 << 8) + (1 << 13)));
```

The register also includes flag bits to activate peak detectors among fixed gain stages in the IF chain and at the ADC itself [44].

3.3. (Re)Generating Interference

With the techniques to accurately measure interference introduced in the previous section, we can now proceed to record and replay those patterns. We describe first how to compress and store traces on motes and then how to playback those recordings.

3.3.1. Recording Interference Traces

When used in *regeneration* mode, HandyMote records interference traces that are later played back accordingly. Those traces can be either stored on the mote in RAM or Flash memory, or – if the HandyMote is connected to a testbed during recording – can be streamed over a wired backchannel to a base station. In any case, the data rate of 480 kbps generated by sampling RSSI with a resolution of 8 bits to hold values between 0 and -100 dBm at 60 kHz is too high to store it directly in memory or to stream it over the back-channel. The very efficient Coffee Flash file system supports a peak write bandwidth of only 376 kbps [45], the MSP430 UART supports a maximum data rate of 460 kbps for writing to the USB back-channel, and the limited 4 kB RAM of the MSP430 could just record a trace of less than 70 milliseconds duration.

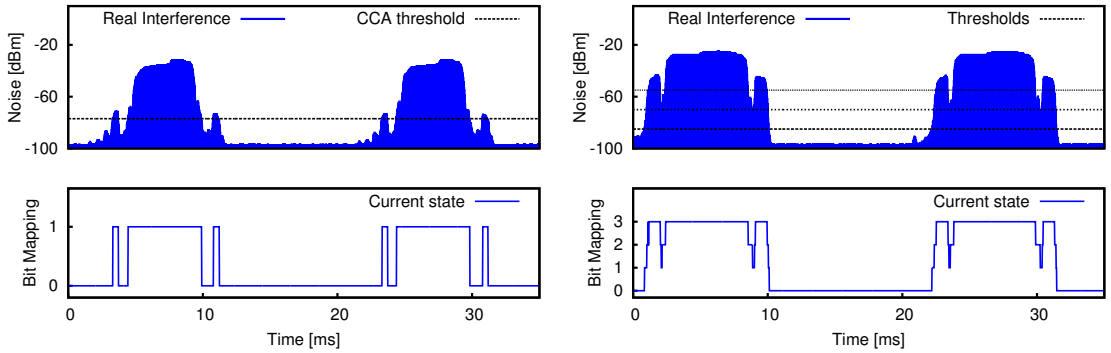
While we need a high compression ratio, the compression method has to be efficient enough to allow sampling of RSSI at 60 kHz. Therefore, we use a simple Run-Length Encoding strategy and a quantization of the samples to a few bits per sample. We store a stream of pairs (v, o), where v is a sample and o is the number of consecutive occurrences of this sample. This method is very effective, as RSSI values typically change slowly over time.

The quantization is justified by the fact that the CC2420 only supports 11 distinct output power levels in the range [-55,0] dBm by setting the PA_POWER register to the values we derived and listed in Table 3.3.1. To obtain the highest possible output resolution, four bits per sample with an appropriate non-linear quantization are hence sufficient. For example, for two-bit resolution one can use thresholds -55, -70, and -80 dBm (or register values 31, 7, and 3) with a spacing of 15 and 10 dBm, respectively, for quantizing the RSSI range into four regions.

PA_POW.	dBm	PA_POW.	dBm	PA_POW.	dBm
31	0	15	-7	2	-45
27	-1	11	-10	1	-50
23	-3	7	-15	0	-55
19	-5	3	-25	-	-

Table 3.1.: Discrete output power levels of the CC2420 radio.

Fig. 3.3(b) shows how original RSSI readings (top) are mapped into 2 bits (bottom): the two-bit quantization of a 35 ms interference recording reduces the amount of data from 2076 Bytes to 84 bytes – a compression ratio of $\frac{1}{25}$. A single bit per sample is enough to support binary



(a) 1-bit precision (b) 2-bit precision

Figure 3.3.: Encoding techniques to save memory resources.

interference regeneration. This corresponds to the outcome of a continuous CCA operation, in which the outcome busy/idle channel is mapped to a binary number [47].

Fig. 3.3(a) shows the outcome of a one-bit quantization of 35 ms of interference. The amount of data is reduced from 2076 Bytes to 20 Bytes – a compression ratio of less than $\frac{1}{100}$. This reduces the raw data rate of 480 kbps to less than 5 kbps (depending of course on the values of the raw samples), a data rate that can be handled by Flash and USB, and allowing us to store several seconds of recording in RAM. In our current implementation, we store traces in RAM.

Recording interference traces is energy demanding, as both CPU and radio need to be constantly active while scanning the radio medium. Using software-based on-line energy estimation [21], we obtain an average power consumption of 65.4 mW for Tmote Sky motes, which allows for a lifetime up to 4 days when powered using primary AA batteries.

3.3.2. Generating Interference Patterns

Recent works have shown that the CC2420 test modes can be used to generate controllable and repeatable interference [7, 8] by transmitting a modulated or unmodulated carrier signal that is stable over time. This approach is superior to common jamming techniques based on packet transmissions, as the emitted carrier signal is independent from packet sizes and inter-packet times.

In order to generate an interference pattern, the interferer has to be enabled and disabled and its output power has to be set according to the compressed recorded trace in regeneration mode or according to the output of models in emulation mode. When enabling the transmitter using the STXON command, the radio oscillator first has to stabilize before a transmission is possible, resulting in a latency of 192μs or a maximum playback frequency of only 5 kHz. Therefore, we leave the transmitter on and just change the output power level to 0 (or -55 dBm) instead of disabling the transmitter. At level 0 the RF output power is so small that even a receiver at a distance of only few centimetres can hardly notice the signal. The advantage of this approach is that the latency for changing the output power is dominated by the SPI access time. We optimized the SPI driver in Contiki, resulting in a latency of only few microseconds –

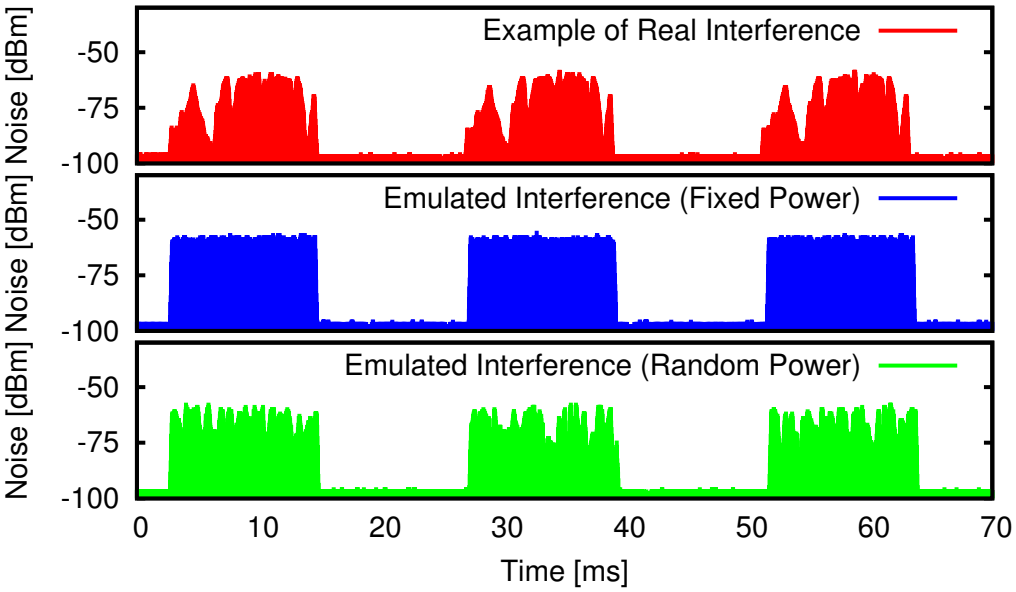


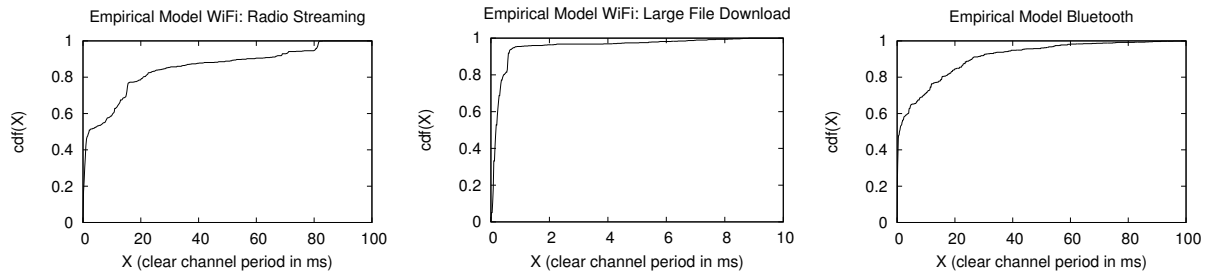
Figure 3.4.: Emulation of microwave oven interference (top) with fixed (middle) and random power (bottom).

allowing us to to playback at the same frequency of 60 kHz that was also used during recording.

Besides the sampling and playback rate, also the jitter during playback of the individual samples needs to be minimized in order to ensure an accurate reconstruction. At 60 kHz, the playback time between two consecutive samples is just 17μs, hence the duration of the execution of a sequence of micro-controller instructions is no longer negligible. In particular, different execution paths in the program to uncompress samples in regeneration mode lead to different execution times and jitter. Therefore, we add NOP instructions to make all execution paths equally long.

3.3.3. Emulation of Interference Through Models

We now describe how we can use an HandyMote to emulate three major sources of external interference on the 2.4 GHz ISM band: WiFi and Bluetooth devices, as well as microwave ovens. We present models that capture the temporal characteristics of these interference sources. A key requirement is the simplicity and efficiency of models, as they need to be executed in real-time on the HandyMotes to generate interference. We are not concerned about the intensity of the generated interference, since when running a HandyMote in *Emulation Mode*, we can decide to adjust the output power of the CC2420 according to different schemes. For example, the output power can be kept fixed or chosen randomly, as shown in Fig. 3.4 (emulation of the interference generated by a Whirlpool M440 microwave oven).



(a) (b) (c)

Figure 3.5.: Empirical Models for WiFi and Bluetooth.

WiFi Emulation

Modeling Wi-Fi traffic is challenging, as it depends on several factors such as the number of active users, their activities, the protocols they use (UDP or TCP), the traffic conditions in the backbone, etc. Under some reasonable assumptions, several theoretical studies have analyzed the performance of 802.11 [6, 26, 32, 38]. However, based on the empirical data we collected, we observed that the models for saturated sources provide a better approximation than the models for non-saturated sources (saturated sources always have data to send). Hence, in order to re-create a realistic representation of interference patterns, we implement an analytical model for saturated traffic sources, and for non-saturated traffic we derive models from empirical data.

Non-Saturated Traffic: Empirical Model. The empirical model for non-saturated traffic is obtained in the following way. Let us denote a random variable X as the *clear* time between two consecutive *busy* times. We obtain the probability mass function $p(x) = P_r\{X = x\}$ from the empirical sampling of the channel, where x is the time in number of slots (each slot is 1 ms). The length of the *busy* times is represented by the transmission delay of packets, which is a rather deterministic variable (for a fixed packet size). Following the methodology described on the previous paragraph, we obtained the $p(x)$ for the scenarios presented on Table 3.2. Figures 3.5(a) and 3.5(b) show the probability mass function $p(x)$ for two WiFi scenarios: an audio-stream application and the download of a large file.

Scenario	Users	Scenario	Users
Radio Str.	1	Video Str.	1
File Transfer	1	File + Radio	1

Table 3.2.: Scenarios.

Saturated Traffic: Analytical Model. There exist several analytical models for the Distributed Coordination Function (DCF) mode of 802.11. Among them, the model proposed by Bianchi [6] has been one the most influential. Bianchi modeled the DCF mode of 802.11 as a discrete Markov process, where the back-off and retransmission mechanisms are represented as discrete states. Based on this model, Garetto and Chiasserini [26] developed a simpler Markov process by merging back-off states. For details, we refer the reader to their paper [26]. In our work, we use Garetto and Chiasserini's model to emulate WiFi interference for saturated sources: whenever there are transmissions of frames in the model, the HandyMote activates

the carrier.

Bluetooth Emulation

IEEE 802.15.1, better known as Bluetooth, specifies 79 channels, spaced 1 MHz, in the unlicensed 2.4 GHz ISM band. Bluetooth stack implementations apply an Adaptive Frequency Hopping (AFH) mechanism to combat interference, which does not permit to anticipate the frequency at which the interference will take place. Bluetooth hops 1600 times/sec., which means that it remains in a channel for at most 625μs. Note that Bluetooth channels are 1 MHz-spaced, while the resolution of our scanner is 2 MHz, which implies that consecutive time slots may eventually coincide within this frequency window and result in a larger interference period. We model Bluetooth using the same method as for non-saturated traffic in WiFi, that is, we obtain the probability density function $p(x)$ for the *clear* periods of the channel, and the transmission time of Bluetooth packets for the *busy* periods. Fig. 3.5(c) shows the probability mass function $p(x)$ for the Bluetooth scenario. The Adaptive Frequency Hopping characteristic of Bluetooth leads to a smoother *cdf* curve compared to WiFi, because the *clear* periods are independent of the application run.

Microwave Oven Emulation

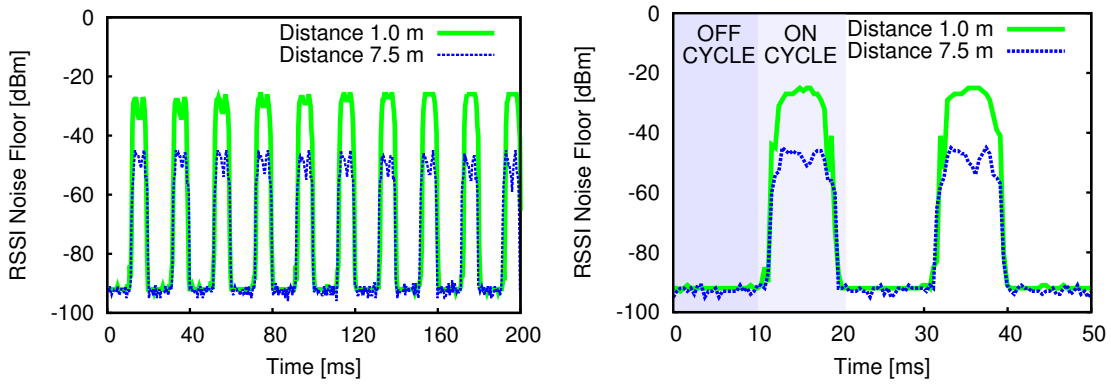
Microwave ovens are a kitchen appliance used to cook or warm food by passing non-ionizing microwave radiations to heat water and other polarized molecules within the food, usually at a frequency of 2.45 GHz. Therefore, they are a potential source of interference for sensornets operating in the 2.4 GHz spectrum.

The detailed characteristics of the interference patterns emitted by domestic microwave ovens depend on the model; nevertheless they all present the same basic properties.

Firstly, on a spectral basis, our experiments show that microwave ovens tend to interfere all the 802.15.4 channels, with a higher impact on channels 20-26. It is not possible to state with certainty which channel will be mostly affected, as our experiments confirm that the peak frequency of the ovens depends on multiple factors, including the oven content, the amount of water in the food, and the position within the oven, as all these parameters affect the temperature of the magnetron [46].

Secondly, on a temporal basis, the generated noise is rigorously periodic and depends on the frequency of the AC supply line powering the microwave oven [43]. For example, Fig. 3.6 shows the temporal pattern of the interference caused by a Lunik 200 microwave oven retrieved experimentally: in one period of approximately 20 ms, there is an 'on' and 'off' cycle, whose duration is roughly 10 ms each.

For all the above reasons, microwave oven interference is the simplest dynamic to model, as it follows a deterministic on/off sequence. Defining the period of the signal τ, the duty cycle λ (fraction of time the oven is 'on'), and hardcoding these two parameters into the HandyMote, we can generate interference patterns such as the ones shown in Fig. 3.4.



(a) Zoom out

(b) Zoom in

Figure 3.6.: Temporal characteristics of the interference caused by microwave ovens. The ovens emit frequencies with a periodic pattern with period $T \approx 20$ ms.

3.4. Testbed Configuration

As outlined in Section 3.1, we partition the area of a testbed into different cells to deal with the limited RF output power of the HandyMotes compared to interference sources such as WiFi or microwave ovens. In this section we explain how to configure the testbed, i.e., how to select the HandyMotes. This implies that every mote should be covered by a cell and cross-talk between neighboring cells should be minimized (i.e., a HandyMote does not interfere with motes outside of its cell).

3.4.1. Coverage and Cross-Talk

A key issue we need to understand is under which conditions the packet reception of a mote is actually affected by an interference signal generated by a HandyMote. The impact of interference on reception in the CC2420 radio is closely dependent on the modulation scheme used, namely OQPSK (Offset Quadrature Phase Shift Keying) and DSSS (Direct Sequence Spread Spectrum). With these modulation schemes, the interference signals generated by two HandyMotes do not simply "add up" at the receiver as it would be the case for ASK (Amplitude Shift Keying) used in older radios, but the receiver will pick the stronger of the two signals if their strength differs by a certain minimum. This is called co-channel rejection: according to [44], the CC2420 is able to receive a signal at -82 dBm if the second signal is at least 3 dB weaker.

In order to enable a HandyMote to interfere with the motes in its cell, we therefore need to make sure that a mote belonging to the cell will receive interference signals from that HandyMote with a signal strength at least 3 dB higher than the maximum strength of other signals that mote may receive. To minimize cross-talk between neighboring cells, we need to make sure that motes outside of the cell will receive that interference signal with a signal strength that is at least 3 dB weaker than the minimum strength of other signals that this mote may receive. Finally, we need to make sure that all testbed motes are covered by the cells.

In practice, an ideal configuration without cross-talk and with complete coverage typically

does not exist. Also, due to environmental dynamics, the amount of cross-talk and coverage may vary over time. We can only try to find a configuration that maximizes coverage and minimizes cross-talk. Note that there is a tradeoff between the size of the cells and the accuracy of the spatial distribution of generated interference: the smaller the cells, the higher is the spatial sampling resolution and the smaller are the cross-talk regions. However, smaller cells also implies that more HandyMotes are needed to cover the testbed.

3.4.2. A Theoretical Model

In this section we develop a theoretical model that allows us to estimate the radius of a cell such that a HandyMote can still interfere with all nodes in the cell. We will also model the amount of cross-talk between neighboring cells. Finally, we develop a model that allows us to estimate how many HandyMotes are at least needed to cover a testbed deployed over a geographical area A.

In order to derive the models, we need to make a number of practical assumptions. Firstly, we assume that the minimum distance between a pair of motes in the testbed equals D_{min} with typical values in the order of few meters. For example, it is common practice to place a mote in each room on an office floor. Secondly, we assume that we can reduce the RF output power level of the testbed motes to a value P_{mote} below the maximum of 0 dBm (e.g., -10 dBm) without loosing connectivity. In practice, this is often done to obtain multi-hop topologies with a large diameter even on the constrained space of an office floor. Thirdly, we assume that a mote is only able to receive a packet with a certain minimal signal strength P_{min}, with typical values in the order of -90 dBm. Finally, we assume the signal propagation can be modeled with the widely used log-normal model [30, 31, 50]:

$$P(d) = P_T - P_L(d_0) - 10 \cdot \eta \cdot \log_{10} \frac{d}{d_0} + \chi_\sigma \tag{3.1}$$

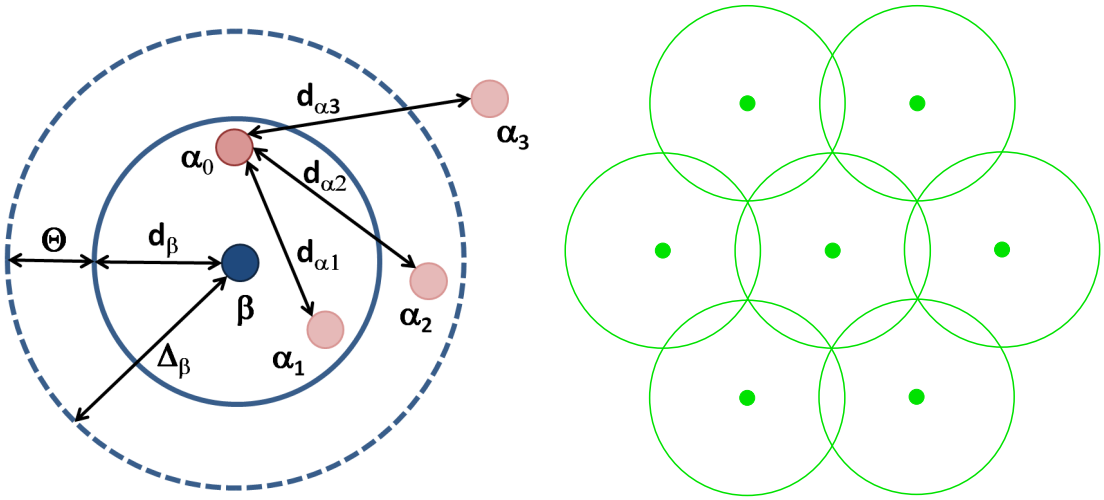
where $P_L(d_0)$ is the path loss measured at reference distance d_0, η is the path loss exponent, χ_σ is a zero-mean Gaussian random variable with standard deviation σ that models the random variation of the RSSI value due to shadowing. We use the well-known $P_L(2) = 46$ dBm, and the typical path loss exponent for indoor environments $\eta = 6$ without accounting for shadowing.

Consider the scenario in Fig. 3.7(a) with a HandyMote β and several motes α_i. We are interested in computing the cell radius d_β such that HandyMote β can block the reception of any message by motes contained in its cell (i.e., α_0 and α_1 in the figure). Further, we are interested in the radius Δ_β of the cross-talk region. The cross-talk region is defined as the region where the reception of a message by a mote (i.e., α_2 in the figure) may but need not be blocked by HandyMote β.

Knowing output power P_{handy} of the HandyMote and P_{mote} of the mote, as well as the minimum distance D_{min} between motes, we can compute the maximum RSSI P_{max} a mote can receive from another mote:

$$P_{max} = P_{mote} - P_L(d_0) - 10 \cdot \eta \cdot \log_{10} \frac{D_{min}}{d_0} \tag{3.2}$$

Using that value and the output power P_{handy} of the HandyMote, we can compute the radius of the cell d_β as follows:



(a) (b)

Figure 3.7.: JamLab's division in cells.

$$d_\beta = 10^{\frac{-P_{max}+P_{handy}-P_L(d_0)+10\cdot\eta\cdot\log_{10}(d_0)}{10\cdot\eta}} \tag{3.3}$$

Knowing the minimum RSSI P_{min} at which a mote can still receive a message, we can compute the radius of the cross-talk region Δ_β as follows:

$$\Delta_\beta = 10^{\frac{-P_{min}+P_{handy}-P_L(d_0)+10\cdot\eta\cdot\log_{10}(d_0)}{10\cdot\eta}} \tag{3.4}$$

From that we can compute the difference between the cell radius and the radius of the cross-talk region as $\Theta = d_\beta - \Delta_\beta$.

Knowing the cell radius d_β, we now derive a simple model to estimate the number of HandyMotes needed to cover a given area A. As illustrated in Fig. 3.7(b), we consider the sparsest-possible coverage of an area with disks. Ignoring border effects, the area covered by a single cell can be estimated with the area of the hexagon defined by the intersection points of one circle with the six adjacent circles. Dividing A by the area of the hexagon, we can estimate the number of HandyMotes N needed to cover area A:

$$\mathbf{N} = \frac{A}{\frac{3*\sqrt{3}}{2} * d_\beta^2} \tag{3.5}$$

We now illustrate those model with concrete examples. If we have a sparse testbed with a distance between nodes of $D_{min} = 4$ meters and transmission powers $P_{mote} = -15$ dBm, $P_{handy} = 0$ dBm, we derive $P_{max} \approx -80$ dBm and the radius of our cells $d_\beta = 8$ meters. This configuration would imply that the size of the cross-talk area $\Theta \approx 4$ meters when using $P_{min} = -90$ dBm.

This cell size is obviously very large, and the consequence would be that in theory only $N = 6$ HandyMotes would be needed to cover a testbed area $A = 750m^2$.

However, with this configuration the cross-talk area Θ is quite large. The accuracy of the regenerated interference may therefore be low as all nodes contained in cross-talk areas are potentially interfered by multiple HandyMotes in neighboring cells with different interference traces. To gain more accuracy, we need to decrease the size of the cross-talk area Θ. This can be achieved by reducing the radius of the cells by means of reducing P_{handy}, which requires more cells and HandyMotes to cover the testbed area. To obtain $\Theta \approx 2$ meters, using the same parameters as above, one would need to use a cell radius of $d_\beta \approx 4$ meters, which would imply that to cover the same testbed area $A = 750m^2$, we would need at least $N = 19$ HandyMotes.

3.4.3. Automatic Testbed Configuration

The original JamLab [11] does not provide an automatic approach to select a suitable set of HandyMotes to enable successful recreation of arbitrary interference patterns. Instead, a manual step-by-step procedure is used for HandyMote selection. This process is cumbersome and may lead to a suboptimal selection of HandyMotes.

To simplify the configuration of JamLab, we propose an automatic selection process that determines an optimal selection of HandyMotes without manual intervention. The original six step procedure is replaced by the following procedure:

1. In the first step an RSSI reading for each unidirectional link is obtained from the testbed. The RSSI values are measured by having the motes in the testbed sequentially broadcast a message and all others nodes record the maximum RSSI value.

2. The resulting measurements are fed to a constraint logic programming solver and an optimal selection of HandyMotes, which ensures that all regular motes are covered, is computed.

The resulting configuration may subsequently be used to program the testbed motes.

To implement this automatic process, we reformulated it as a constrained optimization problem. The automatic process assumes that a subset of the already installed motes is used as HandyMotes and no additional modes are deployed. Formally, the resulting constrained optimization problem can be expressed as follows:

$$\begin{aligned}
&\text{minimize } \|J\| \\
&\text{subject to } N \cap J = \emptyset, && \text{(3.6a)}\\
&\qquad \forall n \in N \exists j \in J : \text{neighbor}(j, n), && \text{(3.6b)}\\
&\qquad \forall n, o \in N \exists j \in J : \text{neighbor}(j, n) \wedge \text{neighbor}(o, n) && \text{(3.6c)}\\
&\qquad\qquad \rightarrow \text{rssi}(j, n) \geq \text{rssi}(o, n) + 3
\end{aligned}$$

with N being the set of regular motes and J being the set of HandyMotes. The $\text{rssi}(x, y)$ function returns the RSSI reading for the directional link between the sender x and the receiver y in dBm. The neighbor function is defined as follows:

$$\text{neighbor} : N \times N \rightarrow \mathbb{B}$$
$$(x, y) \mapsto \begin{cases} \text{true} & \text{rssi}(x, y) > -100 \\ \text{false} & \text{otherwise} \end{cases}$$

RSSI values of −100 dBm and less are considered to be insufficient for communication or jamming. Consequently, links with an RSSI value of −100 dBm or less are ignored.

The goal is to utilize as few motes as HandyMotes as possible while still maintaining full coverage of the testbed. Consequently, we minimize the cardinality of the set of HandyMotes. While minimizing the number of nodes, we also need to comply with the following set of constraints:

Constraint 3.6a Each mote can only assume one role at a time and cannot be configured as HandyMote and regular mote at the same time. As a consequence, we require the sets N and J to be disjunct.

Constraint 3.6b Each node needs to be covered by at least one HandyMote. To ensure coverage there needs to be a link with an RSSI value of more than −100 dBm from the HandyMote to the covered regular mote.

Constraint 3.6c One of the HandyMotes in the neighborhood of each regular mote needs to have a link to the mote with an RSSI value at least 3 dB higher than any other neighbor to ensure reliable jamming, as described in Sec. 3.4.1.

The optimization problem is implemented in the ECLiPSe Constraint Programming System [1]. The ECLiPSe framework allows to express optimization and constraint satisfiability problems in a superset of the Prolog programming language. ECLiPSe provides a number of constraint solver libraries and can interface with several third-party solvers. The ECLiPSe system can be easily integrated with custom software, nevertheless the current version of the JamLab configuration tool is a stand-alone ECLiPSe application.

The current implementation of the JamLab configuration tool employs the default hybrid integer/real interval arithmetic constraint solver *ic*. The *branch_and_bound* library is used to implement search.

As a prerequisite to run the automatic configuration, one needs to empirically record RSSI measurements for the directional communication between each pair of nodes in the testbed. The resulting data is converted into a list of Prolog facts that allow to directly query the RSSI strength for each link between a pair of motes. The selection of HandyMotes is internally represented by a Binary array and each mote ID is mapped to one position in the array. A value of 1 at this position indicates that the corresponding mote is selected as a HandyMote. This representation automatically ensures that Constraint 3.6a is met. Constraint 3.6b is implemented by summing up the number of HandyMotes in neighborhood of each mote and ensuring that this sum is greater or equal than one for each neighborhood of a regular mote. To implement Constraint 3.6c, for each mote the maximal RSSI value of all links between a HandyMote and this mote is determined. Second, the maximal RSSI value of all links between a regular mote and this mote is determined. A final check ensures that either the node is a HandyMote itself, in which case the constraint does not need to apply, or the maximal RSSI received from HandyMotes is more than 2 dB higher than the maximal RSSI receive from any regular node. After setting up the constraints, the branch-and-bound algorithm is used to determine the optimal solution with the least possible number of HandyMotes. Finally, the resulting selection of HandyMotes is returned as a list of mote identifiers.

The current model does not consider all properties of an ideal solution as outlined in Sec. 3.4.1. Nevertheless, it allows to create a suitable configuration with a minimal number of HandyMotes,

without requiring a cumbersome manual process. Future work could optimize the model and the resulting configuration, by also considering cross-talk. In addition, the performance of solving the optimization problem for very large networks could be improved.

3.5. Evaluation

In this section, we first evaluate the accuracy with which a HandyMote can regenerate a previously recorded interference trace in the time domain. We then compare the packet loss rate caused by real interferers and compare it to the one caused by the interference produced by JamLab.

3.5.1. Temporal Accuracy

We evaluate the accuracy with which a HandyMote can regenerate a previously recorded interference in the time domain. We run a HandyMote in regeneration mode in proximity of an active Lunik 200 microwave oven warming a bowl of tea. The HandyMote is placed at 1 meter distance from the oven, and records a trace of channel 24 at a sampling rate of 60 kHz.

Fig. 3.8(a) (top) shows the interference generated by the microwave as measured by the HandyMote. Next, the trace is quantized to single-bit resolution (middle). Finally, once the microwave oven stopped operating, the HandyMote plays back the recorded binary interference (bottom) using transmission power 0 dBm. As we can notice from the figure, the regeneration is quite accurate in the time domain.

We quantify the accuracy of the regenerated signal with respect to the originally recorded signal using the the *cross-correlation* coefficient ($\mathbf{c}$). We represent original and regenerated signals by the series $x(i)$ and $y(i)$, respectively, where $i = 1, \ldots, N$. These series are binary, and take 0 (clear channel) or 1 (busy channel) values. Considering this representation, $\mathbf{c}$ is given by:

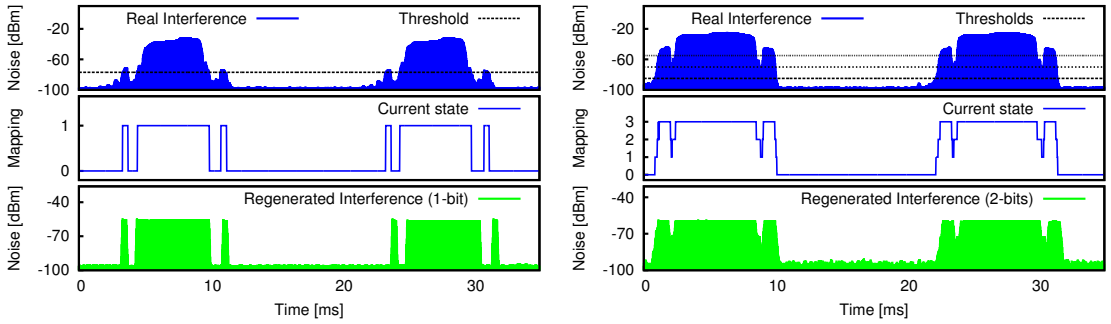
$$\mathbf{c} = \frac{\sum_{i=-\infty}^{\infty} x(i)y(k-i)}{rms(x)rms(y)} \tag{3.7}$$

where $rms()$ denotes the root mean square value of a signal. We tested eight pairs of original and regenerated samples and the maximum value of $\mathbf{c}$ was selected for each pair:

$$\mathbf{c}_{xy} = \max_{k \in [-(N-1),(N-1)]} \{\mathbf{c}\} \tag{3.8}$$

The average correlation $\mathbf{c}_{xy}$ is 0.93 with a standard deviation of 0.065. Hence, our implementation does a commendable job with respect to the cancellation of the jitter between sampled and regenerated interference and hence regenerates interference with a fairly high accuracy.

We carry out the same experiment using 2-bit quantization with thresholds -55, -70, and -80 dBm, and we then regenerate the interference using transmission power register levels 31,



(a) 1-bit Mapping

(b) 2-bit Mapping

Figure 3.8.: Regenerated interference of a microwave oven.

7, and 3 (i.e., 0, -10, -25 dBm), respectively. The results match the above ones with binary interference. Fig. 3.8(b) shows the regeneration process when using a two-bit quantization.

3.5.2. Impact on Packet Reception Rate

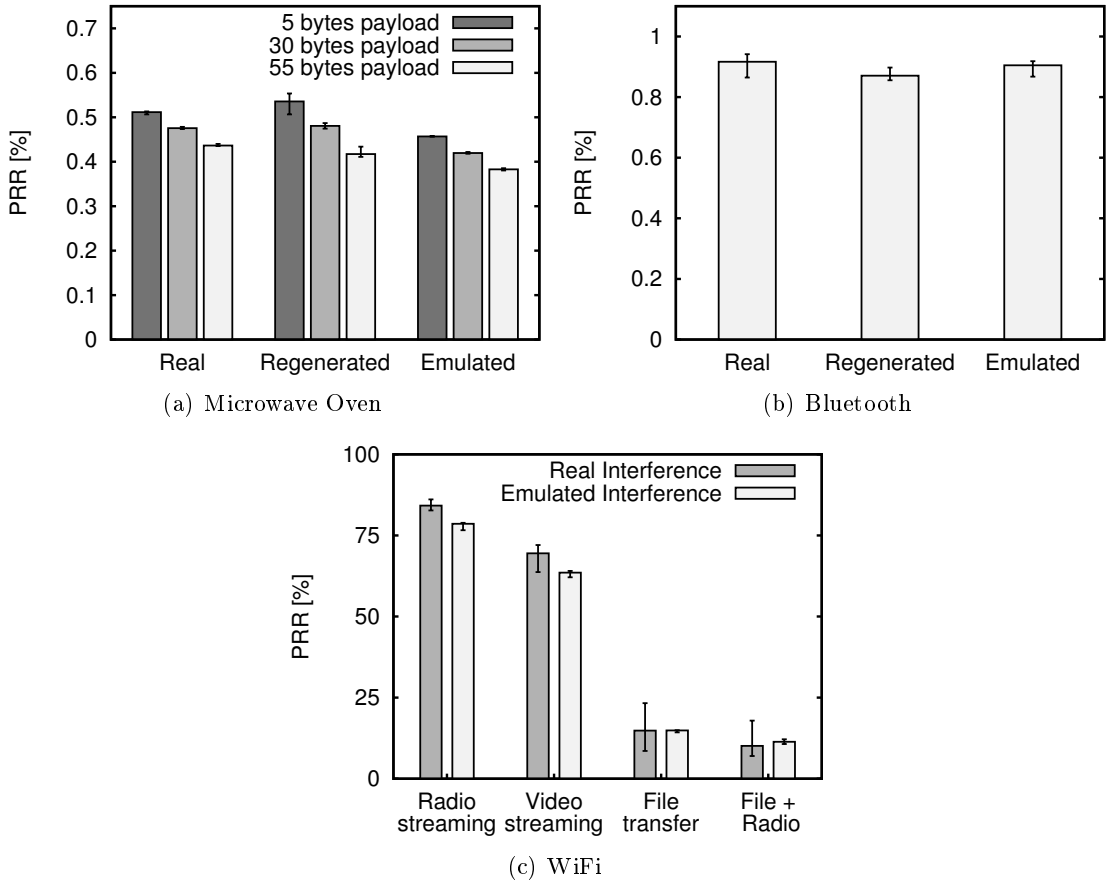
In this section we experimentally study the impact of interference on Packet Reception Rate (PRR), comparing the PRR for original, emulated, and regenerated interference signals. We use the same Lunik 200 microwave oven as in the previous experiment, and collect data at the receiver side of a pair of sensor nodes at about 1 meter distance, with the sender transmitting packets at a rate of 128 packets/sec. The sensor just transmits the packet without any clear channel assessments or duty cycling. We place an HandyMote between the two nodes and we run it both in emulation and regeneration mode, once the microwave oven stopped being active.

We carry out different experiments with different payload sizes, and we run the HandyMote using transmission power 0 dBm in both emulation and regeneration mode, such that the generated interference signal blocks communication between the sensor nodes.

Fig. 3.9(a) shows the results. The PRR collected when the microwave oven is active decreases when the payload size increases as the probability of periodic microwave interference hitting a packet increases with increasing payload size. The PRR obtained for regenerated interference differs by 5.6% from the original one, hence showing a reasonable accuracy. For emulated interference, the PRR differs from the original one by 12.83%, the reason for that being the noisy amplitude of the original interference signal as depicted in Fig. 3.4, such that occasionally the interference is too weak to block the transmission. In contrast, the emulated interference signal is binary and always blocks communication. Accuracy could be improved in this case by randomly varying the transmission power of the HandyMote as discussed in Sect. 3.3.3.

We repeat the experiments in presence of Bluetooth interference. We first measure PRR during a Bluetooth file transfer between a laptop and a mobile phone. We place the HandyMote between the 2 communicating motes and we measure the PRR obtained with original, emulated, and regenerated interference. We run the HandyMote in emulation mode using the models derived in Sect. 3.3.3.

Fig. 3.9(b) shows that the packet reception rate obtained under regenerated interference



(a) Microwave Oven

(b) Bluetooth

(c) WiFi

Figure 3.9.: Impact of real, emulated, and regenerated interference on packet reception rate.

differs by 5.02% from the the original one, while in emulation mode it differs by only 1.31%.

Finally, we repeat the experiment with WiFi interference. Using the same setup as above, we run the HandyMote in emulation mode using the models derived in Sect. 3.3.3 while generating WiFi traffic from a laptop according to the scenarios presented on Table 3.2.

Figure 3.9(c) shows the results. Also in this case the HandyMote generates interference quite accurately, and the difference between the PRR obtained under real interference and the one obtained under emulation varies between 0.25% and 8.56%. The reason for this difference is that emulation repeats the same pattern over and over, while actual WiFi interference might change in time, due, for example, to TCP adaptation mechanisms.

4. Conclusions

The ability of a testbed to replay realistic environmental effects plays a crucial role for the investigation of protocol performance. In this deliverable, we have presented the design and implementation of TempLab and JamLab, two low-cost extensions of sensornet testbeds that allow to study the impact of temperature and interference on the performance of IoT protocols.

These testbed extensions allow to rerun experiments under identical environmental conditions and hence play a crucial role for the investigation of protocol performance in WP1 [15, 51] and WP2 [12]. For example, TempLab has played a fundamental role in identifying and quantifying the strong impact of temperature on low-power radio communication and capturing precise platform models [13], whereas JamLab has been used to evaluate the performance of newly designed protocols [12].

A. Source Files

This chapter describes the main software components appended to this document in the `source` root directory. After describing the folder hierarchy of the `source` directory, it guides the reader towards the content of the different folders.

A.1. Structure of the `source` Directory

The structure of the `source` directory, sorted alphabetically, is the following:

- **TempLab**
  - **closed_loop_controller**
    - **Main.cpp**
    - **Makefile**
    - **PIController.hpp**
    - **start_cl_controller.sh**
    - **timeTable.csv**
    - **turn_off.cpp**
  - **open_loop_controller**
    - **Main.cpp**
    - **Makefile**
    - **start_ol_controller.sh**
    - **timeTable.csv**
  - **temperature_reading**
    - **control_adc.c**
    - **control_adc.h**
    - **Makefile**
    - **project-conf.h**
    - **temperature_reading.c**
  - **testbed_scripts**
    - **alarm_templab.sh**
    - **closing_tty.sh**
    - **closing_tty_removing.sh**

    - starting_tty.sh
    - starting_tty_forwarding.sh

- JamLab
  - emulation
    - jamlab.c
    - Makefile
    - project-conf.h
    - settings_cc2420_interferer.c
    - settings_cc2420_interferer.h
    - settings_cc2420_rssi.c
    - settings_cc2420_rssi.h
    - settings_noisefloor_sampling.c
    - settings_noisefloor_sampling.h
  - optimization
    - input.ecl
    - opt.ecl
  - regeneration
    - 1bit_regeneration.c
    - 2bit_regeneration.c
    - interferer_settings.c
    - interferer_settings.h
    - Makefile
    - project-conf.h
  - rssi_collection
    - automat_testbed.c
    - java_parser.java
    - Makefile
    - project-conf.h
    - settings_addresses.h
    - settings_cc2420_rssi.c
    - settings_cc2420_rssi.h

A.2. TempLab Source

The *TempLab* folder contains the code used to implement TempLab's open- and closed-loop controllers as well as the scripts used to manage the communication with the sensor nodes.

open_loop_controller The open-loop controller is a standalone application implemented using the open-zwave library, an open-source interface to Z-Wave networks. The main application program is *Main.cpp*, that forms the network and varies the intensity of the wireless dimmers according to a predefined dimming level at a given point in time. The file *timeTable.csv* contains the dimming levels to which each node in the network should be set. The file has the format (time, dimmer ID, dimming level), where time is specified as a string with format "%d %H:%M:%S", the dimmer ID is an integer number between 0 and 255, whereas the dimming level is an integer number in the range [0-99]. The files *Makefile* and *start_ol_controller.sh* are used to compile and run the controller.

closed_loop_controller To precisely regenerate trace- or model-based temperature profiles, TempLab uses a closed-loop proportional-integral (PI) controller that tries to minimize the difference between the desired temperature profile and the on-board temperature of the sensor node of interest. The closed-loop PI controller is a standalone application implemented using the open-zwave library, an open-source interface to Z-Wave networks. The main application program is *Main.cpp*, that forms the network and varies the intensity of the wireless dimmers according to a desired temperature profile at a given point in time. The file *timeTable.csv* contains the desired temperature to which a specific node in the network should be set. The file has the format (time, dimmer ID, desired temperature), where time is specified as a string with format "%d %H:%M:%S", the dimmer ID is an integer number between 0 and 255, whereas the desired temperature is a floating point number specifying the desired temperature in Celsius degree. The file *turn_off.cpp* is an application that turns off all the lamps in the network and is used whenever an anomalous behaviour of the PI controller is detected. The files *Makefile* and *start_ol_controller.sh* are used to compile and run the controller.

temperature_reading TempLab uses an in-band approach using the USB back-channel to periodically convey temperature readings to the controller. This folder contains the Contiki code used to measure the temperature on each node in the testbed. This task is carried out using a low-priority routine executing only when the processor is idle. The main source file is *temperature_reading.c*.

testbed_scripts This folder contains the scripting used to manage the testbed, i.e., auxiliary scripts that specify how to read the output from the sensor nodes and to verify that the temperature on sensor nodes does not exceed predefined bounds. The files *starting_tty.sh* and *starting_tty_forwarding.sh* run a customized version of Contiki's serialdump that reads the serial output from the nodes and prints it to file. The files *closing_tty.sh* and *closing_tty_removing.sh* are used to archive the collected traces for persistent storage. Finally, the script *alarm_templab.sh* verifies that the temperature on sensor nodes does not exceed predefined bounds.

A.3. JamLab Source

The *JamLab* folder contains the code used to implement JamLab's (re)generation of interference, as well as the code used to derive an automatic testbed configuration.

emulation This folder contains the code used to let a HandyMote emulate three major sources of external interference on the 2.4 GHz ISM band: WiFi and Bluetooth devices, as well as microwave ovens. The main source file is *jamlab.c*.

optimization The *optimization* folder contains code to automatically select a suitable set of HandyMotes as described in Sec. 3.4.3. The optimization problem is implemented in the ECLiPSe Constraint Programming System.

As a prerequisite to run the automatic configuration, one needs to empirically record RSSI measures for the directional communication between each pair of nodes in the testbed. A special mote firmware is used to record these RSSI readings in the testbed (see paragraph rssi_collection). The results are stored in a simple trace format that can be read by the ECLiPSe solver to setup a model of the network topology with the help of the `input/1` predicate defined in the file *input.ecl*. Based on this input, an optimal selection of HandyMotes is computed which employs as little HandyMotes as possible without violating any of the given constraints.

The actual implementation of the optimization process can be found in the file *opt.ecl*. It employs the default hybrid integer/real interval arithmetic constraint solver *ic*. The *branch_and_bound* library is used to implement search. A suitable selection of HandyMotes can be generated by calling the main predicate `solve/2` and providing the path of a previously recored trace file as first parameter. The second parameter is unified to a list of the chosen HandyMotes. A more throughout description of the representation of the optimization problem can be found in Sec. 3.4.3.

regeneration This folder contains the code used to let a HandyMote measure and replay the interference recorded in the environment using a quantization of either 1 or 2 bits. The main source files are *1bit_regeneration.c* and *2bit_regeneration*.

rssi_collection This folder contains the code used to capture the RSSI between each pair of nodes in the testbed. All nodes run a Contiki program *automat_testbed.c* in which each node broadcasts packets and computes statistics about the received signals strength from each neighbour. These statistics are printed on stdout and then parsed by *java_parser.java*.

Bibliography

[1] "The ECLiPSe Constraint Programming System," http://www.eclipseclp.org/.

[2] A. Jiménez and J.R. Martínez de Dios and J.M. Sánchez-Matamoros and A. Ollero, "Design of a testbed for cooperation of robots and wireless sensor network," in *Proc. of the* 7^{th} *EWSN Conf.*, Feb. 2010.

[3] ——, "Towards an open testbed for the cooperation of robots and wireless sensor networks," in *Proc. of the* 10^{th} *Robotica Conf.*, Mar. 2010.

[4] N. Baccour, A. Koubâa, L. Mottola, H. Youssef, M. Z. niga, C. Boano, and M. Alves, "Radio link quality estimation in wireless sensor networks: a survey," *ACM TOSN*, vol. 8, no. 4, Nov. 2012.

[5] K. Bannister, G. Giorgetti, and S. Gupta, "Wireless sensor networking for hot applications: Effects of temperature on signal strength, data collection and localization," in *Proc. of the* 5^{th} *HotEmNets Worksh.*, Jun. 2008.

[6] G. Bianchi, "Performance analysis of the IEEE 802.11 distributed coordination function," *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 3, pp. 535–547, 2000.

[7] C. Boano, Z. He, Y. Li, T. Voigt, M. Zúñiga, and A. Willig, "Controllable radio interference for experimental and testing purposes in wireless sensor networks," in *Proc. of the* 4^{th} *SenseApp Worksh.*, Oct. 2009, pp. 865–872.

[8] C. Boano, K. Römer, Z. He, T. Voigt, M. Zúñiga, and A. Willig, "Generation of Controllable Radio Interference for Protocol Testing in Wireless Sensor Networks," in *Proc. of the* 7^{th} *SenSys Conf., demo session*, Nov. 2009, pp. 301–302.

[9] C. Boano, T. Voigt, A. Dunkels, F. Österlind, N. Tsiftes, L. Mottola, and P. Suárez, "Exploiting the LQI variance for rapid channel quality assessment," in *Proc. of the* 8^{th} *IPSN Conf., poster session*, Apr. 2009, pp. 369–370.

[10] C. Boano, J. Brown, N. Tsiftes, U. Roedig, and T. Voigt, "The impact of temperature on outdoor industrial sensornet applications," *IEEE Trans. Ind. Informatics*, vol. 6, no. 3, pp. 451–459, Aug. 2010.

[11] C. Boano, T. Voigt, C. Noda, K. Römer, and M. Zúñiga, "JamLab: Augmenting sensornet testbeds with realistic and controlled interference generation," in *Proc. of the* 10^{th} *IPSN Conf.*, Apr. 2011, pp. 175–186.

[12] C. Boano, M. Zúñiga, K. Römer, and T. Voigt, "JAG: Reliable and predictable wireless agreement under external radio interference," in *Proc. of the* 33^{th} *RTSS Conf.*, Dec. 2012, pp. 315–326.

[13] C. Boano, H. Wennerström, M. Zúñiga, J. Brown, C. Keppitiyagama, F. Oppermann, U. Roedig, L.-Å. Nordén, T. Voigt, and K. Römer, "Hot Packets: A systematic evaluation of the effect of temperature on low power wireless transceivers," in *Proc. of the* 5^{th} *ExtremeCom Conf.*, Aug. 2013, pp. 7–12.

[14] C. Boano, M. Zúñiga, J. Brown, U. Roedig, C. Keppitiyagama, and K. Römer, "Templab: A testbed infrastructure to study the impact of temperature on wireless sensor networks," in *Under Submission*, Oct. 2013.

[15] J. Brown, U. Roedig, M. A. Zúñiga, C. Boano, N. Tsiftes, K. Römer, T. Voigt, and K. Langendoen, "D-1.2 - report on learning models parameters," http://www.relyonit.eu/, RELYonIT: Research by Experimentation for Dependability on the Internet of Things, Grant Agreement no: 317826, Tech. Rep., Jun. 2013.

[16] A. Chattopadhyay, "Basic RF testing of CCxxxx devices," Application Report SWRA370, Aug. 2011.

[17] https://conet.us.es/cms/, CONET Integrated Testbed.

[18] http://sourceforge.net/projects/contikiprojects, The Contiki Projects Community.

[19] http://www.crew-project.eu/, The CREW Project: Cognitive Radio Experimentation World.

[20] M. Doddavenkatappa, M. Chan, and A. Ananda, "Indriya: A low-cost, 3D wireless sensor network testbed," in *Proc. of the* 7^{th} *TridentCom Conf.*, Apr. 2011, pp. 302–316.

[21] A. Dunkels, F. Österlind, N. Tsiftes, and Z. He, "Software-based on-line energy estimation for sensor nodes," in *Proc. of the* 4^{th} *EmNetS Worksh.*, Jun. 2007.

[22] A. Dunkels, Björn Grönvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in *Proc. of the* 1^{st} *EmNetS Worksh.*, Nov. 2004.

[23] S. Duquennoy, F. Österlind, and A. Dunkels, "Lossy links, low power, high throughput," in *Proc. of the* 9^{th} *SenSys Conf.*, Nov. 2011, pp. 12–25.

[24] E. Ertin, A. Arora, R. Ramnath, M. Sridharan, and V. Kulathumani, "Kansei: A testbed for sensing at scale," in *Proc. of the* 5^{th} *IPSN Conf.*, Apr. 2006, pp. 339–406.

[25] H. Fotouhi, M. Z. niga, M. Alves, A. Koubâa, and P. Marrón, "Smart-HOP: A reliable handoff mechanism for mobile wireless sensor networks," in *Proc. of the* 9^{th} *EWSN Conf.*, Feb. 2012, pp. 131–146.

[26] M. Garetto and C. Chiasserini, "Performance analysis of 802.11 WLANs under sporadic traffic," in *Proc. of the* 4^{th} *NETWORKING Conf.*, May 2005.

[27] V. Handziski, A. Köpke, A. Willig, and A. Wolisz, "TWIST: a scalable and reconfigurable testbed for wireless indoor experiments with sensor networks," in *Proc. of the* 2^{nd} *RealMAN Worksh.*, May 2006, pp. 63–70.

[28] D. Johnson, T. Stack, R. Fish, D. Flickinger, L. Stoller, R. Ricci, and J. Lepreau, "Mobile Emulab: A robotic wireless and sensor network testbed," in *Proc. of the* 25^{th} *INFOCOM Conf.*, Apr. 2006, pp. 1–12.

[29] X. Ju, H. Zhang, and D. Sakamuri, "NetEye: A user-centered wireless sensor network testbed for high-fidelity, robust experimentation," *Int. J. Commun. Syst.*, vol. 25, no. 9, pp. 1213–1229, Sep. 2012.

[30] D. Lymberopoulos, Q. Lindsey, and A. Savvides, "An empirical characterization of radio signal strength variability in 3d IEEE 802.15.4 networks using monopole antennas," in *Proc. of the* 3^{rd} *EWSN Conf.*, Feb. 2006, pp. 326–341.

[31] E. Miluzzo, X. Zheng, K. Fodor, and A. Campbell, "Radio characterization of 802.15.4 and its impact on the design of mobile sensor networks," in *Wireless Sensor Networks*, ser. Lecture Notes in Computer Science, vol. 4913. Springer Berlin/Heidelberg, 2008, pp. 171–188.

[32] R. Musaloiu-E. and A. Terzis, "Minimising the effect of WiFi interference in 802.15.4 wireless sensor networks," *International Journal of Sensor Networks (IJSNet)*, vol. 3, no. 1, pp. 43–54, Dec. 2007.

[33] H. Packard, "Fundamentals of quartz oscillators," Application Note 200-2, May 1997.

[34] C. Park, K. Lahiri, and A. Raghunathan, "Battery discharge characteristics of wireless sensor nodes: An experimental analysis," in *Proc. of the* 2^{nd} *SECON Conf.*, Sep. 2005, pp. 430–440.

[35] M. Petrova, L. Wu, P. Mähönen, and J. Riihijärvi, "Interference measurements on performance degradation between colocated IEEE 802.11g/n and IEEE 802.15.4 networks," in *Proc. of the* 6^{th} *ICN Conf.*, Apr. 2007, pp. 93–98.

[36] J. Polastre, J. Hill, and D. Culler, "Versatile low power media access for wireless sensor networks," in *Proc. of the* 2^{nd} *SenSys Conf.*, Nov. 2004, pp. 95–107.

[37] D. Puccinelli and S. Giordano, "ViMobiO: Virtual mobility overlay for static sensor network testbeds," in *Proc. of the* 4^{th} *EXPonWireless Worksh.*, Jun. 2009, pp. 1–6.

[38] P. Rathod, O. Dabeer, A. Karandikar, and A. Sahoo, "Characterizing the exit process of a non-saturated IEEE 802.11 wireless network," in *Proc. of the* 10^{th} *MobiHoc Conf.*, May 2009.

[39] T. Schmid, "Time in wireless embedded systems," Ph.D. dissertation, University of California, 2009.

[40] A. Sikora and V. Groza, "Coexistence of IEEE 802.15.4 with other systems in the 2.4 GHz-ISM-Band," in *Proc. of the* 22^{th} *I2MTC Conf.*, May 2005, pp. 1786–1791.

[41] J. Slipp, C. Ma, N. Polu, J. Nicholson, M. Murillo, and S. Hussain, "WINTeR: Architecture and applications of a wireless industrial sensor network testbed for radio-harsh environments," in *Proc. of the* 6^{th} *CNSR Conf.*, May 2008, pp. 422–431.

[42] K. Srinivasan and P. Levis, "RSSI is under appreciated," in *Proc. of the* 3^{rd} *EmNets Worksh.*, May 2006.

[43] T. Taher, M. Misurac, J. LoCicero, and D. Ucci, "Microwave oven signal modelling," in *Proc. of the IEEE WCNC Conf.*, Apr. 2008.

[44] *CC2420 datasheet - 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver*, Texas Instruments, feb 2013.

[45] N. Tsiftes, A. Dunkels, Z. He, and T. Voigt, "Enabling Large-Scale Storage in Sensor Networks with the Coffee File System," in *Proc. of the* 8^{th} *IPSN Conf.*, Apr. 2009.

[46] M. Vollmer, "Physics of the microwave oven," in *Physics Education 39/1.* IOP Publishing Ltd, 2004, pp. 74–81.

[47] Q. Wang and T. Zhang, "Source traffic modeling in wireless sensor networks for target tracking," in *Proc. of the* 5^{th} *PE-WASUN Symp.*, Vancouver, Canada, Oct. 2008, pp. 96–100.

[48] H. Wennerström, F. Hermans, O. Rensfelt, C. Rohner, and L.-A. Nordén, "A long-term study of correlations between meteorological conditions and 802.15.4 link performance," in *Proc. of the* 10^{th} *SECON Conf.*, Jun. 2013.

[49] G. Werner-Allen, P. Swieskowski, and M. Welsh, "MoteLab: a wireless sensor network testbed," in *Proc. of the* 4^{th} *IPSN Conf.*, Apr. 2005, pp. 483–488.

[50] M. Zúñiga and B. Krishnamachari, "Analyzing the transitional region in low-power wireless links," in *Proc. of the* 1^{st} *SECON Conf.*, Oct. 2004, pp. 517–526.

[51] M. Zúñiga, , C. Boano, J. Brown, C. Keppitiyagama, F. Oppermann, P. Alcock, N. Tsiftes, U. Roedig, K. Römer, T. Voigt, , and K. Langendoen, "D-1.1 - report on environmental and platform models," http://www.relyonit.eu/, RELYonIT: Research by Experimentation for Dependability on the Internet of Things, Grant Agreement no: 317826, Tech. Rep., Jun. 2013.

