Whatever is worth doing is worth doing well.

– Sir Philip Dormer Stanhope, Earl of Chesterfield, 1694-1773

University of Alberta

Wireless Sensor Network Development for Urban Environments

by

Nicholas M. Boers

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computing Science

Fall 2011
Edmonton, Alberta

To my grandparents, parents, sister, and Xiye.

Abstract

In this thesis, we focus on topics relevant to developing and deploying large-scale wireless sensor network (WSN) applications within real dynamic urban environments. Given few reported experiences in the literature, we designed our own such network to provide a foundation for our research.

The Smart Condo, a well-defined project with the goal of helping people age in place, provided the setting for our WSN that would non-intrusively monitor an occupant and environment. Although we carefully designed, developed, and deployed the network, all of our planning did not prepare us for a key challenge of that environment: significant radio-frequency interference. Most researchers tend to ignore the existence of interference along with its potentially serious implications: beyond impacting network performance, it can lead researchers to misleading or unrealistic conclusions.

Interference is a particularly difficult problem to study because it varies in time, space, and intensity. Other researchers have typically approached the problem by investigating only known interferers. Instead, we approach the problem more generally and consider interference of unknown origins. We envision nodes periodically observing their environment, recognizing patterns in those observations, and responding appropriately, so we use only standard WSN nodes for our data collection. Unfortunately, collecting high-resolution data is difficult using these simple devices, and to the best of our knowledge, other researchers have only used them to collect rather coarse data.

Within the Smart Condo urban environment, we recorded a transceiver's received power level at 5000 Hz, a higher rate than we encountered elsewhere in the literature, using 16 synchronized nodes. We explored traces from 256 channels and observed a number of recurring patterns; we then investigated classifying traces automatically and obtained rather promising results. We focused on the two patterns most detrimental to packet reception rates and fur-

ther investigated both sampling and classification techniques tailored to them. As part of our work, we extended our simulator, making it capable of generating impulsive interference, and developed a proof-of-concept pattern-aware medium access control (MAC) protocol.

Through experiments using both the simulator and WSN devices, we evaluated the classifier and proof-of-concept MAC. Our results show that impressive gains in the packet reception rates are possible when nodes can recognize and appropriately react to interference. Using our techniques, nodes can communicate more efficiently by reducing the number of failed transmissions and consequently decreasing overall network congestion.

Acknowledgements

First and foremost, I'd like to thank my supervisors (Drs. Pawel Gburzynski and Ioanis Nikolaidis), the examining committee, and my other colleagues in the department. My supervisors provided ideas, encouragement, advice, and comments over the past five years that helped shape this research. The excellent feedback of the examining committee, Drs. Lu, Nascimento, Cockburn, and Lutfiyya, helped improve this thesis. My other colleagues also provided me with ideas and feedback on several occasions.

The Smart Condo team, primarily Drs. Stroulia, Liu, and King and Mr. Lederer, gave enormous support. I was given the freedom to explore and was provided with advice and assistance when needed. Other members of the team, specifically David Chodos and Jianzhao Huang, made the 3D visualization of the Smart Condo a reality.

My family deserves a great deal of thanks for their support. My grandparents, parents, sister, and wife (Xiye) provided constant love and encouragement. Moreover, in the final push before submitting this thesis to the committee, my parents and Xiye carefully proofread it.

Finally, I'd like to thank those who provided me with financial support. These groups include Alberta Advanced Education and Technology, the University of Alberta, the Natural Sciences and Engineering Research Council of Canada (NSERC), the Informatics Circle of Research Excellence (iCORE), Mathematics of IT and Complex Systems (MITACS), Alberta Scholarship Programs, and the Killam Trusts.

Contents

List of Tables

List of Figures

List of Acronyms

ACK	acknowledgement
ADC	analog-to-digital converter
ALS	Assignment and Licensing System
API	application programming interface
ARFF	attribute-relation file format
ARQ	automatic repeat request
AWGN	additive white Gaussian noise
CPM	closest-fit pattern matching
CPU	central processing unit
CR	cognitive radio
CRC	cyclic redundancy check
CSMA	carrier sense multiple access
CSMA/CA	carrier sense multiple access with collision avoidance
CSV	comma-separated values
DAC	digital-to-analog converter
DAG	directed acyclic graph
DC	direct current
FN	false negative
FP	false positive
FSK	frequency-shift keying
FSM	finite state machine
GIS	geographic information system
GNU	GNU's Not Unix
GPIO	general purpose input/output
GPRS	general packet radio service
HTTP	HyperText Transfer Protocol
ID	industrial design; identifier
IEEE	Institute of Electrical and Electronics Engineers
IPC	inter-process communication
ISM	industrial, scientific and medical
JTAG	Joint Test Action Group
LBT	listen before talk
LED	light-emitting diode
LSSA	least-squares spectral analysis
MAC	medium access control
MCU	microcontroller unit
NACK	negative acknowledgement

OS	operating system
OSS	operations support system
OT	occupational therapy
PA-MAC	pattern-aware medium access control
PC	personal computer
PHP	PHP: Hypertext Preprocessor
PIR	passive infrared
PRR	packet reception rate
RAM	random access memory
REST	representational state transfer
RF	radio frequency
RISC	reduced instruction set computing
RSSI	received signal strength indicator
RT	real-time
SER	symbol error rate
SIDE	Sensors In a Distributed Environment
SINR	signal-to-interference-plus-noise ratio
SL	Second Life
SMURPH	System for Modeling Unslotted Real-time PHenomena
SNR	signal-to-noise ratio
SPI	serial peripheral interface
SQL	structured query language
TAN	tree-augmented naïve Bayes
TARP	Tiny Ad hoc Routing Protocol
TDMA	time division multiple access
TI	Texas Instruments
TN	true negative
TP	true positive
TTL	transistor-transistor logic
UART	universal asynchronous receiver/transmitter
USART	universal synchronous/asynchronous receiver/transmitter
USB	universal serial bus
VNETI	Versatile NETwork Interface
VUE^2	Virtual Underlay Execution Engine
WLAN	wireless local area network
WSN	wireless sensor network
XML	extensible markup language

Chapter 1

Introduction

A wireless sensor network (WSN) consists of a geographically-distributed collection of devices – essentially special-purpose computers – called *wireless nodes* or *motes*. Each device has a radio transceiver that allows it to communicate wirelessly and do so over often short distances. It may have a number of attached sensors and actuators for observing and affecting its surroundings, respectively. It often transmits its observations to a collection point called a *sink* and receives acknowledgements and commands from the sink.

Although it would be convenient to design WSNs in a vacuum or even isolation, all WSNs operate within some sort of environment. The specific environment can affect the WSN: in some cases, it may present particular opportunities, and in others, it may impose inconvenient challenges. For example, nodes placed within a densely populated building may be able to take advantage of electrical outlets as a power source, but at the same time, unknown devices may interfere with their communication.

Our research centres on topics relevant to developing and deploying large-scale wireless sensor network applications within real dynamic *urban* environments. Urban environments can exist on a number of scales, ranging from a single-occupancy apartment unit to a much larger metropolitan area. They are characterized by a number of features (Section 1.2), with the most relevant to our present work being the probable presence of high and dynamic interference. Some other researchers have focused special attention on these environments, too, but they have looked at different problems. For example, Oehen [59] calls them *non-deterministic environments* as he investigates deployment support networks.

Mainstream research often overlooks their unique characteristics and focuses – intentionally or not – on rural or even fictitious scenarios. When working with WSNs and even more typically their simulators, researchers often make a number of simplifying assumptions that detach their work from reality [46]. Some of these assumptions include

1. representing node locations as simple two-dimensional (x, y) values and failing to account for hills, buildings, or other obstacles,
2. modelling a radio's transmission range as a circle,
3. assuming that all radios have an equal transmission range,
4. believing that the link between two communicating nodes is symmetric,

5. ignoring the possibility of corrupt transmissions (packets), and
6. calculating signal strengths simply as a function of distance.

The effect of using these assumptions can be serious: they can lead researchers to misleading or unrealistic conclusions [47]. Our work within urban environments suggests an addition to the list published in [46]:

7. ignoring the existence of external interference.

We have noticed that researchers rarely consider the serious impact that external interference can have on network performance [57].

Given very few reported experiences from deployed *urban* networks, we began exploring this area ourselves. Our foray started within the context of a well-defined urban project named the Smart Condo (Chapter 2). After defining the project's requirements and establishing its architecture, we began to develop software for both the nodes and support systems (Chapter 3). To speed up this development, we made extensive use of the SMURPH/SIDE simulator. Using this powerful tool, we emulated the software and tested it with the many other involved components. After seeing that the (emulated) system achieved its objectives, we deployed the nodes in the real environment – the Smart Condo.

After deploying the network, we soon discovered that although the system generally met its design goals, characteristics of the particular environment negatively affected the wireless communication. When we searched for the potential sources of the problems, we found the presence of strong interference. In our desire to alleviate the effect of the interference, we had to address a number of sub-problems that include

- sampling received signal strength indicator (RSSI) values at a high rate while using low-end hardware (Chapter 4),
- generalizing the patterns that we observed in the time series (RSSI) traces (Chapter 4),
- using low sampling rates to obtain feature vectors still suitable for classification (Chapter 5),
- classifying channels within the strict resource confines of low-end WSN nodes (Chapter 5), and
- exploiting in-node classifications to improve network performance (Chapter 6).

While our results from investigating these topics will benefit WSN research generally, they are particularly relevant to the emerging field of cognitive radios, which attempt to opportunistically use unoccupied frequency bands after predicting the near-term occupancy of a channel [99].

In this chapter, we review WSNs, both in terms of their hardware and software, and identify some of their applications. We then describe our assumptions for the specific class of WSN that we call an *urban* WSN. Most of these assumptions relate not to the communication channel between nodes, which it turns out is highly unpredictable, but instead to the environment's more general opportunities and challenges. After this gentle introduction, we list some topics relevant to our research. Finally, we close the chapter by enumerating our contributions and providing an outline for this thesis.

1.1 Wireless sensor networks

In its typical form, a wireless sensor *node* consists of a microcontroller unit (MCU), a radio transceiver, and supporting components including a voltage regulator, a clock, resistors, capacitors, and light-emitting diodes (LEDs) all soldered to a circuit board. One or more sensors or actuators and an energy source may be either connected to or integrated on this board. Given that these nodes are often battery powered, energy consumption plays a key role in the components selected for them.

The MCUs used in these devices are often from the lower end of the performance spectrum. Two popular examples are the Texas Instruments MSP430 series, built around a 16-bit central processing unit (CPU) that now runs at up to 25 MHz, and the Atmel megaAVR series, built around an 8-bit CPU that now runs at up to 20 MHz. Table 1.1 describes three specific models that have been used within WSN nodes. Notice that they often have just a few kilobytes of random-access memory (RAM) and flash storage in addition to their relatively slow clock speeds. Although these MCUs are rather meagre computing platforms, they consume very little energy when active and offer a variety of reduced-functionality low-power modes that allow them to use just a few microamps of current. Given the right settings, they can literally run for years on even relatively small batteries (e.g., AAA- or AA-sized batteries). The boards that we use incorporate the MSP430 running at either 5 MHz for the RF Monolithics DM2200 [76] (given a 9 V battery with the voltage regulator) or 4.5 MHz for the Olsonet EMSPCC11 [62] (given two AA batteries).

Table 1.1: Specifications for three MCUs used in wireless sensor nodes. They all use the reduced instruction set computing (RISC) architecture. I/O pins support data input and output from the MCU. An ADC converts analog signals to digital values, and a DAC makes the opposite conversion. Finally, the USART allows the MCU to communicate with other receiver/transmitter devices asynchronously.

	MSP430F148	MSP430F1611	ATmega128L
architecture	RISC	RISC	RISC
bit width	16	16	8
frequency	$\leq$ 8 MHz	$\leq$ 8 MHz	$\leq$ 8 MHz
flash memory	48 KB	48 KB	128 KB
RAM	2 KB	10 KB	4 KB
I/O pins	48	48	53
ADC	12-bit	12-bit	10-bit
DAC	N/A	12-bit	10-bit
USART	2	2	2
hardware multiplier	N/A	integer	fractional
example usage	[76]	[62]	[26]

Many different radio transceivers are suitable for these devices as well,

with low energy consumption again being their common characteristic. Over the course of our research, we used two rather different transceivers: the RF Monolithics TR8100 [75] incorporated in the DM2200 and the Texas Instruments/Chipcon CC1100 [91] incorporated in the EMSPCC11. The former has very few features, e.g., it only communicates at a single frequency (916.5 MHz), at a single power level, at a single data rate, and with a single physical modulation. The latter, on the other hand, has a large number of parameters, e.g., frequencies, power levels, data rates, and physical modulations. Much research now uses transceivers built around the IEEE 802.15.4 (ZigBee) standard, e.g., the Texas Instruments/Chipcon CC2420 [89], which shares similarities with our CC1100 transceiver. One such similarity relevant to our work is the easy retrieval of the received signal strength indicator (RSSI) value – a measure of the power received at the transceiver. Given the CC2420 rather than the CC1100, we could have used the same methodology to perform similar research in the 2.4 GHz spectrum rather than the 900 MHz spectrum.

A variety of sensors are available to detect physical phenomena such as doors opening, temperature, humidity, motion, and tactile pressure. Furthermore, several sensors may be integrated on a single circuit board, which is called a sensor board, e.g., [53]. Some sensors produce digital signals that can serve as input to the MCU directly, and others produce analog ones that must be interpreted by an analog-to-digital converter (ADC). After receiving an observation, a program running on the node then processes it and possibly communicates it through the radio transceiver.

A node may incorporate actuators as well that allow it to influence its surroundings. For example, an attached relay could allow a node to enable or disable an external device such as a light. In the case of an actuator, a node can either autonomously control it or act on commands received through the wireless network.

The nodes within a single network deployment may be heterogeneous. One subset of nodes may detect a certain environmental characteristic (e.g., sound), and another subset may detect another (e.g., light). Nodes may also fulfil different communication roles. For example, some nodes may strictly sense the environment and transmit observations, while others may actively participate in relaying messages to the collection point (sink). Multiple sinks may exist in a single network deployment.

The small physical size, low cost, and diverse and ad hoc capabilities of these nodes make them well-suited to a variety of applications. Some of the proposed applications include (a) intrusion detection, classification, and tracking [8], (b) indoor [13] and outdoor [53] environment monitoring, and (c) structural monitoring [98]. For a thorough discussion of possible applications, see references [4, 7]. The ability to deploy these nodes at high densities promises to provide insight into many existing problems at previously impractical scales. For example, a recent (2010) study of soil ecology within urban forests used WSNs to measure both moisture and temperature at previously impossible temporal and spatial levels [87].

The use of low-performance hardware components in a node restricts the software that can run on it. Within limited flash memory, it becomes im-

portant to keep code lean, without the waste of unused functions in large libraries. Given limited RAM, traditional multitasking (i.e., allocating one stack per thread) also becomes impractical. The limited processing power makes it particularly important to consider the computational complexity when implementing algorithms. For example, we derived an algorithm to detect knocking on a door from a real-time QRS complex (heartbeat) detection algorithm *published in 1985* [65]. Using 16-element circular buffers, our implementation uses only around 101 bytes of RAM.

Many of the traditional wired and wireless networking problems persist with WSNs (e.g., medium access control [28]), but the unique characteristics of these small devices have motivated new approaches to well-studied problems. Routing techniques are possibly the largest body of research [5]. Both energy constraints and the desire to scale WSNs to thousands of nodes have motivated new work. The different schemes make different assumptions about the network (e.g., memory to store routes [37] and availability of location data [16]) and focus their optimization effort on different characteristics (e.g., scalability [52] or energy consumption [92]). Other hot topics in wireless sensor networks include localization [54], sleep cycles [44], emulation/simulation [14], and programming abstractions [95].

1.2 Assumptions

We are interested in the development and large-scale deployment of WSNs within dynamic urban environments. With this focus, we depart – to a degree – from the *dominant* assumptions about WSNs and instead seek *appropriate* assumptions for this class of environments.

We consider environments that likely experience high levels of radio-frequency (RF) noise and interference. Noise, RF byproducts from devices, may originate from a number of sources, e.g., fluorescent lighting, elevators, and office equipment [21]. The amplitude of the introduced noise may be non-Gaussian, and it cannot necessarily be modelled analytically. Interference, intentional RF transmissions from devices, may result from devices using overlapping frequencies, e.g., a cordless telephone. Both noise and interference sources may appear and disappear apparently randomly over time, e.g., lights being switched on or a cordless telephone call.

We expect that networks will be deployed in stages, i.e., they will evolve as dictated by either conditions or user needs. In the former case, the nodes may collect network performance measurements over time, e.g., transmission (packet) loss rates, and these statistics may suggest preferred locations for new nodes. In the latter case, we expect that the hardware and software will undoubtedly evolve and be heterogeneous, possibly with vast performance differences. At the same time, however, new devices must be able to communicate with old ones, and the protocols must be backwards compatible.

We realize that extrinsic, application-specific constraints may affect the placement of sensors. For example, a passive-infrared motion sensor detects motion within a conical sensing area and must be pointed at the area of interest. These types of constraints subsequently restrict the placement of the

attached wireless nodes. In some cases, nodes will be placed in suboptimal locations that yield poor-quality wireless links. One way to address this problem is with the addition of relay nodes, i.e., intermediate nodes to repeat transmissions.

Although energy conservation remains a goal, we assume that some nodes will use the abundant power sources available in urban settings. These nodes will typically be those with a degree of placement flexibility, e.g., relay nodes. Nodes with more stringent restrictions on their placement will still tend to be energy-constrained. From an optimization standpoint, this creates an interesting mix of node properties. The addition of further devices powered by small but perpetual energy sources, e.g., solar cells, presents the possibility for a third class. These nodes would offer flexible placement, and at the same time, more communication potential than strictly battery-operated nodes.

We assume that most – not necessarily all – communication in the WSN passes directly between a sensor node and its sink (i.e., with only a single hop). To support such communication, we assume that an environment will have a reasonable number of sinks that are connected to a backbone (often wired) network. Such a backbone network will offer significant performance advantages over the WSN in terms of data carrying capacity and reliability. For areas outside of the range of the sinks, we envision that hard-wired relay nodes placed between sensor nodes and the sinks will connect the network with extra hops where necessary. To support these few multi-hop links, a low-complexity small-footprint routing protocol, e.g., TARP [61], will suffice. For the sensor nodes, this design results in a reduced energy burden (no relaying) and lower latency (typically single-hop communication).

Robustness and reliability are major requirements as we perceive that the data produced by sensors may have direct use in life-critical applications, e.g., healthcare. However, for the sake of our work to date, we are not interested in real-time (RT) criticality. Hence, even though a deliberate effort is made to reduce all relevant latencies, the systems in question are non-RT critical ones.

1.3 Implications

The results presented in this thesis have significant implications for a number of urban WSN development topics. Here, we list selected topics and briefly describe their relationship to our work. Many of them also relate to the aforementioned assumptions.

Accurate emulation

Early in this chapter, we introduced a number of common assumptions that are used in wireless network simulators/emulators, and at the same time, we highlighted the general lack of attention that they give to background noise and interference. Meanwhile, simulators and emulators are frequently used when developing WSNs to test new techniques and evaluate performance. By incorporating our observations on interference patterns into these virtual environments, network researchers will be able to improve the fidelity of their

experiments and have more confidence in their results.

Protocol development

The Smart Condo project introduced a framework and associated protocol suitable for healthcare applications. When we deployed nodes with this protocol, the system achieved our design goals even within an interference-filled environment. Our architecture could serve as a blueprint for other similar WSNs.

Protocols, particularly those for medium access control, can benefit from an awareness of background noise and interference. Nodes can use this knowledge of their environment to either avoid certain channels or time transmissions, for example. Later, in Chapter 6, we present our work on a transmission scheduling technique within the context of a new medium access control (MAC) protocol.

Site surveys

In order to classify channels, nodes must first observe the noise and interference present in their surrounding environment. Rather than simply transmit a handful of packets to measure packet loss rates, nodes can measure the background noise and interference. Through classification, nodes can better conclude why a channel performs poorly, and they can potentially improve the performance through appropriately designed protocols.

1.4 Contributions and outline

As described in the previous section, our work on investigating urban environments applies to a number of areas. In this thesis, we make the following contributions:

1. We present a well-defined urban WSN application named the Smart Condo (Chapter 2). With this description, we show its overall architecture with its associated components and protocols. Our design addresses each of our assumptions for urban environments.
2. We describe the development of software for the Smart Condo application, along with the tools that we used (Chapter 3).
3. We present a technique for sampling RSSI values at a high rate while using low-end hardware (Chapter 4). While other researchers have sampled channels using WSN hardware, we do so at much higher rates.
4. We explore the time-series behaviour of RSSI traces and identify a number of common patterns (Chapter 4). These patterns provide a starting point for further work on recognizing specific patterns.
5. We investigate using lower sampling rates to obtain feature vectors still suitable for classification (Chapter 5). This work is made possible by the high sampling rate that we achieved using WSN hardware.
6. We describe a classifier that fits within the strict resource requirements of low-end WSN nodes (Chapter 5).

7. We implement the described classifier and a pattern-aware medium access control technique, and we show their combined ability to improve network performance (Chapter 6).

In Chapter 7, we summarize our contributions in greater detail and suggest some future research opportunities.

Chapter 2

The *Smart Condo* Project

With a strong desire to develop and deploy an *urban* wireless sensor network (WSN), we joined a multifaceted interdisciplinary research project named the *Smart Condo* in the summer of 2008 [12, 13, 85]. The team, which included students and faculty from Computing Science (CS), Occupational Therapy (OT), Industrial Design (ID), and Pharmacy, endeavoured to create an accessible and supportive living environment. We planned to use locally-developed WSN technology to form the basis for the environment's support system.

2.1 Overview

In one facet of the project, which primarily spanned from September to December 2008, classes of undergraduate students from OT, ID, and pharmacy received the blueprint for a room located within the Telus Centre for Professional Development at the University of Alberta (Figure 2.1). Following universal design principles [20] and with the goal of producing a barrier-free and accessible environment, they divided the open space into rooms and designed furnishings for those rooms. As part of this process, they generated computer renderings of their vision and built foam core models that they placed within the physical space (Figure 2.2).

In another facet, we searched for an appropriate way to incorporate WSN technology into this *urban* environment, and before long, we found our application. As baby boomers continue to age and life expectancies increase, the elderly are placing an increasing strain on the Canadian healthcare system. This fact has led governments, private companies, and researchers to actively search for ways to enable the elderly to live independently – at home – for longer. The elderly often share the same goal, too: they want to remain in their homes rather than transition to assisted-living facilities. If they are to remain in their homes, however, they need support. Although these individuals are, by and large, able to live independently, they are still susceptible to harmful incidents related to diverse physical and cognitive impairments. Our focus is thus designing a supportive environment that includes monitoring functionality to detect concerning behaviour and alert caregivers (healthcare professionals or family) when appropriate.

Although home environments initially motivated our work, the resulting

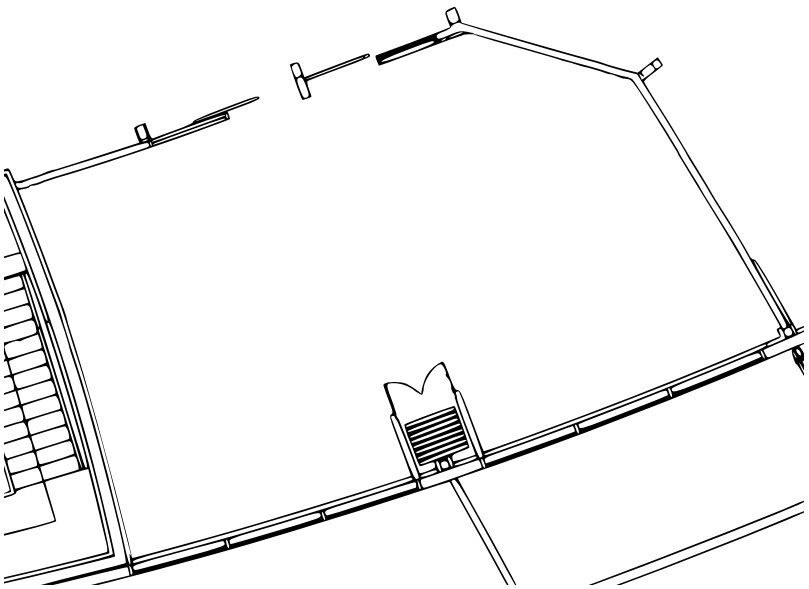


Figure 2.1: The *Smart Condo's* 79-square-metre space within Telus Centre. The two sliding doors at the top of the figure have now been replaced by two swinging doors. The accordion shape at the bottom of the figure can divide the space into two rooms and remains unused in the final design. The unit's six large windows are on the south wall.

Figure 2.2: A rendering of the kitchen (left) and corresponding foam core models placed within the Smart Condo (right). Foam core consists of polystyrene foam (like foam drinking cups) sandwiched between pieces of thin cardboard.

technology transfers easily to other environments. For example, seniors living in an enhanced-living or an assisted-living community could also benefit. In fact, these settings would have the advantage of on-site support staff to respond promptly to system-generated alerts.

As we began to develop this system, we considered questions that included:

1. Why are we monitoring the environment, and given that motivation, what would we like to monitor?

2. What sensors are suitable for monitoring these aspects of the environment?
3. How can data from multiple sources be combined to infer clinically relevant information about patients?
4. How can the inferred information be effectively communicated to patients and their caregivers (health professionals and family members)?

Our answers to these questions, which we outline in the following subsections, helped shape our system.

2.1.1 Objective of monitoring

Our primary goal is to detect behaviour of potential concern within independent-living environments and subsequently alert caregivers, and concerning behaviour can take any number of forms. For elderly patients generally, it might be an usually high number of bathroom visits during the night, waking up at an unusually late time, or failing to maintain regular meal schedules. For a dementia patient, it might be trying to leave the home. These situations suggest a few aspects worth monitoring.

A patient's location within the home might loosely suggest a certain activity. For example, a patient in the bedroom might suggest sleeping and a patient in the kitchen might suggest eating. For that reason, we endeavour to locate the patient within the home.

Given our goal to detect concerning behaviour, we can then monitor other aspects that will help us refine the initial inferences. Particularly for dementia patients, it is relevant to monitor doors to detect wandering. To infer eating, it is relevant to monitor the use of appliances in the kitchen (the fridge or microwave) or access to cupboards. To infer activity levels, it is relevant to monitor the use of chairs and the bed.

2.1.2 Selection of sensors

With all of the monitoring, we have an overall goal of doing it *non-intrusively*. In our use of the term *non-intrusive*, we consider both (a) physical intrusion, e.g., requiring the patient to wear a device and (b) privacy intrusion, e.g., observing the patient in excessive detail. In the first case, a patient living independently could easily forget to wear the required device, and even if the patient were to remember it, a device could potentially restrict movement or be otherwise uncomfortable. We can collect a large amount of data *without* wearable devices, so we elect to use only environmental observations. In the second case, certain sensors intrude on privacy more than others, e.g., researchers have found resistance to video cameras in private living environments [27]. In our approach, we help preserve the patient's privacy by collecting only very coarse data without the use of video cameras and microphones.

To collect these environmental data, our design uses

- passive-infrared motion sensors to detect the occupant's movement (Panasonic AMN43121 and AMN44121),
- magnetic reed switches to monitor the state of doors (Hamlin 59140-010),

- tactile pressure sensors to detect the occupant sitting on chairs (Tekscan FlexiForce A201), and
- accelerometers to detect vibration (STMicroelectronics LIS302DL).

We also investigated the incorporation of (a) a bed occupancy sensor to detect laying on the bed (S4 Sensors BEDsensor Pro) and (b) electric current sensors to recognize the use of appliances. Our most abundant sensor is the passive-infrared motion sensor, and it only provides very coarse data indicating whether motion occurred within its conical sensing region.

2.1.3 Necessity for data fusion

A key aspect of monitoring the environment is identifying the location of the occupant, and our passive-infrared (PIR) motion sensors have this explicit purpose. Unfortunately, their wide-angle view and simple binary output allow for only very coarse localization. To improve their precision, sensor readings from multiple sources must be considered collectively.

In some cases, we consider readings from multiple motion sensors, which we can strategically place to introduce conveniently overlapping sensing regions. For example, consider two motion sensors (A and B) observing a kitchen with some overlap. These two sensors provide three possible localization regions:

1. output from only A implies motion within the region only seen by A,
2. output from only B implies motion within the region only seen by B, and
3. simultaneous output from both A and B implies motion within the overlap (intersection) of their sensing regions.

In other cases, we combine readings from motion sensors with non-motion sensors. For example, a motion sensor might observe a kitchen; a switch, the microwave door. If both these sensors fire together, their readings suggest that the occupant is standing within the kitchen in front of the microwave.

To accomplish such processing, software must have access to all of an environment's observations. For this reason, all nodes transmit their observations through a WSN collection point (sink) to a server that stores them in a database. Software running on a server can query this database and consider all of the observations collectively rather than individually.

To help strategically place sensors, a colleague (Jianzhao Huang) developed the program named Visual Manager (Figure 2.3). Within it, we can place and orient the various types of sensors on a blueprint of a space. In this particular example (Figure 2.3), the coverage of the space is particularly poor given environmental constraints. Our prototyped space lacks walls between rooms, so to prevent unnatural overlap between sensors, we were forced to focus them on the perimeter walls.

After placing all of the nodes within Visual Manager, the software can analyze the layout and determine the overlap between sensing regions. Finally, it can produce the structured query language (SQL) necessary to populate a database table that describes both single sensors that cover isolated regions and combinations of sensors that cover more complex intersections of regions. We used this software when deploying the Smart Condo's WSN.

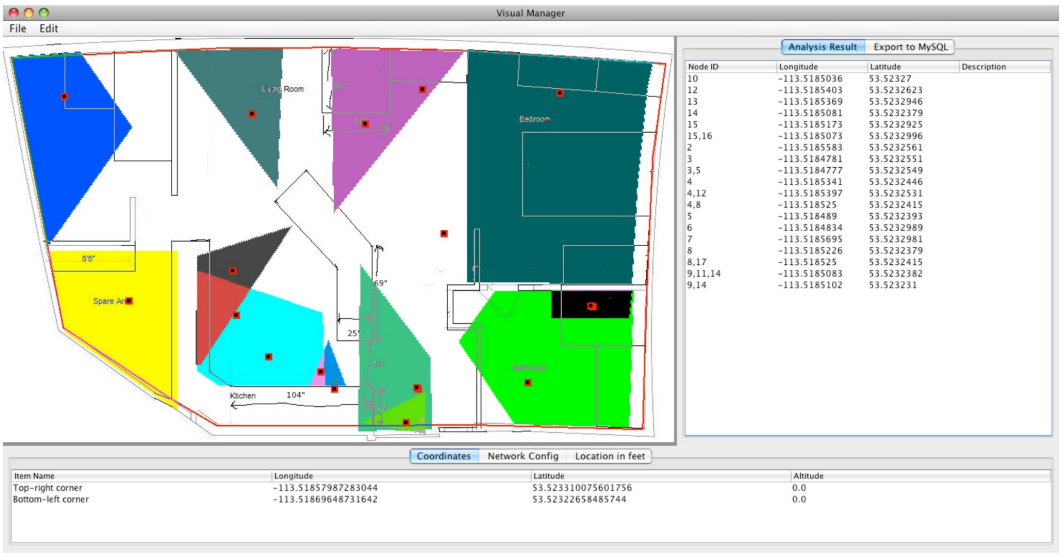


Figure 2.3: We use the program Visual Manager to describe sensor locations and visualize their coverage within the Smart Condo.

As a stream of observations arrives at a server, stream-mining software can infer the occupant's location using the just-mentioned region table. By buffering incoming observations to account for slight delays, it looks at windows of time and attempts to identify the most likely state during each window. Both the individual observations and the inferred locations are archived within a database for visualization purposes.

2.1.4 Communication to caregivers

Another one of our colleagues, David Chodos, focused on communicating the information to caregivers, and he built a particularly useful view using a virtual world, currently, Second Life (SL) (Figure 2.4). Within the virtual world, a model of the Smart Condo represents the physical space, and within that space, it is possible to view a patient's activity realistically and intuitively. At the same time, this display is relatively non-intrusive compared to a video camera.

If and when this project evolves into a mature and final product, caregivers would not be expected to constantly watch the virtual-world display, but instead, they would use it to explore time segments of interest. These time segments could be identified by data-mining methods that inspect the archive of recorded sensor readings to extract patterns and recognize trends associated with clinically significant symptoms and also raise alarms (e.g., [51]). In addition to alarms, graphs could provide a concise summary of an occupant's behaviour (e.g., [64]).

Figure 2.4: Within Second Life, it is possible to view the Smart Condo and the occupant's approximate location within the space.

2.2 Proposed architecture

We developed a three-tier architecture for the Smart Condo system (Figure 2.5). The central resource is a database (SensorDB), which constantly receives new (raw) data from the connected WSN. Server-side scripts, the stream processing and location engine on the server crescer (Figure 2.5), further process the data to infer higher-order information, such as an individual's location given multiple instantaneous motion sensor readings, and store their inferences in SensorDB.

The upper half of Figure 2.5 shows the WSN-specific and data generation components. Although the data originate in the WSN, they pass through a WSN collection point (sink) to the operations support system (OSS), which stores them in the database. The sink is an on-site computer that bridges the WSN with the Internet, and the OSS, a server-based component, monitors the state of the network and provides network data to other components as a data stream.

On the other side of SensorDB, the HTTP/PHP server (hypatia) exposes a set of application programming interfaces (APIs) based on representational state transfer (REST) to one or more client applications. The client most relevant to this work is based on SL. Within it, a user can request (a) a visualization for a certain period of time (e.g., a 10-minute span 4 hours ago from when the patient was preparing lunch) or (b) a live display of the environment's current state. Given an SL request, the server queries the database and returns the result to SL as XML data. The visualization engine then displays the retrieved data as an avatar walking around the space and interacting with objects (e.g., the microwave, doors, or chairs). A second client, a 2D GIS component named

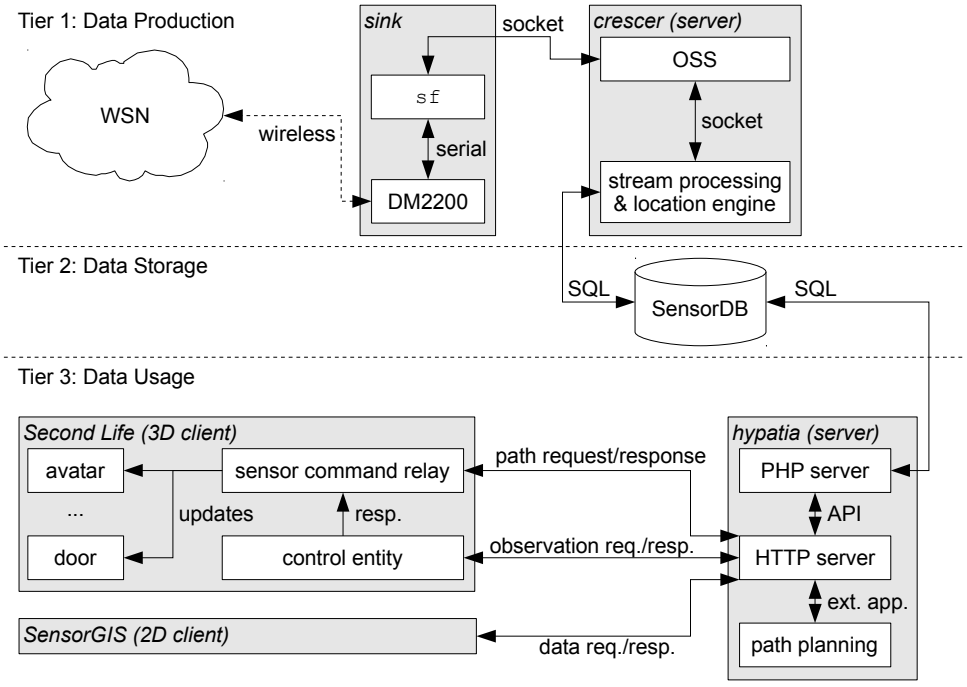


Figure 2.5: The Smart Condo's three-tier architecture.

SensorGIS, can show data as points on a map. Thus, sensor data are decoupled from their representation to the user, allowing for many different ways of visualizing the same sensor data.

2.3 WSN design

Prior to developing software for the Smart Condo, we identified some goals for our design, looked at the capabilities of our hardware, chose an appropriate network topology, developed a protocol for the network, and developed a scheme to flexibly configure nodes for their attached hardware.

2.3.1 Goals

In designing the Smart Condo WSN, we had several key goals in mind. These goals loosely match many of our initial assumptions in Section 1.2. They include (a) reliability, (b) responsiveness, (c) energy conservation, and (d) extensibility. In the following paragraphs, we describe further details for each.

First and foremost, our highest priority in a healthcare environment is reliability. Many of the sensors in our network, such as a switch on a cupboard door, produce data of little consequence. If a message is lost, it has no impact on the occupant. Given a different location, however, that same switch could produce critical data: consider the importance of a switch installed on the exterior door of a dementia patient's home. Now, the same switch produces data

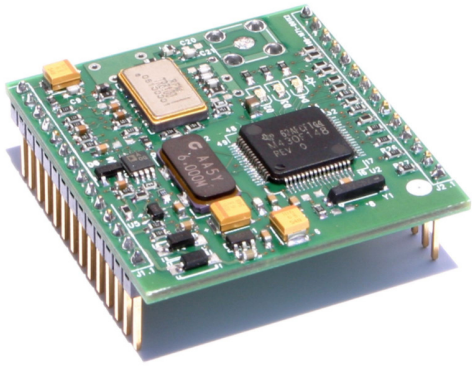


Figure 2.6: An RF Monolithics DM2200 wireless node.

essential to detecting potentially dangerous wandering; moreover, an alert generated based on an observation from this node would require immediate action from a caregiver.

A parallel goal to reliability is responsiveness. If the occupant calls for help, for example, we want the message to reach help in a matter of seconds so that assistance can arrive as quickly as possible. We must be cognisant of responsiveness since it has the potential to conflict with our next goal, energy conservation.

While paying the most attention to the above goals, we always consider energy consumption. We aimed to produce a system that could be easily retrofitted in homes, and as such, we opted for fully-wireless sensor nodes (i.e., no dependence on a wired power source). As soon as we made this decision, energy conservation became key. In selecting sensors for nodes, we aimed to find the most energy efficient ones available. In terms of communication, we wanted to minimize communication while maintaining a certain and flexible level of responsiveness.

One of the final features that we wanted was extensibility. As time passes, we will certainly want to add new sensors to the network with new capabilities. For that reason, whatever protocol we use should be flexible enough to allow for these additions while remaining backwards compatible.

2.3.2 Hardware capabilities

We began the project with two hundred DM2200 wireless nodes from RF Monolithics, Inc. that were used in earlier projects (Figure 2.6). As described in the introduction, these boards feature the MSP430F148 microcontroller (5 MHz, 2 KB of RAM, and 48 KB of flash memory) and TR8100 radio transceiver (916.5 MHz, 9600 bps). We also already had three Metrix Mark II outdoor wireless kits from Metrix Communication LLC (for more information, see [56]). These are very flexible platforms (233 MHz, 64 MB of RAM, and 64 MB of flash memory) that can bridge a WSN (via USB) with the Internet (via Ethernet or its two 802.11 interfaces). With this given hardware, we then had much flexibility in designing the network, selecting sensors, and writing software.

In September 2007, prior to our work on the Smart Condo, we performed a number of experiments with the DM2200 wireless nodes in both indoor and outdoor environments. In each environment, we fixed the location of the transmitter and varied the location of the receiver. The receiver controlled the experiment, and for each location, it would wirelessly request that the transmitter begin sending a number of packets, each containing a predetermined sequence of bytes.

For all DM2200 transmissions (both regular packets and the test sequence), the node converts each 4-bit nibble into a 6-bit symbol immediately prior to transmission. This conversion prevents a node from physically transmitting a long sequence of 0s or 1s, which could cause the receiver to lose synchronization with the transmitter. All of the symbols that it uses are DC-balanced, i.e., they have the same number of 0s as 1s.

In the driver for the DM2200's radio transceiver, the arrival of each symbol triggers a conversion to the corresponding nibble. When the receive process encounters an error, it immediately aborts the whole packet. For our purposes, this behaviour is undesirable because it would only allow us to record one error per packet. To overcome this limitation, we modified the driver so that after observing a start symbol, it processes every symbol for the length of the predetermined sequence regardless of individual symbol failures.

In order to record results at the receiving DM2200, we connected the node to an interface module, the IM2200. This board adapts the DM2200's serial interface to USB, and we connected it to a notebook computer. For each detected start symbol, the receiver wrote to the serial port

- the number of correctly received symbols,
- the number of incorrectly received symbols, and
- the received signal strength indicator (RSSI) value.

In these experiments, we observed better than expected error rates in both environments.

Outdoors, on the west side of the University of Alberta campus at 116 Street near 87 Avenue, nodes could successfully communicate with line-of-sight communication at distances of up to 400 metres (Figure 2.7). As the distance increased between the two nodes, the RSSI smoothly decreased as expected. For the symbol error rate, the curve was not nearly as smooth, which is likely an artifact of changing the receiver's environment with each change in the distance.

We then attempted some indoor experiments by varying the propagation distance within a 20-floor 321-unit apartment building. We found that nodes separated by nine concrete floors (a distance of just over 23 m) could transmit *symbols* with a 99.37554% success rate (Figure 2.8). For this largest tested distance, extrapolating from the SER to a modestly sized 16-byte packet suggests an equivalent *packet* loss rate of around 20%. Although this error rate is now quite significant, distances between nodes in the Smart Condo are much smaller, and in many cases, there are no obstructions.

The Smart Condo is approximately 79 m^2 and roughly 11 m by 7 m. This floor-space is quite typical of one-bedroom apartments or condominiums in our area. The maximum propagation distance for nodes placed on the unit's ex-

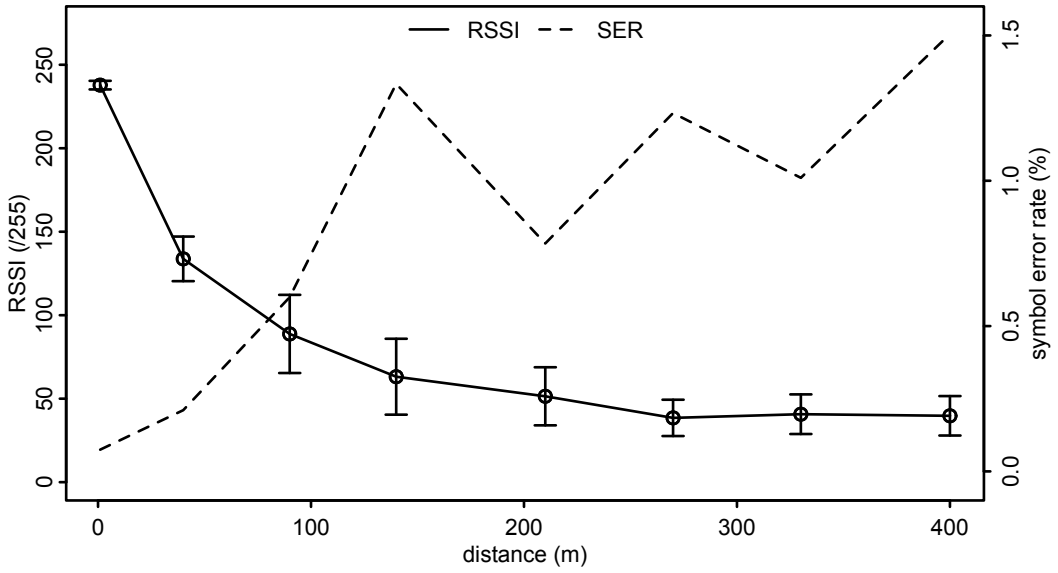


Figure 2.7: Results from outdoor line-of-sight experiments with the DM2200 wireless node. It shows both the average RSSI values (with standard deviations) and the symbol error rates that we observed.

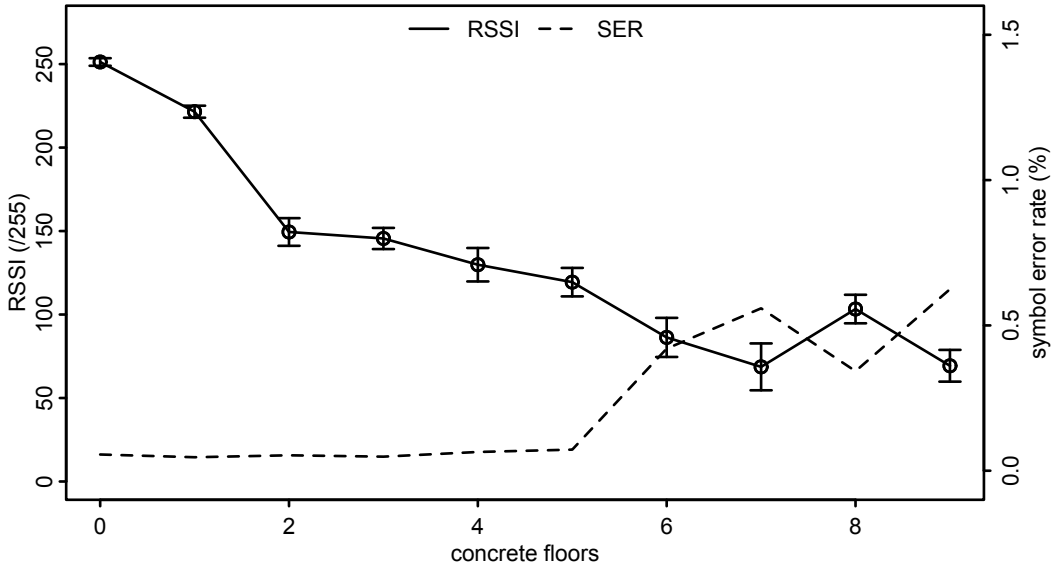


Figure 2.8: Results from indoor experiments with the DM2200 wireless node. It shows both the average RSSI values (with standard deviations) and symbol error rates that we observed.

tremities is 13 m. The actual maximum propagation distance in our deployed network is only 8 m.

Given our outdoor and indoor results, the Smart Condo distances are well within the communication range of a DM2200 wireless node. Consequently, we had no hesitation in adopting these nodes for our prototype environment.

2.3.3 Network

Given the adequate communication range of the DM2200, we decided to use a single sink node (collection point) and support only single-hop communication. There are several advantages to single-hop communication in this environment:

- Nodes can communicate observations at any time to the sink. Given only a single hop, they do not rely on other nodes in the network. This results in flexible *responsiveness* that can be determined at the transmitting node.
- When not transmitting, nodes are free to sleep for extended periods of time. They do not need to periodically help relay packets. This results in significant *energy conservation* at the nodes.

With these advantages and our observations about the range of our nodes and the size of our environment, we had no hesitation in restricting our network to a single hop.

A number of components collectively act as the sink. Sensor nodes transmit their observations to a centrally located DM2200, which is an essential component because it communicates using the WSN's proprietary protocol. As with our indoor and outdoor propagation experiments, we connect the DM2200 to an IM2200 that provides a USB interface. Using USB, we connect the IM2200 to a Metrix Mark II that provides an Ethernet interface. Software runs on both the DM2200 and Metrix that effectively bridges the WSN to a host over the Internet. This remote host, a server running in the Department of Computing Science, stores all of the measurements in a database. Although our current design only uses single-hop *wireless* communication, the overall path from a sensor node to the server is multi-hop by virtue of the wired connections.

Recall that the highest priority in our design is *reliability*. To this end, our nodes use a stop-and-wait automatic repeat request (ARQ) scheme. In such an approach, all received packets result in either an acknowledgement (ACK) or a negative acknowledgement (NACK). The transmitting node will retransmit a packet, up to a certain limit, until it receives an ACK. When it receives neither an ACK nor a NACK, it uses a timeout to initiate the retransmission. If it does not receive an ACK before reaching the specific retransmission limit, it begins searching for alternate sink nodes.

Furthermore, we adopt a *fate sharing* approach [24] and insist on end-to-end reliability at the application layer. That is, "information about transport level synchronization is stored in the host which is attached to the net," and in our case, this information is stored at the sensor node. When nodes transmit observations, they do not remove them from their transmit queue until they have received confirmation (i.e., an acknowledgement) that the data reached

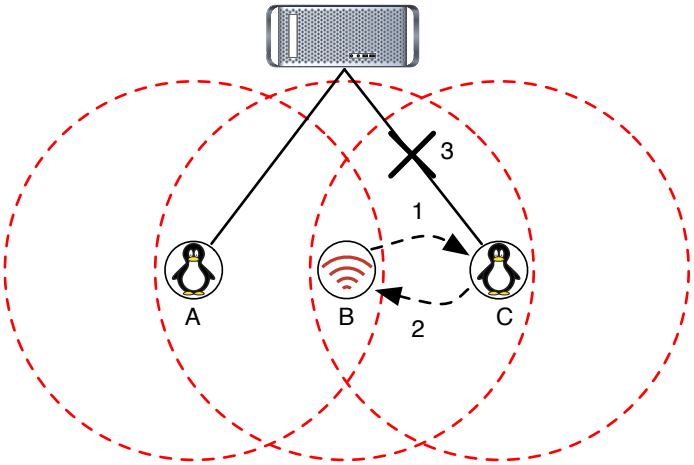


Figure 2.9: Suppose that node B transmits its observation to node C and receives a custody acknowledgement (1 and 2, respectively). If node C's link to the database fails (3), the observation is stuck at node C because it has no way to communicate with the redundant sink (A). Moreover, since node B handed off the observation and cleared its buffer, it will not retransmit the packet to the redundant sink (A).

its endpoint (i.e., the database). Essentially, we have placed the responsibility on the sensor nodes for ensuring the delivery of their observations. This approach helps protect against intermediate failures, and it gives us the option of later introducing redundant sink nodes that could use higher-cost redundant connections to the Internet (e.g., general packet radio service, GPRS).

An alternative approach to addressing reliability is known as *custody transfer* [31]. In this approach, when a relay node accepts a message, it promises not to delete it until the message is passed off to another node via custody transfer or it reaches its destination. In this scheme, successful handoffs do not provide end-to-end reliability, but instead delegate the responsibility for end-to-end delivery to another node. This allows nodes to clear buffers quicker since the round-trip time to complete a custody transfer is less than the end-to-end round trip.

To see the potential problem with custody handoffs, consider Figure 2.9. The problem arises given multiple sink nodes, while at the same time, not providing them a means to communicate with each other. Depending on the timing of a failure, observations could end up being stuck at a sink node.

2.3.4 Communication protocol

To address our goal for *extensibility*, we carefully designed our network's communication protocol. At the network layer, the Tiny Ad hoc Routing Protocol (TARP) [61] includes the following fields in each packet:

- *the type of message (1 byte)*,
- *a network-layer sequence number (1 byte)*,

- *the node ID of the transmitter (2 bytes)*,
- *the node ID of the receiver (2 bytes)*,
- the number of hops travelled so far (1 byte), and
- the number of hops previously travelled by packets in the reverse direction (1 byte).

We explicitly make use of the emphasized fields. Furthermore, we define four types of TARP messages for the Smart Condo application: `msg_poweron`, `msg_announce`, `msg_command`, and `msg_ack`.

When a node is new to the network or fails to receive multiple acknowledgements through its associated sink, it begins to generate power-on packets (type `msg_poweron`). These packets consist of only the TARP headers and are broadcast into the environment. Generally, sensor nodes are sleeping and do not receive these messages, although they may occasionally overhear and ignore them. When received at a sink and forwarded to the server, the OSS running on that server generates a `msg_announce` packet that may include details relevant for sink selection (in the case of multiple sinks). Such details may include a quality rating for its link or the expense of the link.

Further sensor node to OSS communication occurs using our general-purpose `msg_command` packets. They build on the basic TARP headers by including an application-layer sequence number (1 byte) and length (1 byte), which allow for a variable number of subsequent key/value pairs. Table 2.1 summarizes the currently-implemented key/value pairs.

In the key, the three most significant bits indicate the length of the following value. To support values that are seven bytes and larger, we reserve 111 to indicate that a 1-byte length immediately follows the key. The reasoning behind this approach originates in the difficulty in updating deployed sensor nodes. As the software evolves, we will add new key/value pairs to the protocol and can easily update the sink node's software. In order for older nodes to continue to function given these additions, they need to be able to parse packets and ignore what they cannot understand. The encoded length allows for this type of processing.

The final message type, `msg_ack`, includes the same application-layer sequence number, length, and support for key/value pairs found in our general-purpose command type. After the OSS receives observations in a `msg_command` packet, the acknowledgement always contains the `BIN_ACK` key/value pair. The OSS may add other key/value pairs to the acknowledgements as necessary such as `GLOBAL_TIMESTAMP` to synchronize the sensor node clocks.

2.3.5 Behaviour overview

When we first power on a node in the field, it broadcasts its existence, including its unique node identifier. It collects responses through the sinks, and after a timeout, associates with the *best* sink. It currently defines *best* based solely on the highest RSSI value that it receives. Future work may add additional sink nodes that use redundant, but more expensive, Internet connections. After we make this addition, our software will need to consider extra information when determining the best sink.

Table 2.1: Key/value pairs supported in the implemented Smart Condo software. The three most significant bits of the key indicate the length of the value that follows it.

Mnemonic	Key	Description
CLEAR_SINK	000 00010	request sensor node to deselect the current sink
REQ_CONFIG	000 00011	to sensor, request sensor node to request its configuration; to sink, request the current configuration
BIN_ACK	001 00010	acknowledge n binary events
TS_SHIFT	001 00011	shift reported time-stamps by n bits
COL_SINK_DELAY	001 00100	delay t seconds when awaiting sink responses
COL_SINK_RETRIES	001 00101	retry n times when awaiting sink announcements
ACK_DELAY	001 00110	delay t seconds when awaiting an acknowledgement
ACK_RETRIES	001 00111	retry n times when awaiting an acknowledgement
MIN_TX_DELAY	010 00000	delay t seconds (minimum) between reporting observations
MAX_TX_DELAY	010 00001	delay t seconds (maximum) between reporting observations
COL_SINK_SLEEP	010 00010	delay t seconds before retrying to find a sink node
GLOBAL_TIMESTAMP	100 00000	to sensor, set global time-stamp to n; to sink, base for relative time-stamped observations
ANN_BIN_SENSORS	111 00000	announce the connected binary sensor at the node: 1B: len, [1B: type]...
BIN_CONFIG	111 00010	the binary configuration for a sensor node: 1B: len, [3b: sensor idx, 5b: bin. flags, 2B: period/debounce]...
BIN_OBSERV	111 00100	observations for binary events: 1B: len, [1B: ts, 4b: sensor index, 4b: event type]...

As part of associating itself with a sink, the node describes its attached sensors and its software version. The OSS then replies with both general (e.g., the sleep cycle) and sensor-specific (e.g., debounce timeouts) configuration parameters. After processing that response, the node then begins to monitor its attached sensors and then goes to sleep.

Nodes wake up and transmit when either (a) they have an observation to report to the OSS or (b) a timer expires because no observations were made within a configurable amount of time. The latter case informs the server that the node is functioning (but had no observations). Recall that (a) nodes spend most of their time sleeping in a state where they cannot receive packets and (b) all transmission from sensor nodes to the OSS are acknowledged by the OSS. These acknowledgements, with their support for key/value pairs, provide an opportunity for OSS-to-sensor communication.

Sensor nodes use a rather clever technique for time-stamping observations that only requires one byte per observation. Nodes will periodically receive a global time-stamp from the OSS and store it locally. When a node caches its first observation, it

1. determines the time elapsed between the global time-stamp and the current time,
2. increments its locally stored global time-stamp by that elapsed time, and
3. uses the global time-stamp henceforth as a reference point for the current and any subsequent observations in the cache.

Based on the reference point, the one-byte time-stamp field allows nodes to stamp observations for up to 255 seconds with one-second precision. If the node adds another observation to the queue after 255 seconds, it shifts the time-stamp for each entry in the queue one bit to the right. Now, the queue supports time-stamps from 0 to 510 seconds with two-second precision. The node will continue to make further shifts as necessary, and thus it can queue many observations in its limited memory. After a node successfully transmits all of its observations, it returns to updating its global time-stamp as it receives new values from the OSS.

2.4 Summary

Given few reported experiences using *urban* WSNs, we endeavoured to create one ourselves. In our search for a practical application, we joined an interdisciplinary research project named the Smart Condo. Through discussions with our colleagues, we identified a significant and well-defined application. The Canadian healthcare system is straining given an increasing elderly population, and various stakeholders are searching for ways to keep them out of hospitals and living independently in their homes for as long as possible. For them to live in their homes, however, they need support, so we focused on developing a WSN for monitoring independent living environments. After identifying and defining the problem, we designed our system.

We developed a three-tier architecture centred around a database, SensorDB. On one side of the database, a WSN and supporting components feed

it with data and make inferences about the environment. On the other side, components can access data through well-defined APIs to support a number of views.

We then focused on the development of the WSN. We optimized the network for a number of goals including (a) reliability, (b) responsiveness, (c) energy conservation, and (d) extensibility. With these goals in mind, we reviewed prior evaluations of our existing hardware in both indoor and outdoor environments, and in both cases, their communication range exceeded our requirements. We then decided to use a single-hop network, and we developed a communication protocol and node behaviour around this topology.

In the next chapter, we discuss the software development. Moreover, we will present results on whether our chosen design achieves its objectives.

Chapter 3

WSN Development and the *Smart Condo* Deployment

At this point, we have defined an urban wireless sensor network (WSN) named the Smart Condo. We have determined its overall objective, architecture, and communication protocol. In this chapter, we focus on our experience developing its software and deploying it.

We begin by describing the software components that constitute our development environment: (a) an operating system (OS), (b) a simulator, and (c) an implementation of the OS's application programming interface (API) for that simulator. We often refer to the combination of the latter two components as an emulator because it emulates an application at the level of the OS's API.

With these available components, we developed our software using the emulator, rather than the hardware, with the knowledge that we could later recompile the same code for the hardware. To improve the emulator's fidelity, we configured it to consider the capabilities of our nodes. Once our emulated application was relatively stable, we transitioned it to the hardware and addressed a specific challenge that we encountered in the process (addressed later).

Finally, we discuss our experience with the complete *Smart Condo* system and specifically address whether it met the project's initial objective of real-time non-intrusive visualization of an independent-living environment. At that point, we highlight how this system inspired the research questions that we address later in this thesis.

3.1 Development environment

Developing software for a WSN node is quite different than developing it for a personal computer. Nodes are not self-sufficient: software must be developed on a personal computer and then programmed into a node's non-volatile memory. This approach statically links the OS with the application *prior* to programming the hardware. In this section, we introduce our OS and its related tools that simplify software development within these constraints.

3.1.1 PicOS

PicOS [2] is a lightweight operating system that supports the organization of a reactive application's multiple activities on a microcontroller with limited resources. It implements a flavor of multitasking within a very small amount of RAM and provides simple, orthogonal, and expressive tools for event-driven input and output (I/O) and inter-process communication (IPC).

A PicOS application, which may be logically split over multiple source files, consists of (a) global data structures and variables, (b) finite state machines (FSMs), and (c) functions that may be called from within the FSMs. The execution of an application begins at the `root` FSM, which is akin to the `main` function of a C/C++ program. Here is the customary "Hello World" program written for PicOS:

```
#include "sysio.h"
#include "ser.h"

fsm root {
  state WRITE_MSG:
    ser_out (WRITE_MSG, "hello, world\r\n");
    finish;
}
```

This complete PicOS application consists of one FSM, `root`, which contains a single state `WRITE_MSG`. The first line within this state writes the message to the device's serial port (universal asynchronous receiver/transmitter, UART) using the library function `ser_out`, which has been made available by including the header file `ser.h`. Since a busy serial port would cause this function call to block, its first argument is the state in which it should resume when the port becomes available. If required by the application, PicOS supports a number of ways to interrupt a blocked FSM, e.g., timeouts and inter-process communication (IPC). While one FSM remains blocked, other FSMs can continue to operate independently.

This FSM paradigm is fundamental in PicOS's support for multitasking. When execution reaches the end of a state, the FSM releases the CPU back to the operating system. At that point, the OS scheduler can allocate it to a ready FSM. If none are ready, the OS can transition it into a sleep state where it consumes less energy and can remain until an event occurs. In this paradigm, one FSM cannot interrupt another, so a responsive application tends to use FSM's with relatively short states.

Since WSN applications are predominantly reactive, i.e., not CPU bound, it is quite natural to express them using PicOS's FSM paradigm. The format is especially useful and natural for applications that respond to possibly complicated configurations of events, as once the application identifies the events of interest, the underlying operating system handles much of the complexity. Moreover, this representation stimulates clarity – to the extent that well-written and appropriately partitioned blocks of code often appear self-documenting. At the same time, it is not restrictive: developers can write and incorporate standard C functions within a PicOS application. In doing so, however, they must be cognizant of their function's run time and its potential impact on the overall

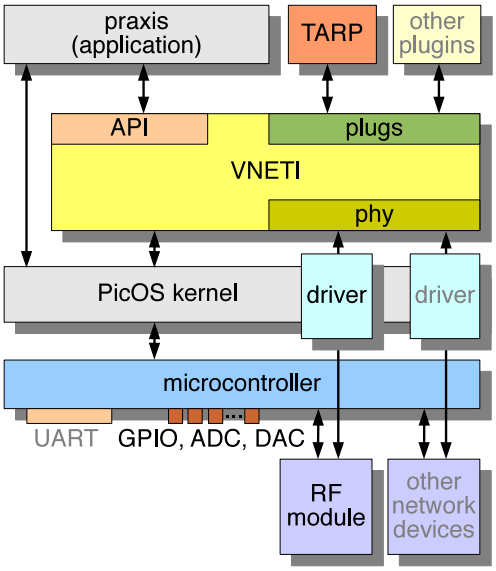


Figure 3.1: The relationships between the various PicOS components.

system responsiveness.

The operating system provides a variety of system calls and library functions, many of which can be selectively included by using a handful of header files. The system calls fall in a number of categories that include

- manipulating FSMs: creating, managing, and destroying them;
- allocating memory: dynamically allocating and freeing memory using the familiar `malloc`/`free` operations of C;
- managing power: selecting the different low-power modes available to the microcontroller for duty cycling;
- communicating between processes: providing tools for FSM IPC;
- timing events: implementing delays within FSMs, including both non-busy and precisely-timed busy waits;
- generating random numbers: deriving truly random numbers based on external entropy sources;
- debugging software: printing debug messages, asserting Boolean expressions, and throwing exceptions; and
- monitoring faults: resetting an unresponsive device using a watchdog.

The operating system also includes libraries for interacting with (a) a selection of abstracted sensors via built-in drivers, (b) the general purpose input/output (GPIO) pins, (c) light-emitting diodes (LEDs), (d) the universal asynchronous receiver/transmitter (UART), and (e) radio transceivers.

For the last two hardware components, the UART and radio transceivers, their drivers interact with PicOS through an API provided by the Versatile NETwork Interface (VNETI) (Figure 3.1). In addition to its device driver API, VNETI provides additional APIs for the application and communication protocols. This three-facet mediator makes it relatively easy to redeploy an existing application on new hardware platforms.

3.1.2 SIDE

PicOS inherits its programming paradigm from SMURPH/SIDE [35], which is best summarized as "a programming language for describing reactive systems (e.g., communication networks) and specifying event-driven programs (e.g., network protocols) in a natural and straightforward way" [36]. After writing a program using the SIDE language, the user can compile and link it with the SIDE kernel to produce an executable file. Running this file will then simulate the specified reactive system.

The SIDE simulator has a long history that began in 1986, and until recently, it lacked support for two desirable features: (a) simulating *wireless* communication channels and (b) emulating a PicOS application as its specification of a reactive system. In 2006, it gained the first ability with the introduction of flexible abstractions for describing realistic wireless propagation environments [38]. It also recently gained the second ability with the introduction of a SIDE-based implementation of PicOS's API called the Virtual Underlay Execution Engine (VUE^2). In the next subsection, we focus on VUE^2 and specifically address how it supports descriptions of both devices and their environment.

3.1.3 VUE^2

Our desire to develop software using an emulator is motivated by two factors. First, although an application running on a node can communicate wirelessly, nodes must be individually wired to a computer to be programmed.[1] Given a collection of nodes, simply loading them with an updated application is a time-consuming process, so we want to minimize the number of times that we need to load them. Second, although PicOS provides extensive support for writing reactive multitasking programs, debugging those programs while they run on a node is a tedious process. The form factor of a node (they typically have a surface area of just a few square centimetres) means that they provide very little output about their internal state. Debugging the software on a node typically uses its LEDs and UART extensively, and when the application itself uses the UART, the developer might be restricted to just the LEDs.[2] For that reason, we regularly *emulated* our software while we were developing it.

When (generally) evaluating a protocol using SIDE, i.e., without the concept of a PicOS application, SIDE is arguably a simulator. The code describing the protocol may be written specifically for the evaluation, and outside of SIDE, it has little relevance. With the introduction of VUE^2, however, the SIDE/VUE^2 combination is arguably an emulator. A developer can produce a PicOS application that is ultimately destined for a piece of hardware, and SIDE will emulate it at the level of the PicOS API. It is precisely this ability – writing source code

[1] One of our colleagues investigated the ability to remotely update even these small devices. However, for this project, the nodes were incapable of remote updates.

[2] Using the program `msp430-gdbproxy` and an extended version of `gdb`, it is possible to trace the execution of code running on a node. This approach, however, requires a connection between each node and a personal computer using the JTAG interface. Therefore, although it can provide reasonable insight into the state of a single node, it is quite impractical given a collection of nodes.

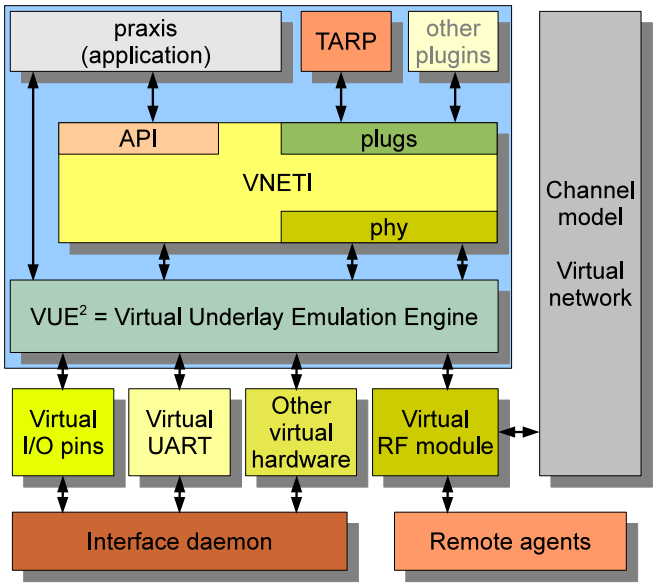


Figure 3.2: The relationship between PicOS components and VUE^2, the Virtual Underlay Execution Engine.

once and targeting it for either the hardware or the emulator during compilation – that makes it easy to incorporate SIDE into the development cycle.

Development using the emulator immediately provides a number of benefits:

- flexible output: SIDE supports additional file streams that can be used for debug messages,
- rapid re-evaluation: SIDE allows for the immediate testing of new code that runs at multiple virtual nodes simply by recompiling the application, and
- global view: SIDE displays the state of all nodes simultaneously to the developer, which can provide insight into the precise timing of a protocol.

In summary, SIDE provides significant support to developers that makes development more efficient.

The feasibility of extending SIDE with VUE^2 was made possible in part because of PicOS's clear division between components and use of APIs (Figure 3.2 shows the SIDE/VUE^2 equivalent to Figure 3.1). Recall that VNETI provides three different APIs: one for the application, one for protocols, and one for device drivers. The clear delineation between these components allows SIDE to use the existing implementation of VNETI (and hence the same application and protocol implementations) while emulating the devices.

In order to execute a PicOS application within VUE^2, the developer must provide VUE^2 with a description of the network, and to a degree, the hardware. In the next section, we describe configuring VUE^2 for our Smart Condo network.

3.2 Configuring the emulator

Generally speaking, an emulator's configuration consists of both hardware- and network-specific properties. The former properties typically affect an application's execution on a node, and the latter, communication between the nodes.

Hardware-specific properties include such things as

- the amount of memory available at a node,
- the rate at which a node can communicate over both its wired and wireless interfaces,
- the number of light-emitting diodes (LEDs) available at a node, and
- the type and number of peripherals that are attached to a node.

A node also has a specific processor speed, but since our applications tend to be reactive, our emulator quite reasonably ignores this detail.

Given that our emulator simulates the wireless channel, a number of factors directly impact the quality of wireless communication. Beyond the channel model being used, these properties include such things as

- the location of a node relative to the others in the network and
- properties of the (virtual) environment in which all of the nodes exist.

To accommodate different environments, simulators typically adjust a channel model parameter.

A high-fidelity emulator must account for these details in order to produce accurate results. In our case, most of this information is provided to the emulator in a VUE^2-specific XML configuration file when it starts.

3.2.1 VUE^2 configuration

When running a VUE^2-based instance of SIDE, a file containing extensible markup language (XML) provides it with hardware- and network-specific parameters. Appendix A (pp. 110-113) lists an example configuration file. VUE^2 supports a wide range of tags and attributes for these files, and in this section, we simply provide an overview of them guided by the sample in the appendix. The interested reader can find a complete description of the XML configuration file within the VUE^2 manual [63].

The number of nodes (and possibly different number of radios) is very important, and these values prominently appear as attributes to the outermost tag `<network>`. In a single configuration file, only one `<network>` tag may exist, and its children provide the hardware- and network-specific parameters.

The hardware-specific attributes appear within the tag `<nodes>`. Typically, an experiment uses many nodes of the same type, so a `<defaults>` section reduces repetition. The tags in this section typically describe such things as

- the amount of memory available at a node (`<memory>`),
- radio parameters (`<radio>`) including the default transmit power level (`<power>`), the number of bits of preamble to transmit (`<preamble>`), the settings for the listen-before-talk medium access control (MAC) protocol (`<lbt>`), and the delay that a transmitter backs off after detecting an occupied channel (`<backoff>`).

- the configuration of light-emitting diodes available at a node (<leds>),
- the configuration of general-purpose input/output (GPIO) pins (<pins>), and
- the settings for the universal asynchronous receiver/transmitter (UART) (<uart>).

A collection of <preinit> tags generally appear at the end of the <defaults> section. These tags provide the values for constants defined within the application. Following the defaults section, the user often provides per-node values for constants and the node's location. In the sample presented in Appendix A, the SENS_0 constant identifies the specific sensor available at a given node.

An XML configuration file also specifies a number of network-specific parameters, with the most prominent being the description of the communication channel (<channel>). Our simulated communication links use a log-normal shadowing model [73, p. 104] (<shadowing>), and the user can specify the parameters to the model using both attributes to the tag and text in its body. Common attributes include (a) the level of background noise (bn) and (b) the number of preamble bits that a receiver must correctly receive in order to synchronize with the signal (syncbits). The body of the tag provides the parameters for the shadowing model's attenuation equation

$$\left[\frac{P_r(d)}{P_x}\right]_{dB} = -10\beta \log\left(\frac{d}{d_0}\right) + \chi(\sigma_{dB}) - L(d_0) \tag{3.1}$$

and subsequently allows the emulator to calculate the received signal level for a given transmit level and distance. Within the sample provided in Appendix A

```
RP(d)/XP [dB] = -10 x 5.1 x log(d/1.0m) + X(1.0) - 33.5
```

VUE^2 ignores non-numeric characters and extracts the initial -10, the loss exponent ($\beta = 5.1$), the reference distance ($d_0 = 1.0$ m), the standard deviation ($\sigma_{dB} = 1.0$) for the log-normal random Gaussian component (χ), and the loss at the reference distance ($L(d_0) = 33.5$).

Beyond the shadowing model, the <channel> section also specifies

- <cutoff>: the signal level at which signals become irrelevant,
- <ber>: a mapping from signal-to-interference ratios to bit error rates,
- <frame>: the number of bits necessary in order to frame a packet,
- <rates>: the supported communication rates,
- <rssi>: the mapping from received signal values to transceiver-reported received signal strength indicator (RSSI) values, and
- <power>: the mapping from abstracted to real transmit power levels.

Given all of these parameters, VUE^2 provides a reasonable amount of flexibility for describing an environment's radio propagation characteristics.

3.3 Challenges with transitioning to the hardware

Using our software tools (PicOS, SIDE/SMURPH, and VUE^2), we developed our wireless sensor network within the virtual environment. We were even able to

interface the virtual network with the external components in our overall architecture (Figure 2.5): VUE^2 provided sockets to which the operations support system (OSS) could connect. We tested our work under a variety of conditions, e.g., differing the propagation distances, varying event arrival rates, and even periodically restarting nodes. When we were satisfied with the stability and performance of our software, we began to transition it to the hardware.

The primary challenge in moving the application from a virtual to real environment was setting node-specific attributes. When using VUE^2, we took advantage of its ability to efficiently define per-node constants with the syntax

`<preinit tag="`*const_name*`" type="`*const_type*`">`*const_value*`</preinit>`

We made such declarations for each node's identifier and type of attached sensors.

Compiling the Smart Condo application for the hardware, however, produces a generic image lacking individual identifiers, and we desired a way to set per-node constants when programming individual nodes. To this end, we carefully placed constant values (tags) within our program to serve as placeholders for the node ID, sensor configuration, and software version. Just prior to actually programming a node, we use an extended version of the `genihex` script to search for these constant identifiers and replace them with node-specific values to create a new derived image. We derive a node-specific image for each node that we deploy.

3.4 Discussion

After deploying the WSN in the Smart Condo, we evaluated its performance with respect to our vision of non-intrusively monitoring an independent living environment. Video cameras provided us with the ground-truth for our evaluation. They allowed us to compare the virtual reality produced by our system against reality.

We concocted a scenario whereby an actor would move around the Smart Condo and perform a number of tasks. These tasks included opening and closing doors, using the microwave, and sitting on furniture. Meanwhile, evaluators in a separate room could watch both the live video feed from the environment and the virtual feed using Second Life. With both displays in front of them, they could judge the fidelity of the virtual display.

In terms of the system's performance, the Second Life display generally matched reality. We observed that the real-time display of the virtual environment would periodically miss events from particular nodes. If we were to go back and replay the period in question, however, the event would occur in the replay. By monitoring the database, we quickly concluded that the problem was not with the virtual world display, and instead, it involved delays within the sensor network. Congestion was not responsible for the delays, so we explored packet loss rates.

Every packet sent from a sensor node to the collection point (sink) includes both a network- and application-layer sequence number. The sink perceives

Table 3.1: A comparison of packets sizes (in bytes) sent from the sink ($n = 151\,161$) and sent from sensor nodes ($n = 151\,163$).

sender	min	mean	max	med	stddev
sink	8.00	15.98	32.00	15.00	1.90
sensor	10.00	15.88	42.00	10.00	7.48

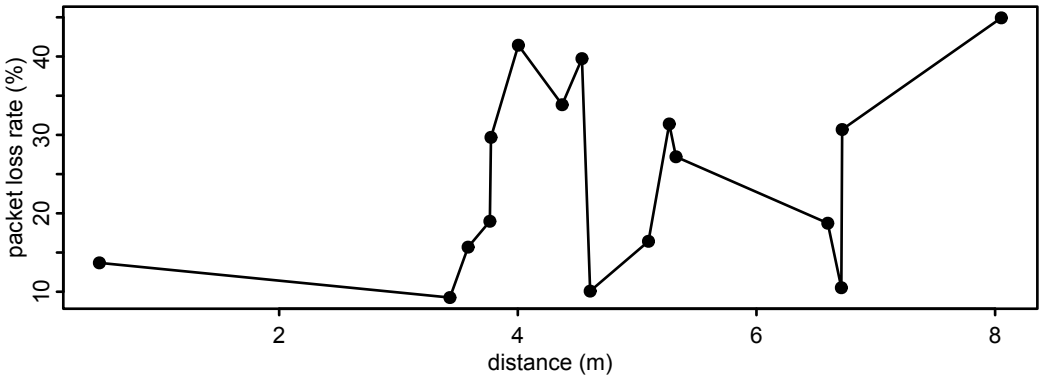


Figure 3.3: Packet losses versus distance.

packet losses using these sequence numbers. A duplicate application-layer sequence number indicates a lost acknowledgement from the sink. A skipped network-layer sequence number indicates a lost packet from the sensor node. For each associated node, the sink records both types of sequence numbers.

Using this technique, differences in the incoming versus outgoing packet sizes may bias the packet error rates. In data collected from the Smart Condo, we observed the packet size statistics in Table 3.1. The packets sent from sensors show much greater variance because nodes regularly report their status in small packets, and during periods of high activity, nodes queue observations to produce large packets and reduce the impact of overheads.

In Figure 3.3, we show the packet loss rate plotted against distance. Compare this figure with the smooth curve in Figure 2.7 that clearly showed correlation between symbol error rates (SERs) and distance. Within the Smart Condo, notice that as the distance increases, the change in the packet loss rate is unpredictable. Statistically, the Smart Condo packet loss rates are not significantly correlated with propagation distance ($n = 16,$ Spearman's $\rho = 0.329,$ p-value $= 0.2105$).

In Figure 3.4, we show the same loss rates spatially, and the larger dot near (8.4, -0.6) represents the sink. The interpolation shown in the graph is only to emphasize the lack of correlation with distance. Our observations are consistent with the extreme variability in reception probability for various distances noted in [97]. As an example of the intricate behaviours at play here, [103] mentions that the packet reception ratio depends on node orientation – even with whip antennas.

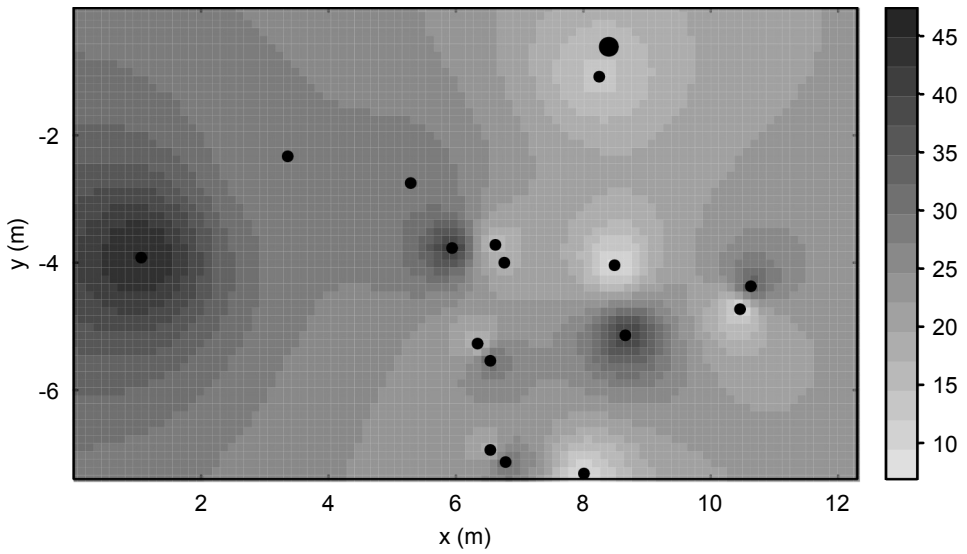


Figure 3.4: Packet losses presented spatially.

After encountering such high loss rates within the Smart Condo, we investigated received signal strength indicator (RSSI) values in the absence of transmissions. Radio transceivers generally output an RSSI value, and it refers to the current signal level that a node observes. The TR8100 transceiver incorporated in the DM2200 does not provide it as a digital value. Instead, a circuit smooths the signal and the result feeds into a 12-bit analog-to-digital converter (ADC) on the MSP430. We measured the RSSI value in three environments: the Smart Condo, the Computing Science Centre at the University of Alberta, and a 1-bedroom apartment in a 321-unit apartment building (Figure 3.5). We discovered that the Smart Condo had the highest received signal levels (in the absence of transmissions) of all three environments. These observations motivate the next chapter's exploration of interference in an urban environment.

3.5 Summary

This chapter provided an overview of our software development environment after Chapter 2 presented the design of the Smart Condo's WSN. As background material, we introduced the PicOS operating system, which inherited its programming paradigm from SIDE, and VUE^2, an implementation of the PicOS API for our wireless simulator named SIDE/SMURPH. This software allowed us to design our software efficiently and provided us with such benefits as (a) flexible output of diagnostic messages from several nodes at once, (b) easy use of the GNU Debugger (GDB), and (c) quick turn-around times between a compilation and an experiment involving the application.

After developing our application in the virtual environment, we moved it to the real hardware. The primary problem that we encountered was the need to set per-node constants prior to programming nodes. Using a technique already used for setting host IDs, we expanded `genihex` to allow us to set a node's

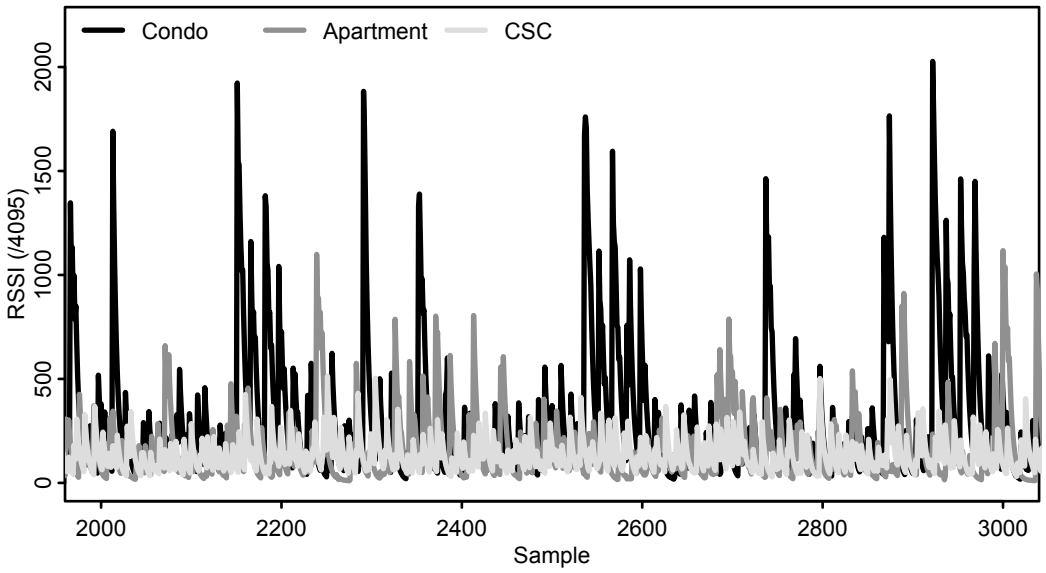


Figure 3.5: Interference observed across three environments.

sensor types and version time-stamps when programming nodes.

We then deployed the WSN and evaluated it in terms of unobtrusively monitoring independent living environments. Although the software generally worked as expected, we noticed that the virtual environment would miss some events. After investigating the problem, we found that the events would eventually arrive, and if replayed, the display would appear correct.

We then focused on why these events were delayed, and to this end, we investigated the packet loss rates between nodes and the sink. These links experienced high losses – in some cases around 45% – which were unexpected given our earlier tests. Using the RSSI output from the DM2200, we measured the received signal strength (in the absence of transmissions) in the Smart Condo as well as other environments for informal comparison purposes. The interference observed in the Smart Condo far exceeded that observed elsewhere, which leads us to the next chapter.

Chapter 4

Data Collection and Pattern Overview

The impact of external interference became blatantly obvious to us in our 2008 deployment of the Smart Condo network to passively monitor an independent living environment [13, 85]. As soon as the network's simple transceivers (RF Monolithics TR8100 [75], 916.5 MHz) began operating, we noticed significant packet losses even over short distances and with no network congestion. Those losses disappeared when we moved the same set of nodes to another environment several blocks away for an in-lab study of their poor performance. Having thus confirmed the environment as the culprit, we informally checked it for interference. To our surprise, we found stronger interference in the Smart Condo than we had encountered in two other environments (Figure 3.5). This chapter explores the environment's interference in greater depth.

The Smart Condo is an example of an *urban* environment, and for such environments, we expect a degree of interference. WSN nodes typically operate at frequencies that are heavily used by other man-made signals [29], e.g., cordless phones, wireless local area networks (WLANs), building automation networks, and microwave ovens. Given the location of the Smart Condo, within Telus Centre (a moderately-sized office building) and across the street from a large apartment building, any number of interference sources could exist.

Noise technically consists of anything other than the signal [77, p. 134]. When a node receives a transmission, the ratio between the signal strength and the system (antenna and receiver) noise, the signal to noise ratio (SNR), ultimately determines the quality of the radio link [72, p. 275]. A higher ratio indicates a greater distinction of the signal from the interfering components and an increased likelihood that the receiver can decode it. In fact, it is possible to estimate the probability of an error given the SNR and the physical modulation scheme used by a transceiver (e.g., [72, p. 302]).

In an environment without motion, we have found that received signal strengths tend to remain relatively stable over time. For example, consider the signal (and noise) strength trace shown in Figure 4.1. We collected this trace while receiving packets at 85 Hz over a 13-minute period within the Smart Condo (channel $= 94$, $\text{port}_{\text{tx}} = 15$, $\text{port}_{\text{rx}} = 10$, $d = 2.59$ m). Notice that the signal strength (grey) remains stable even while the noise strength (black) fluctu-

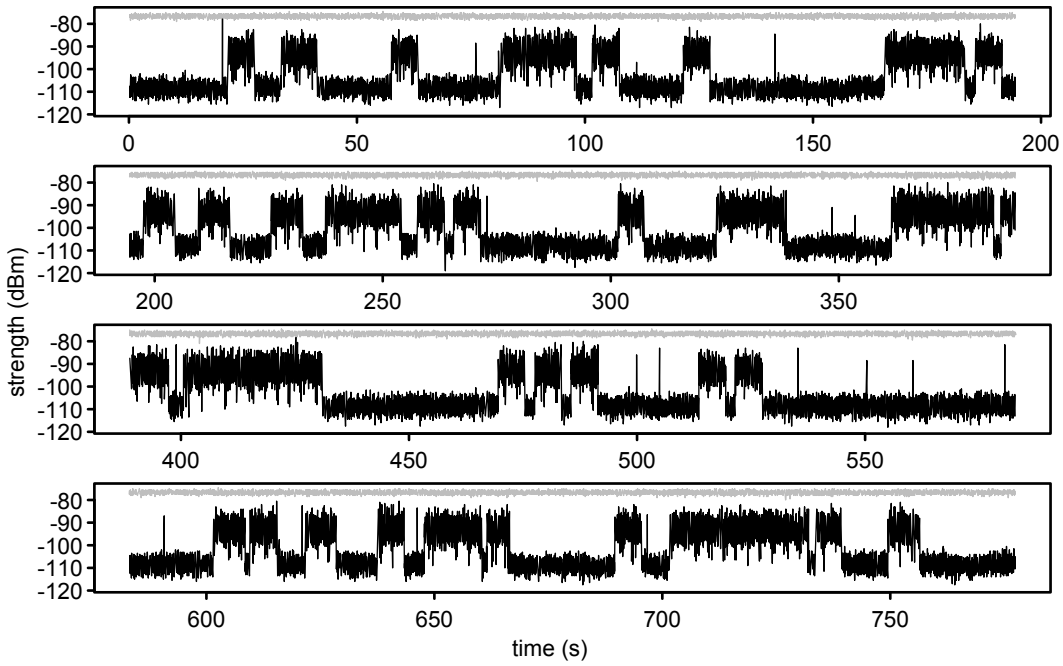


Figure 4.1: Trace from a node that received packets at 85 Hz and recorded (a) the signal strength after each recognized start symbol (grey) and (b) the noise immediately following the packet (black).

ates wildly. For this particular trace, the transmitting node sent 66 300 packets and the receiving node recognized 66 282 start symbols (99.973%) and correctly received 66 245 packets (99.917%). The success rate was so high because the fluctuating noise was significantly weaker than the signal.

As shown in Figure 4.1, noise can vary significantly with time. Meanwhile, the predominant noise model used in the design and analysis of communication systems is the *additive Gaussian noise channel* [69], a stationary process. This model applies best to stable noise traces, e.g., Figure 4.7(a), that lack external (and unpredictable) interference. Out of curiosity, we graphically checked the normality of some of these stable traces, and we saw the expected nearly-normal distribution of samples (Figure 4.2).

In systems with external interference, e.g., Figure 4.1, that interference may be separated from noise in the SNR to give the signal-to-interference-plus-noise ratio (SINR) [39, p. 160]. With this distinction, the interference becomes the large uncontrolled source of variation in the SINR while the signal and noise components remain relatively stable. The former component could then be modelled as a non-stationary process and the latter as a stationary one.

When channels operate near their receive sensitivity, they are especially sensitive to even small changes in the SINR. For example, consider the trace in Figure 4.3, which we collected while receiving packets at 85 Hz over a 13-minute period within the Smart Condo (channel $= 94$, $\text{port}_{\text{tx}} = 14$, $\text{port}_{\text{rx}} = 11$). This channel is the same as the one shown in Figure 4.1, but we collected the trace at a different time and for a different pair of nodes. Unlike the previous

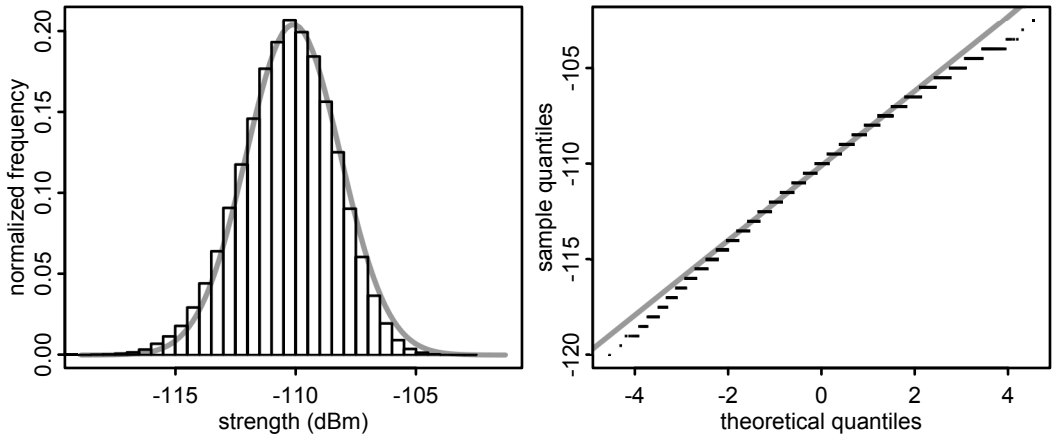


Figure 4.2: Graphical normality tests for a trace (channel $= 237$, port $= 2$, $n = 175\,000$ samples) of the noise in a stable (and apparently interference-free) environment. On the left, a histogram (frequency normalized to match the density) includes a grey normal curve. On the right, a normal quantile-quantile plot includes a straight grey line.

trace, the signal strength here was much lower, and the receiver only detected 24 354 start symbols (36.733%) and only successfully received 2383 packets (3.594%). Here, the interference had a huge impact on the packet reception rate.

In simulation studies, researchers typically compute their results using over-simplistic environmental models assuming that the only disturbance to the *proper* signal from a transmitting node at the receiver comes from white Gaussian background noise plus possible interference from peer devices (members of the same networked wireless system). The two types of disturbance have received considerable attention in research under the umbrellas of channel modelling (e.g., [43]) and MAC protocol design (e.g., [28]), respectively. The third type of disturbance, namely external interference from a different wireless system, has been much overlooked. This is unfortunate, considering that the incessantly growing number of wireless applications, combined with the limited spectrum available to them, will make the impact of external interference more and more pronounced. Based on our experience, external interference is already the predominant source of communication problems in many WSN systems, especially those deployed in densely populated urban areas.

In this chapter, we make a number of contributions. We describe a setup for high-frequency RSSI sampling using off-the-shelf wireless sensor nodes and sufficient cabling, and we present our results on sampling the Smart Condo environment. We specifically wanted that assessment to be carried out by a WSN (as opposed to some specialized and sophisticated spectrum analyzing equipment), because one of its objectives was to make the network aware of the interference via its own means, such that it could analyze the problem and respond to it all by itself. We show the variety of time-series behaviour that we encountered on channels and classify them into a few groups. In many cases,

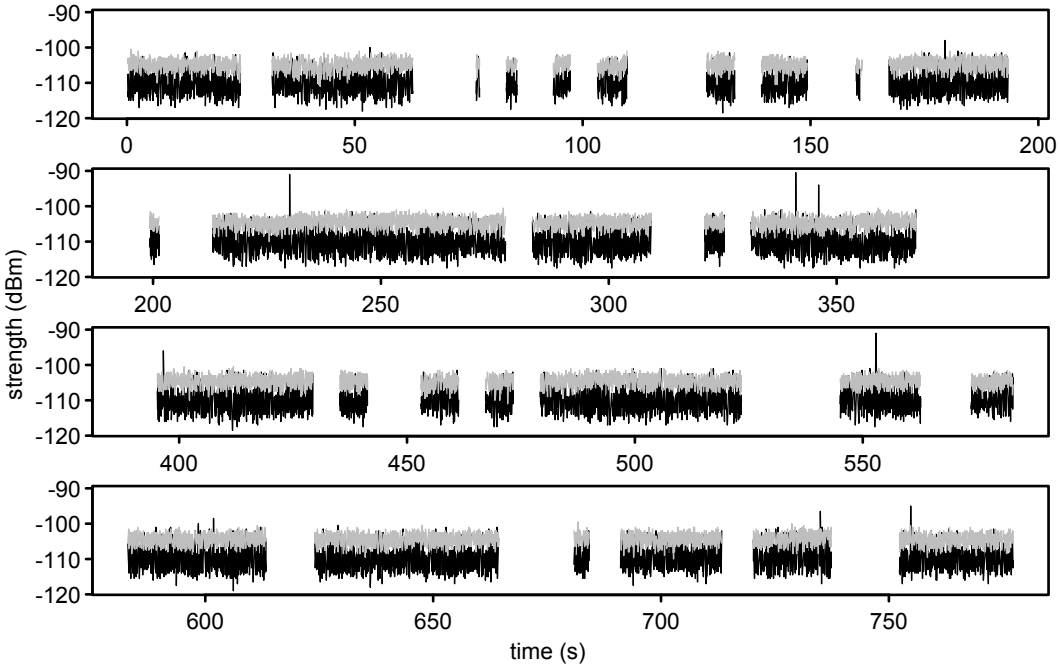


Figure 4.3: Trace from a node that received packets at 85 Hz and recorded the signal strength just after each recognized start symbol (grey) and the noise immediately following the packet (black).

these channels differ greatly from the commonly assumed additive white Gaussian noise (AWGN) model. In support of these classes, we compute statistics from each trace, use them to train a Bayesian network classifier, and present the results of using that classifier.

Needless to say, it would be highly presumptuous to claim that any interference patterns that we observed in a particular environment and on a particular day should be immediately generalized into blanket rules applicable to all wireless systems. We realize that the same environment may exhibit different characteristics if sampled again, and other environments may be completely different. However, the fact remains that we clearly saw a small number of simple and easily discernible patterns, and it is highly unlikely that what we saw was specific to the one environment. It is important for researchers to realize that the combination of noise and interference is rarely straightforward AWGN. In particular, the spiky patterns (that we describe later) are intuitively natural and expected to occur in many (otherwise unknown) wireless systems. They are also the most interesting from the viewpoint of navigating packet transmissions around them. For one thing, they may be representative of a typical (generic) WSN operating in the same area. Although unlikely to be the case in our environment, one can reasonably predict that an increasing proliferation of WSNs will bring about spiky interference patterns. Circumventing them can be viewed as solving a slightly augmented medium access control problem, whereby a certain subset of *peers* follow an unknown and presumably non-responsive (but nonetheless systematic) schedule of transmissions. Conse-

quently, the problem appears general and interesting enough to warrant further studies.

One should also keep in mind that the ultimate goal of any WSN performance study is to guide practical deployments involving tangible hardware with painfully idiosyncratic properties. The most extreme example of such a property in our case is the systemic suppression of certain channels separated by half-multiples of the crystal frequency driving the RF chip (seen in the high-level trace of Figure 4.8). It is clearly impossible to account for all such *features* in any blanket theoretical model, as it is in any experimental study whose results one may be tempted to extrapolate onto unexplored hardware. Nonetheless, the actual behaviour of real devices is what ultimately matters the most; thus, we should try to bring as much order as possible into those necessarily unsystematic, but extremely valuable from a practical standpoint, observations from real-life implementations.

4.1 Related work

The related work most relevant to our work falls into a few categories. First, we explore research on the transitional or grey region which discusses how relatively small changes in the SINR can affect whether packets are successfully received – it helps motivate our work. Next, we look at the types of devices that can cause interference, observations about interference, and how other researchers have used WSNs to sample channels. Finally, we describe how work on cognitive radios looks at determining channel occupancy based on observations.

4.1.1 The transitional/grey region

Many researchers have reported on a transitional (or grey) region in the SINR where, over time, a subset of nodes may fluctuate between successful and failed transmissions, e.g., [1,9,74,80,84,97,100,101,104]. Patterns in the interference are most likely to affect these nodes first, and for that reason, we will begin with a review of this work.

Zhao and Govindan [101] quantified the size of the area in three different environments: an office building, a park, and a parking lot. They set up 60 nodes operating in the 70-centimetre amateur radio band at 433 MHz in a line topology and had the node at one end transmit packets at 1 Hz. In the building and park environments, they noticed surprisingly large grey areas of almost one-third and one-fifth of the communication ranges, respectively.

Later experiments by Son, Krishnamachari, and Heidemann [80] considered one less variable: hardware variations. They discovered that for a particular node and level of signal strength, the grey region is actually quite narrow. Furthermore, the specific width and location of the grey region depends on both (a) the transmitter hardware and (b) the transmission power. The grey region only appeared relatively wide when many radios were used, and in that case it spanned roughly 6 dB.

More recently, Zamalloa and Krishnamachari [100] approached the problem from a different perspective – mathematically. Although they derived expressions for the location and extent of the transitional region, their model does not consider interference. They mention that the noise floor varies over time and list large changes in temperature and interference as possible causes.

4.1.2 Observations and sampling

Interference can originate from any number of sources in an environment. For example, Chandra [21] used a spectrum analyzer in a three-story building to explore frequencies in the 900 MHz and 1.8 GHz bands. He looked at the noise generated by electronic equipment in a workshop, a photocopier, an elevator, and fluorescent tubes, and he concluded that all should be modelled as interferers in the building. He did not, however, consider detecting these interferers using WSN hardware.

Using sensor platforms, Srinivasan, Dutta, Tavakoli, and Levis [81] studied packet delivery performance. They encountered large correlated spikes (up to -35 dBm or higher) in their traces and investigated these further. With the nodes synchronized, they checked for spatial correlation among the spikes, and upon finding high correlation, concluded that the spikes originated externally to the nodes. We also noticed correlation in many of our traces, which led us to the same (external) conclusion.

Researchers working on closest-fit pattern matching (CPM) sampled noise in both indoor and outdoor environments [49, 78]. While we sampled at 5 kHz, they sampled at relatively low rates (1 kHz or less) and made only the informal comments about interference patterns that we reiterate in the following paragraphs.

Using the CC2420 IEEE 802.15.4 transceiver, Lee, Cerpa, and Levis [49] sampled the chip's RSSI register at 1 kHz (the register itself updates at 62.5 kHz). By storing the retrieved measurements in the device's flash memory, they could record samples for 197 s. They sampled noise on channels that both overlapped and did not overlap IEEE 802.11b channels in Wi-Fi-enabled buildings, Wi-Fi-enabled outdoor areas, outdoor quiet areas, and controlled areas. They observed three key characteristics in their samples: (a) spikes sometimes as strong as 40 dB above the noise floor, (b) many of the spikes were periodic, and (c) over time, the noise patterns changed. They did not encounter the shifting-mean characteristic that we observed, and they offered little description of the patterns beyond what we summarize here.

Using TelosB nodes with CC2420 transceivers, Rusak and Levis [78] sampled the chip's RSSI register for packets transmitted between a pair of nodes at both 4 and 100 Hz. They sampled channels in buildings at Cornell and Stanford University and concentrated on modelling the signal strength rather than just noise. Both environments had a number of interference sources including 802.11 wireless networks, cordless phones, microwaves, and personal wireless access points. Although they mentioned the presence of interferers, they did not comment on the observed interference patterns.

Most recently, Srinivasan, Dutta, Tavakoli, and Levis [82] expanded on much of their previous work. With six synchronized nodes, they sampled RSSI

values at 128 Hz and explored the correlation of noise traces. They focused on the interaction of specific protocols, e.g., IEEE 802.11b (WLAN), 802.15.1 (Bluetooth), and 802.15.4 (ZigBee). They observed 802.11b interference at 45 dB above the noise floor and suggest avoiding channels that coexist with 802.11b networks.

4.1.3 Cognitive radios

A better understanding of interference patterns without resorting to assumptions about the coding and modulation strategy of the interferers also relates to the wider field of cognitive networking [3, 6, 18, 19, 94]. Wang and Liu [94] summarize the typical cycle of a cognitive radio (CR) as

1. sense the environment,
2. analyze the sensed data,
3. reason about the best response to that analysis, and
4. adapt new operating parameters.

To accomplish these tasks, they usually make use of special hardware. Since we specifically aim to assess channels using a WSN, we will keep our review of CRs brief.

For the lower networking layers, CRs aim to opportunistically use unoccupied frequency bands by determining the near-term occupancy of a channel through noise and interference observations [99]. In our work, we attempt to characterize channels (at a high level) that are potentially occupied, a step needed by cognitive radios, and rather than just producing a binary occupied/unoccupied classification, our approach produces a class. Some of the classes, e.g., those exhibiting signs of spread spectrum interferers, could, depending on the cognitive networking scheme, be used at the same time for narrowband transmissions by the cognitive network. A medium access control (MAC) scheme based on this idea could enhance an existing protocol such as B-MAC [67], which already takes samples between transmissions.

For the upper layers, an awareness of the interference could present other opportunities. For example, an interference-aware routing protocol could potentially route transmissions around weak interference. In another situation, nodes could potentially exploit particularly strong and consistent periodic interference for synchronization purposes.

4.2 Data collection

Within the 80 m^2 space of the Smart Condo [13] at the University of Alberta, we deployed a four-by-four grid of 16 wireless nodes (Figure 4.4). We used 1.83 m grid spacing and elevated each node 28 cm off of the floor. While running the experiments, we kept the room's doors closed and there was no movement within the room. Figure 4.5 shows a number of the nodes (located on the left side of Figure 4.4) sitting on their paper pedestals.

In the experiments, we used EMSPCC11 wireless nodes (Figure 4.6) provided by Olsonet Communications [62]. These devices consist of a Texas Instruments (TI) MSP430F1611 microcontroller and TI/Chipcon CC1100 transceiver

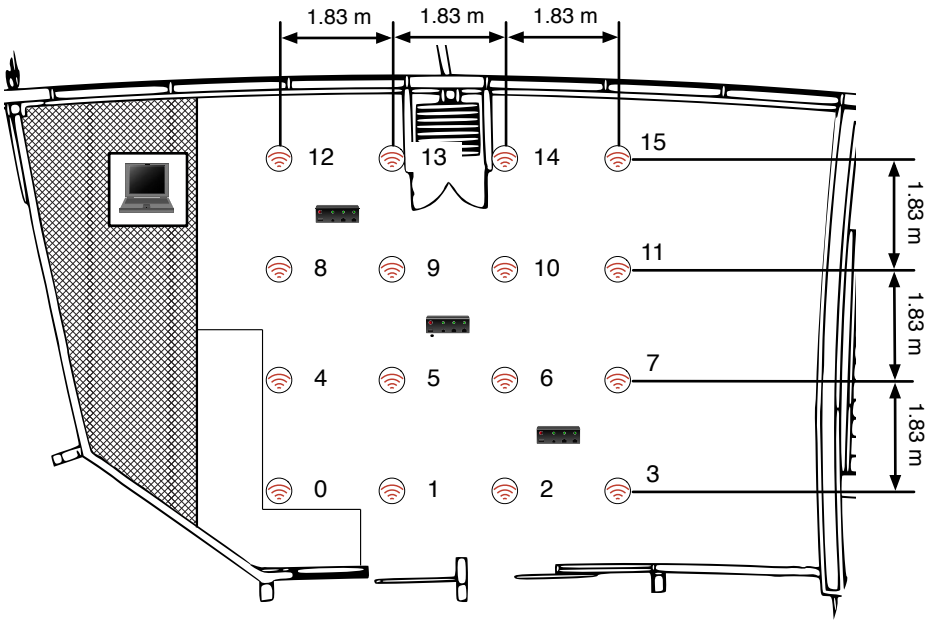


Figure 4.4: The experiment setup within the Smart Condo. The circles represent the nodes. The small black boxes represent the 7-port USB hubs. The notebook computer in the top-left recorded the results.

Figure 4.5: We set up a grid of nodes within the Smart Condo.

Figure 4.6: An Olsonet EMSPCC11 wireless node.

powered by two AA batteries. In terms of software, they run a low-footprint operating system named PicOS [2] that supports multithreaded applications.

Our nodes sample the received signal strength indicator (RSSI), which measures the radio-frequency power input at the transceiver [90]. The CC1100 stores it in an internal register, and the microcontroller can easily retrieve it through the serial peripheral interference (SPI) that connects the two components. The register value update frequency is

$$f_{\mathrm{RSSI}} = \frac{2 \times \mathrm{BW}_{\mathrm{channel}}}{8 \times 2^{\mathrm{FILTER\_LENGTH}}} \tag{4.1}$$

where

$$\mathrm{BW}_{\mathrm{channel}} = \frac{f_{\mathrm{XOSC}}}{8 \times (4 + \mathrm{CHANBW\_M}) \times 2^{\mathrm{CHANBW\_E}}}, \tag{4.2}$$

and where in our configuration the FILTER_LENGTH $= 1$, $f_{\mathrm{XOSC}} = 26 \times 10^6$, CHANBW_M $= 0$, and CHANBW_E $= 3$. By working through the formulae, the RSSI register update frequency f_{RSSI} is $12\,695.31$ Hz. We eventually need to convert the 8-bit 2's complement RSSI value to dBm, and the 2's complement offset depends on the data rate [91]. The CC1100 driver for PicOS implements one set of data rates (4.8 kbps, 10 kbps, 38.4 kbps, and 200 kbps), and TI has published the procedure for another set (1.2 kbps, 38.4 kbps, 250 kbps, and 500 kbps). When configuring the CC1100 for our experiments, we looked for the intersection between these two sets and selected the only choice: 38.4 kbps using 2-FSK modulation.

A PicOS application collects noise measurements by reading the 8-bit 2's complement RSSI value and immediately writing it to the UART (without converting it to dBm). It performs these actions in a tight loop that involves the OS scheduler at every iteration, but since no other user threads are running, the scheduler latency is relatively constant. We verified the periodicity of our sampling by toggling an LED on every call to the output function and monitoring the pin with an oscilloscope. With this software, we can obtain the RSSI at

over 5 kHz, well under its 12.7 kHz update rate, but at the same time, much higher than previously described in the literature.

Writing values to the UART necessitates connecting all the nodes to a computer by wire. The EMSPCC11 provides direct access to the MSP430's UART pins at TTL levels, so we opt to use a TTL RS232 to USB interface cable (FTDI TTL-232R-3V3). Now with a USB interface, it is easy to connect all 16 nodes to a single computer using a combination of USB extension cables and powered 7-port USB hubs (Digitus DA-70227). In making the connections, we do not exceed USB's maximum cable length of 5 m.

A single application on the personal computer (PC) opens all 16 serial ports when it starts. Given 16 connected nodes and a 5 kHz sampling rate, samples arrive at roughly 80 kHz. Each sample becomes a line in a comma-separated values (CSV) text file, with an average line length of 25.6 bytes. Thus, the application writes the file at around 2 MB/s.

The whole application is very sensitive to latency. The TTL-232R-3V3 has a 256-byte receive buffer (about 51 ms of buffer space), and without taking special precautions, we experienced buffer overruns. To eliminate them, we use the following sufficient, but possibly not optimal, steps:

- introduce a large circular buffer of blocks of bytes (using 64 KB blocks),
- read the ports and write measurements to the circular buffer (in the main thread),
- assign this thread real-time priority (in Mac OS X 10.6, give it 2.5 ms of computation time every 5.0 ms and allow it to be preempted), and
- write completed buffer blocks to disk (in a second thread).

With these precautions, our application processed the samples with ease on a late-2006 model MacBook Pro. The FTDI buffers never filled to more than 100 bytes.[1] In fact, in the early tests of this modified setup, they would rarely fill to more than 10 bytes.

The PC-based application produces time-stamps for all measurements as they arrive. Because measurements arrive in blocks, it interpolates times using the time of the last received block and the current time as bounds. We adopted this PC-based time-stamping approach to reduce the data sent over the serial link and avoid assuming that all of our nodes were always perfectly synchronized.

Before taking proper measurements, we verified whether the connection of our nodes by wires to a single data collection point affected their RF behaviour. Note that the careful isolation of the RF tract on the EMSPCC11, which is a common practise in professionally designed RF equipment, gave us no reason to suspect an interaction between the wired and RF interfaces, but we nonetheless verified our expectation. By disconnecting individual nodes, logging RSSI values to memory, and then (visually) inspecting their RSSI readings, we were able to ascertain that neither the USB cables alone, nor their connections to the central hub had a perceptible impact on the RF channels.

[1] In the application, we added code to print out the buffered number of bytes if it ever exceeded 100. In our experiments, this code was never executed; the maximum buffered bytes may have been far less than 100.

With the described measurement framework in place, we proceeded to measure the RSSI on each of the node's 256 channels. Our nodes were configured with a base frequency of 904 MHz. The channels are spaced 199.9512 kHz apart, and the transceiver uses a receive filter bandwidth of 101.5625 kHz. These settings allow our nodes to listen on frequencies from 904 MHz to 928 MHz (within the ISM band) and 928 MHz to 954 MHz (outside the ISM band). For each channel, we collected exactly 175 000 samples for each node for a duration of less than 35 s. For the whole collection process, the time-span was roughly 2.5 h and the final CSV data file consumed 18.38 GB. Please note that in all the measurements reported here, our sensor nodes remained silent and did not introduce any transmissions of their own.

4.3 Channel classification

With the 4096 traces stored in a CSV file, we converted the 8-bit 2's complement RSSI values into signed dBm values (with $\frac{1}{2}$ dB resolution) using the procedure outlined in the CC1100 documentation [91]. To plot the received signal strength (dBm) against time (s) for each of the 4096 traces, we thinned the data first using an implementation of the Douglas-Peucker line simplification algorithm [30]. The thinned results, which had on average 2077 points per trace, maintained the visual characteristics that we needed to hand-classify traces. At the same time, plotting the thinned version was much more efficient than plotting the full 175 000-point traces. We used R for the majority of our data analysis [71].

Through visual inspection, we identified five general categories for the interference patterns.

1. The *quiet* channel is characterized by a low maximum and appears to lack interference.
2. The *quiet-with-spikes* channel is similar to (1), but it has low-frequency (less than 1 Hz) short-duration spikes that give it a higher maximum.
3. The *quiet-with-rapid-spikes* channel has a higher frequency of spikes than (2).
4. The *high-and-level* channel exhibits a high minimum and low deviation.
5. The *shifting-mean* channel exhibits shifts of its mean, and its RSSI samples have a bimodal distribution.

See Figure 4.7 for an illustration of each class of channel.

Given the general classes of interference patterns, we hand-classified the trace for each channel/node combination. The task of classifying the samples was particularly difficult given that a single trace might contain overlapping patterns or a pattern at only a weak strength. We did not test the intra-rater reliability of these hand-classifications, and we would expect some variance. When more than one characteristic was present in a trace, we tried to classify it as the visually dominant pattern. For example, we tended to classify a trace as shifting-mean rather than quiet-with-spikes and quiet-with-rapid-spikes rather than shifting-mean. Figure 4.8 shows the classifications for the

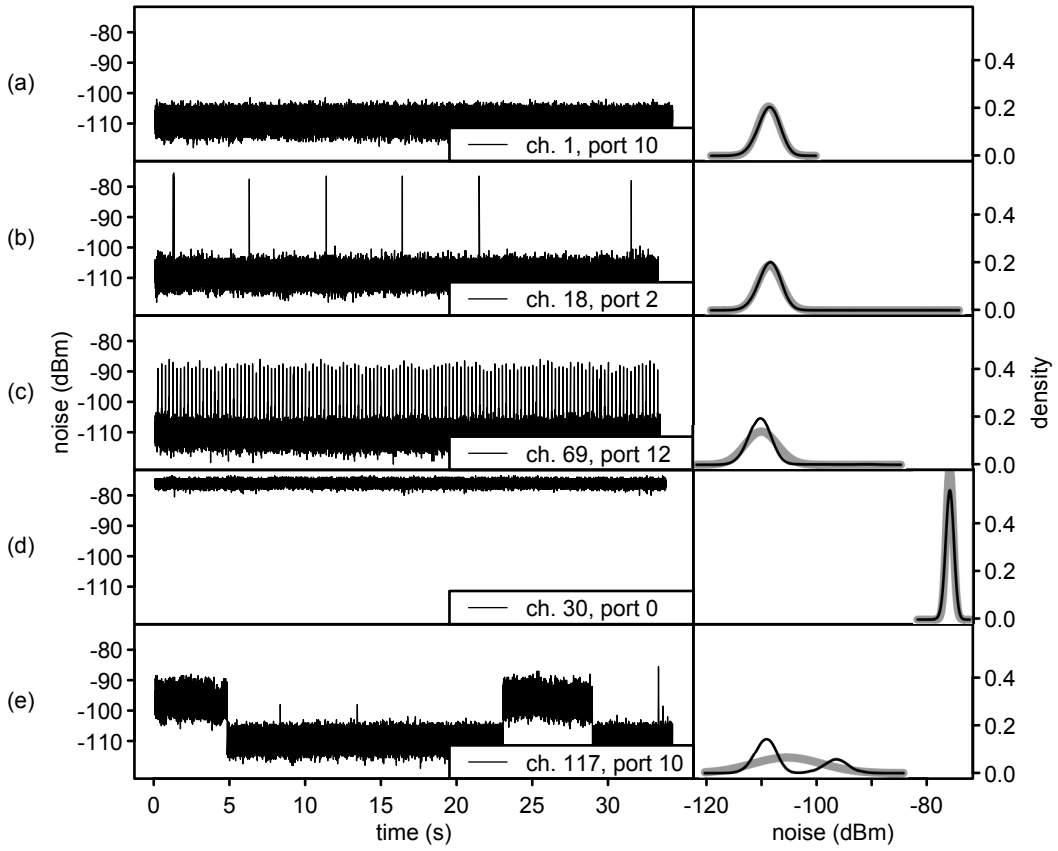


Figure 4.7: The different primary classes of channels that we identified in our RSSI traces. From top to bottom, the figure shows the samples and density plot for the (a) quiet channel, (b) quiet-with-spikes channel, (c) quiet-with-rapid-spikes channel, (d) high-and-level channel, and (e) shifting-mean channel. On the density plot, the grey line indicates the Gaussian distribution given the mean and standard deviation.

Table 4.1: Classification of traces for both ISM and non-ISM bands.

Classification	ISM	Non-ISM	Total
quiet	144	1625	1769
quiet-with-spikes	1142	183	1325
quiet-with-rapid-spikes	523	25	548
high-level	32	69	101
shifting-mean	79	274	353

4096 combinations and Table 4.1 summarizes our counts for ISM and non-ISM bands.

We encountered spikes predominantly within the ISM band. When inspecting some of the spikes, we calculated very short durations of around 6 ms; we speculate that they result from the presence of spread spectrum interferers.

Figure 4.8 highlights a curious pattern in channels grouped in the high-level class. The four cases on channels 30, 95, 160, and 225 are located 65 channels apart. To investigate this curious interference, we set up the nodes in a new environment, added some shielding to them, and again measured these channels to find no decrease in the strength of the interference. That was suggestive of a hardware issue and, indeed, a closer inspection revealed the problem signalled in [23], a systemic attenuation of those channels whose frequencies fall at $806 + n \times 13$ MHz (13 MHz is $\frac{1}{2}$ of the transceiver's crystal frequency), which perfectly agrees with the observed *anomaly*.

In the non-ISM band, we noticed that channels 126-130 and 138-141 had very powerful shifting-mean signals (e.g., Figure 4.9). Using Spectrum Direct at Industry Canada, we searched the Assignment and Licensing System (ALS) database[2] and found that the closest registered frequency is 931.737500. This frequency is registered to Telus Communications Inc. for their 900 MHz paging service. For the lower channels, 126 to 130, we also found that they were registered for paging services.

We also found that the licensed spectrum is relatively quiet apart from the paging service and accounting for the explained anomalous behaviour of channels $30 + n \times 65$.

4.3.1 Automated classification

After manually classifying the traces, we rather informally explored the feasibility of automatically classifying them. Classification in this sense is the task of identifying the best class label given a set of features [22]. For each trace, we produced a feature vector by computing a number of summary statistics: (a) mean, (b) standard deviation, (c) skew, (d) kurtosis, (e) minimum, (f) maximum, (g) dip, (h) 99.5^{th} percentile, and (i) 99^{th} percentile. Suffice it to say that the dip statistic is used when testing for unimodality [41], and we explain it later when we formally explore classification in the next chapter. We included

[2]See `http://www.ic.gc.ca/eic/site/sd-sd.nsf/eng/h_00025.html`.

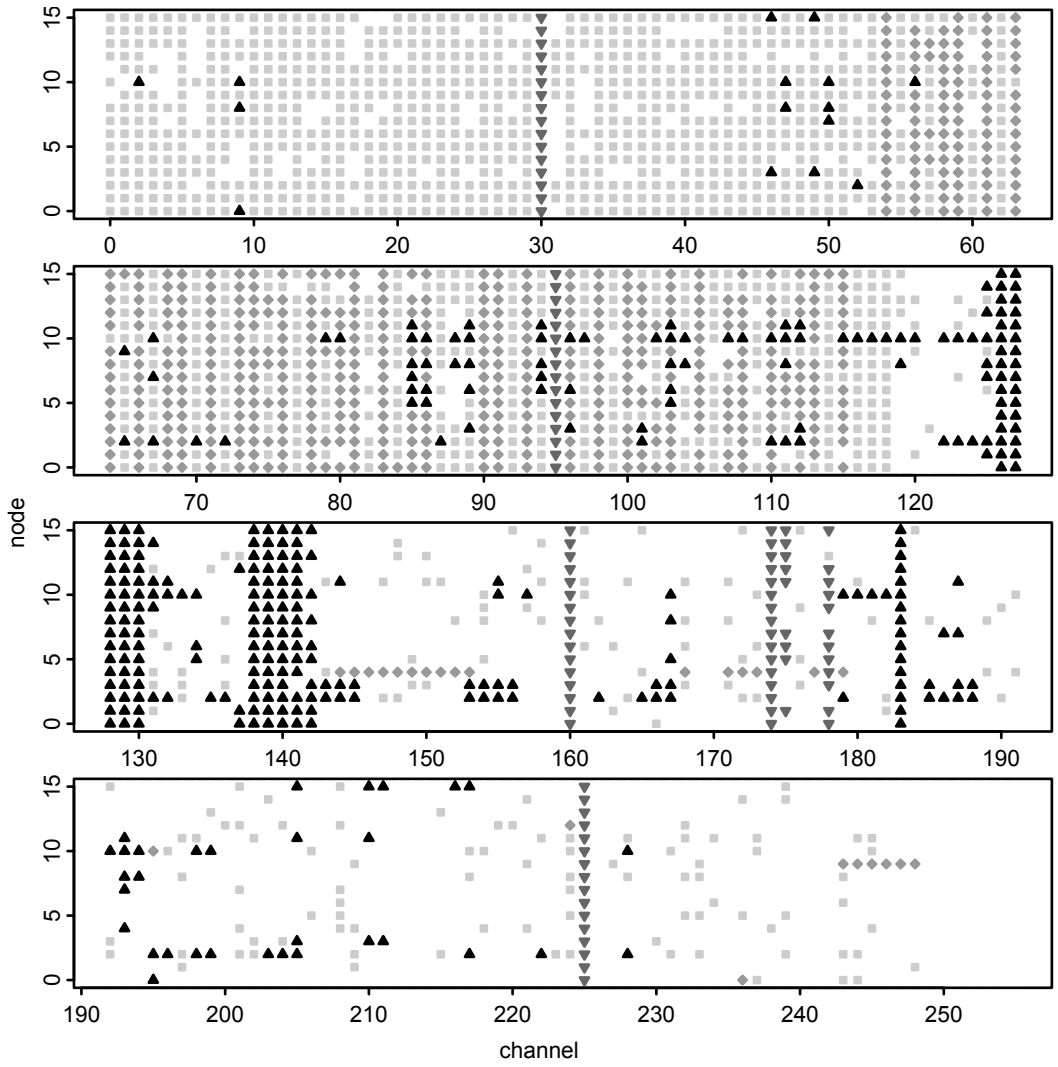


Figure 4.8: The result of hand-classifying noise traces from 256 channels with 16 nodes per channel. The correspondence between symbol and classification is as follows: (a) white: quiet, (b) ■: quiet-with-spikes, (c) ◆: quiet-with-rapid-spikes, (d) ▼: high-level, and (e) ▲: shifting mean.

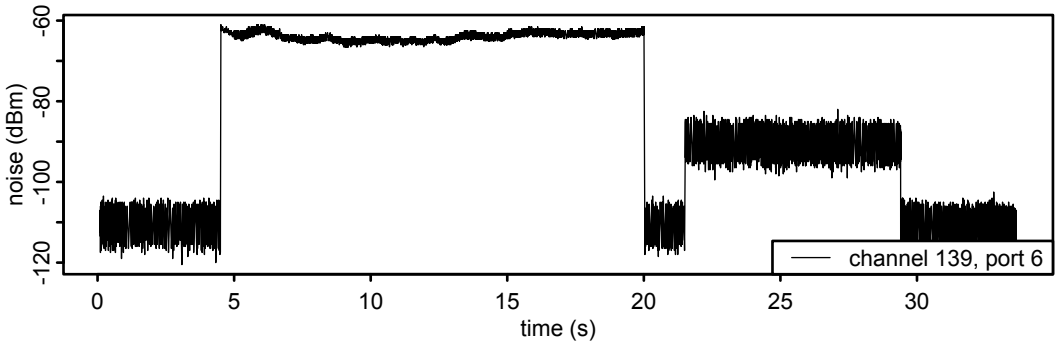


Figure 4.9: A sample showing particularly strong interference at 931.793 MHz; we attribute this interference to pagers.

the percentile metrics in an attempt to better identify the quiet-with-rapid-spikes traces.

We used Weka 3.6.2 [40], a comprehensive collection of data mining and analysis tools, to explore automated classification. Although we tried a number of its available algorithms and options, we made no attempt to rigorously explore and compare the performance of different classifiers to determine the best. In this section, we summarize our experience with the Bayesian network classifier, which we selected because of its particularly good performance in our informal comparison.

A Bayesian network consists of

- a directed acyclic graph (DAG) $G = (V, \vec{E})$ with vertices V and edges $\vec{E}$ and
- a table of probabilities for each vertex.

The vertices in this graph G represent features, and the directed edges indicate direct causal influences between the linked features [66, p. 117]. For each vertex, a table summarizes the strengths of the forward conditional probabilities for every possible state of its direct parents. Given a Bayesian network, it is possible to compute the conditional probability of a node given values assigned to the other nodes [22]. A classifier based on the network will return the class label that maximized the posterior probability of that class given the attributes [33].

The Bayesian network must be learnt prior to classification, and learning generally follows a two-stage process [17]. First, an algorithm learns the network structure, and in our experiments, we used the tree-augmented naïve Bayes (TAN) approach [33]. Second, an algorithm learns the probability tables, and for that, we used Weka SimpleEstimator class, which produces direct estimates of the conditional probabilities [17]. Prior to creating and using the classifier, we discretized the feature vector using Weka's supervised discretize algorithm [32]. By using a supervised approach, the algorithm considers the class attribute as the ground truth.

We loaded a data file prepared in the attribute-relation file format (ARFF) containing the class of channel and the statistics. On these data, we ran 10-

Table 4.2: Accuracy statistics for the BayesNet classifier with 10-fold cross-validation.

	TP rate	FP rate	Precision
quiet	0.931	0.046	0.940
quiet-with-spikes	0.832	0.055	0.878
quiet-with-rapid-spikes	0.786	0.052	0.702
shifting-mean	0.714	0.031	0.685
high-level	0.970	0.002	0.933

Table 4.3: Confusion matrix for the BayesNet classification. Abbreviations as follows: q: quiet, qs: quiet-with-spikes, qrs: quiet-with-rapid-spikes, sm: shifting-mean, and hl: high-level.

		Classified as					
		q	qs	qrs	sm	hl	
Actual	q	1647	59	38	18	7	1769
	qs	65	1103	95	62	0	1325
	qrs	18	63	431	36	0	548
	sm	20	31	50	252	0	101
	hl	3	0	0	0	98	353
		1753	1256	614	368	105	4096

fold cross-validation with the BayesNet classifier. Overall, it correctly classified 86.21% of the instances.

Table 4.2 provides accuracy statistics by class. The true positive (TP) rate refers to the proportion of a given class that was actually classified as the class. The false positive (FP) rate refers to the proportion misclassified as a class. Finally, precision is defined as

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{4.3}$$

and indicates the proportion of the true classifications that are correct.

Table 4.3 shows the confusion matrix for the classifier. The sum of each row indicates the number of instances of each individual class. The columns show how the classifier predicted the actual classes. In the ideal case, it would be a diagonal matrix as non-diagonal elements indicate the classifier's errors.

The BayesNet classifier obtained the best performance with the quiet and high-level traces. Both of these classes were clearly distinguished by characteristics in their feature vectors. In the former case, traces had a low maximum, and in the latter case, they had a high minimum. The classifier had more trouble on the channels with spikes and the shifting means. All three of these types can look quite similar depending on the strength (dBm) of the pattern.

4.4 Summary

This work was inspired by the unusually high packet loss rates that we encountered after deploying an initial WSN within the Smart Condo. To further explore the environment, we deployed a second WSN to sample the received signal strength indicator (noise and interference) on 256 channels ranging from 904 to 954 MHz. By visually inspecting the collected data, we observed five distinct patterns in the traces; we then hand-classified each trace as belonging to one of these types. We later used these classifications for the ground truth when training and evaluating a classifier.

In the ISM band, we encountered a number of quiet traces ($n = 144$), while the vast majority ($n = 1776$) contained some form of interference. While much of this interference would have a very low impact on receptions (spikes, $n = 1142$), the remaining three types would be much more detrimental. The high-and-level and shifting-mean channel types are best avoided. In the former, the interference is constant, and in the latter, it is highly unpredictable. For other channel types, such as the rapid-spikes channels, nodes could employ avoidance strategies given the periodic and consistent interference pattern.

In the non-ISM band, we saw a much greater occurrence of quiet channels ($n = 1625$). Our second-highest class, shifting-mean ($n = 274$), was often very strong with a consistent level across all deployed nodes. To investigate these traces, we searched the Assignment and Licensing System (ALS) database at Industry Canada. We found that the pattern often occurred on frequencies used by the paging services of various wireless providers in Edmonton, Alberta.

Subsequently, we explored using a Bayesian network classifier on statistics extracted from the traces. The classifier performed reasonably well on all types of traces, exhibiting (not surprisingly) somewhat better accuracy for the quiet and high-level classes.

Reflecting back on our initial high packet loss rates, although the DM2200's TR8100 transceiver transmitted at reasonably powerful levels (10 dBm), it used a very simple encoding scheme (on-off keying) that is particularly susceptible to interference [60, 93]. It operated at 916.5 MHz, which is near the EMSPCC11's channel 63, and on this channel, we later observed the quiet-with-rapid-spikes pattern. This pattern helps explain the poor performance that we initially encountered.

Chapter 5

Classifying Patterns and Subsampling Traces

In the previous chapter, we investigated noise and interference samples from 256 channels in an urban environment [15]. We described how we collected the rather extensive time series data, and we identified five common patterns in those data, namely, the (a) quiet, (b) quiet-with-spikes, (c) quiet-with-rapid-spikes, (d) high-and-level, and (e) shifting-mean patterns (Section 4.3). After making these observations, we extracted a feature vector from each trace and explored the automatic classification of traces. To evaluate the classifier, we classified each trace by its predominant pattern to establish the ground truth, used these hand-classified vectors to train the classifier, and compared the classifier's output against the ground truth.

Our earlier work produced promising results, but the classifier was not suitable for a resource-constrained node. In this chapter, we refine our approach so that individual nodes can classify channels. We primarily address two related questions:

1. How can we use a classifier to determine whether a particular pattern is present in an RSSI trace (Section 5.1)?
2. How can such classification occur within the limited resources of a WSN node (Section 5.1.2 and Section 5.2)?

To this end, we focus on developing independent classifiers for the two patterns that have dynamics most detrimental to transmissions, quiet-with-rapid-spikes and shifting-mean. We elect to use independent classifiers because the patterns can coexist, i.e., they are not mutually exclusive, and in both cases, we are interested in detecting the pattern's presence, rather than accurately describing it. Note that when we later evaluate the performance of these refined classifiers, we make neither reference nor comparison to the earlier Bayesian network given its relatively high complexity.

Consistent with work in cognitive networking (Section 4.1.3), and in order to address the second question raised earlier, is the desire to use as small a number of samples, together with the right features and classification algorithm (expressed as a decision tree), to allow nodes to reassess the channel behaviour and re-classify channels on an as-needed basis. An awareness of en-

ergy consumption motivates us to do this collection and classification efficiently, since batteries often power nodes. Consistent with these objectives, we use the same hardware, i.e., radio-frequency (RF) transceiver and microcontroller, for the measurement and classification as we use in our deployed networks. We do not assume the use of any special equipment or special calibration protocols.

5.1 Classification

Recall that in our initial classification of traces (Chapter 4), we encountered traces containing more than one characteristic. To avoid classifying them by the dominant one in this work, we instead consider each characteristic independently. From the 4096 original traces, we randomly selected 1024 traces to closely inspect in a random order. For each trace, we evaluated it for the following five characteristics:

1. spikes: for apparently random and infrequent spikes, their approximate strength (dB) above the noise floor and number,
2. periodic-spikes: for periodic and infrequent spikes, their approximate strength and number,
3. rapid-spikes: for periodic and frequent spikes, their approximate strength,
4. shifting-mean: the strength of the shift, and
5. high-and-level: the strength of the level above where we would expect to find the noise floor.

Most of the frequent spikes occurred at 4 Hz, while most of the infrequent ones occurred at 0.2 Hz or less, and we never encountered any difficult-to-classify cases that required us to measure the time between spikes.

Beyond the visual differences of the infrequent and frequent spikes, other reasons made it quite natural to distinguish the two types. In our observed traces, the frequent spikes occur approximately 20 times more often than the infrequent spikes. Therefore, recognizing the frequent case should require a magnitude fewer samples. Similarly, once the pattern is recognized, the frequent case lends itself much better to opportunistically timing transmissions (Chapter 6), where nodes waking with a pending transmission may need to find a spike in order to track the interference. Finally, the two different patterns have different implications for the packet error rate. Given Smart Condo-sized packets and data rates, a transmission can take up to 50 ms to complete. With the spikes pattern, a node oblivious to the interference can quite possibly achieve a 99% packet reception rate even when the interference obliterates every encountered packet. With the rapid-spikes pattern, however, the higher frequency of spikes now suggests a packet reception rate of 80% – a low enough rate to warrant some way of handling the rapid spikes to avoid this drastic performance degradation.

From these five introduced numeric values, we derive two new Boolean class attributes: rapid-spikes and shifting-mean. We originally created a third one, spikes (for any of the three spike cases), but we later dropped this class given the low impact of its patterns. These attributes indicate the pattern's presence

in a trace – the ground truth: we set these new attributes to true when a pattern's level exceeds 3 dB. We use these values while training and evaluating classifiers, and herein, any reference to their names is to the Boolean rather than numeric attribute. The rapid-spikes attribute is true for 154 of the 1024 traces (15.0%), the shifting-mean one is true for 91 (8.9%), and they are both true for 9 (0.9%).

5.1.1 Decision trees

Many different classification techniques exist, and we focused on those most feasible for low-powered nodes. We elected to use a decision tree classifier, since once built offline, it has low memory and processing requirements. Quite possibly the most well-known algorithm in the literature for building decision trees [45] is the C4.5 classifier [70], and we adopted it for our experiments. C4.5 actually refers to the collection of programs described in [70]; the primary program, sharing the same name, generates a classifier in the form of a decision tree.

Weka 3.6.2 [40], data mining software written in Java, incorporates the implementation of the C4.5 classifier that we used. Algorithm 1 summarizes it: the algorithm essentially builds the tree top-down and attempts to select the best attribute for each decision node as it builds. It begins the process with all of the feature vectors in the training set, and as it goes, the tree's decisions gradually partition those vectors into smaller subsets.

Algorithm 1 Building the C4.5 decision tree

1: check base cases
2: initialize list U for storing useful splits
3: initialize list G for storing their information gains
4: **for each** a in the non-class attributes **do**
5: **if** split(a) is useful **then**
6: U.append (a)
7: G.append (gain (a))
8: **end if**
9: **end for**
10: let a_{best} be one attribute a from U such that
 $\text{gain}(a_{\text{best}}) \geq \text{mean}(G)$ **and**
 $\text{gratio}(a_{\text{best}}) = \max_{a \in U} \text{gratio}(a)$
11: create decision node that splits on a_{best}
12: recurse on each subset below decision node

The algorithm initially tests a number of base cases that can end the recursion if (a) all of the feature vectors belong to the same class or (b) not enough feature vectors exist to warrant a decision node. Both of these cases result in a leaf node with its class determined by the class majority among its feature vectors.

When the base cases fail, it will create a new decision node that contains a test, and it will explore each non-class attribute for that test. For a continuous numeric attribute, the decision node will test whether the attribute value is

above a certain threshold. Such a test results in a Boolean outcome: true or false. For a discrete (nominal) attribute, each possible value of the attribute will result in an outcome.

Each possible outcome of the decision yields a child, and the training set's feature vector is partitioned (split) amongst the children according to the result of the test. It is possible for a child to contain no feature vectors, i.e., when a possible nominal value does not occur in the subset of feature vectors. In these cases, the child becomes a leaf with its class determined by the most frequent class of its parent. A good attribute for a decision node yields two or even a few children, where each child is reasonably pure, i.e., the feature vectors in the child are predominately of the same class. In fact, before an attribute becomes a candidate for the best split, an initial usefulness test limits the candidates to (a) non-numeric attributes with a limited number of outcomes and (b) numeric attributes. After building the tree, a reduction phase (not described here, but found in [70]) collapses and prunes the tree.

To determine the best attribute for a decision node, the concept of *information* plays a fundamental role [70, p. 21]. As usual, the formula $-\log_2 P$ conveys the information content of a message in bits, where P is the message's probability. For example, given four equally probable messages ($P = \frac{1}{4}$), the information content of any one of them is $-\log_2(\frac{1}{4})$ or 2 bits.

Algorithm 1's first use of information occurs on line 7, when it calculates the *gain* of splitting the feature vectors F on the attribute a. Gain represents "the informational value of creating a branch on the ... attribute [a]" [96]. Prior to calculating this difference, the approach first needs a baseline, which in this case is the informational value prior to creating the branch.

Let F be the feature vectors available when creating a decision node. Given a case in F,

$$\text{info}(F) = -\sum_{i=1}^{n} \left[\frac{\text{freq}(C_i, F)}{|F|} \times \log_2 \left(\frac{\text{freq}(C_i, F)}{|F|} \right) \right] \tag{5.1}$$

calculates the average amount of information needed to identify its class where $C_1, C_2, \ldots, C_n$ represent the different classes and the notation $\text{freq}(C_i, F)$ refers to the frequency of class C_i in F.

After splitting the feature vector into the children using a test X with m outcomes, the average amount becomes

$$\text{info}_X(F) = \sum_{i=1}^{m} \left[\frac{|F_i|}{|F|} \times \text{info}(F_i) \right] \tag{5.2}$$

where F_i is the collection of feature vectors partitioned into child i.

With these values,

$$\text{gain}(X) = \text{info}(F) - \text{info}_X(F) \tag{5.3}$$

finally calculates the gain achieved by splitting on the test X.

In a predecessor to C4.5 named ID3, Quinlan used only this gain as its criterion for determining the best split. Unfortunately, notice from 5.3 that the gain

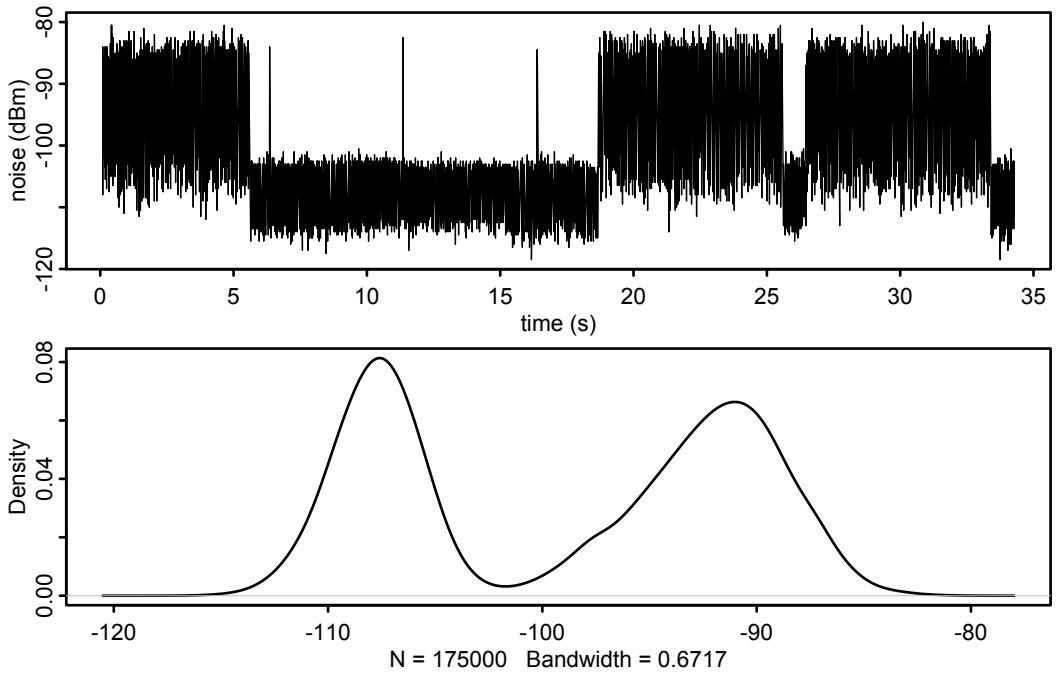


Figure 5.1: The trace from channel 94, port 10 (top) and its density plot (bottom). This trace has the shifting-mean pattern and its density plot is bimodal.

is maximized when $\text{info}_X(F)$ is minimal; if each outcome contains only a single feature vector then $\text{info}_X(F)$ will equal 0. To compensate for this problem, C4.5 uses a normalization of gain called the *gain ratio* when determining the best split. The gain ratio divides gain by the potential information generated by partitioning F into m subsets (one for each outcome)

$$\text{gratio}(X) = \frac{\text{gain}(X)}{-\sum_{i=1}^{m} \frac{|F_i|}{|F|} \times \log_2\left(\frac{|F_i|}{|F|}\right)} \tag{5.4}$$

In order to use this classifier, we extract a number of features (attributes) from our traces. The most obvious metrics include the (a) mean, (b) standard deviation, (c) standard deviation of the moving average, (d) skew, (e) kurtosis, (f) minimum, and (g) maximum. The moving-average standard deviation rather arbitrarily averages ten data points. We also include a number of percentiles: the (h) 95.0^{th}, (i) 96.0^{th}, (j) 97.0^{th}, (k) 98.0^{th}, (l) 99.0^{th}, and (m) 99.5^{th} percentiles. Finally, we include the (n) modal and (o) periodic features as explained in Sections 5.1.3 and 5.1.4.

The modal and periodic features address our desire to capture two specific characteristics in our data. For the shifting-mean traces, we observed that their density plots are rarely unimodal (Figure 5.1). For the rapid-spike traces, we observed that their high-frequency spikes tend to be very periodic (Figure 5.2). We later describe the features that account for these observations.

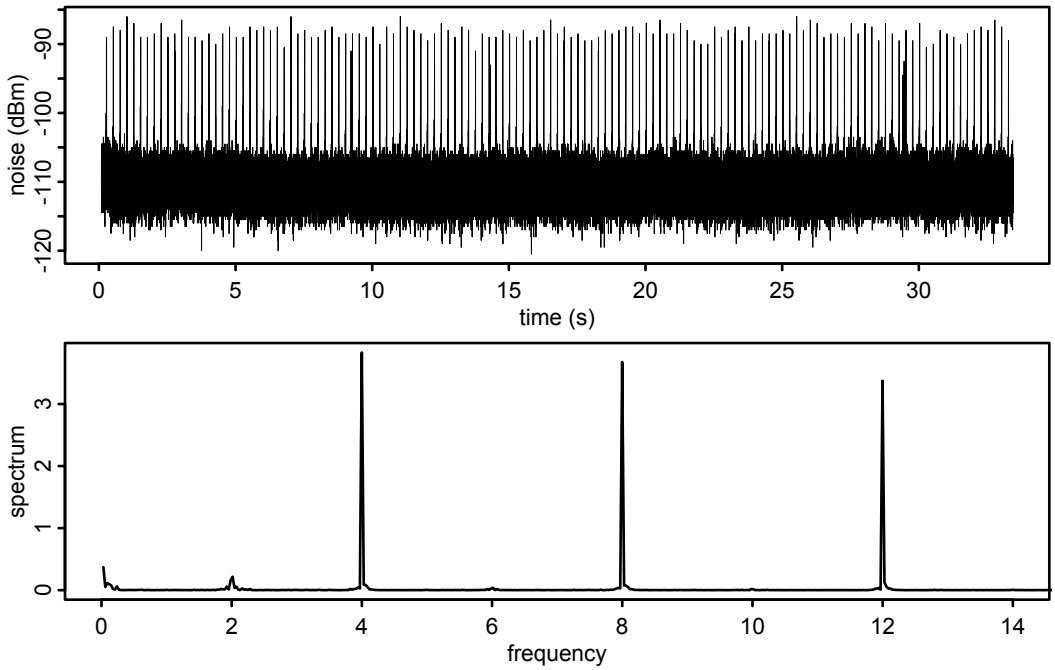


Figure 5.2: The trace from channel 69, port 12 (top) and its periodogram (bottom). This trace has the rapid-spikes pattern and its periodogram shows a significant response at 4 Hz and for the higher-frequency harmonics.

5.1.2 Subsampling

We derive the training set's feature vectors using the 1024 randomly selected traces from the 4096 traces that we collected. After summarizing these full 175 000-point traces into feature vectors, we use those vectors to build decision trees for each pattern of interest.

Given a built decision tree, we want deployed nodes to classify channels with fewer than 175 000 samples. These devices tend to have small memories, and as they collect more data, the computations also consume additional CPU cycles. Moreover, the data collection requires the transceiver to actively listen to the channel, which increases the node's energy consumption. For those reasons, we investigate classification performance on subsamples of our data. Since we passively collected traces, subsampling our data is equivalent to collecting the initial samples at a lower rate. If we had used an active approach that could affect the environment, e.g., packet transmissions, then this approach would be invalid.

With periodic (even) sampling, a constant time separates each sample ($\triangle t = t_{i+1} - t_i$ is constant $\forall i$). In this case, it is possible for the sampling to be synchronized with the trace's characteristic of interest – and all of the samples could miss it. For that reason, we compare periodic sampling with Poisson sampling, since the latter separates samples by random times and also maintains the distribution characteristics of the sampled distribution [25, pp. 327-328].

Suppose that we want to produce an even subsample of length n from a trace

of length N. We divide the trace into n segments of equal duration

$$\left[t_{\frac{0}{n}N}, t_{\frac{1}{n}N}\right), \left[t_{\frac{1}{n}N}, t_{\frac{2}{n}N}\right), \ldots, \left[t_{\frac{n-1}{n}N}, t_{\frac{n}{n}N}\right]$$

Within the first segment, between t_0 and $t_{\frac{1}{n}N}$, we randomly select a starting point t_s and its corresponding value (x_1) is the first in our subsample. For the times $t_{\frac{1}{n}N} + t_s$, $t_{\frac{2}{n}N} + t_s$, $\ldots$, $t_{\frac{n-1}{n}N} + t_s$, we then add their corresponding values ($x_2, x_3, \ldots x_n$) to our subsample, respectively.

To produce a Poisson subsample, we first determine the length of the subsample by drawing a random number from the Poisson distribution with $\lambda = n$. We then draw a simple random sample without replacement of that length from the trace.

To evaluate our results, we use the ϕ coefficient, which is actually a product-moment coefficient of correlation [48] and is also called the Matthews correlation coefficient [55]. It indicates the association between two variables, and as a normalized value, ranges from -1 to 1. Its value reflects the association between the correct classifications, true positives (TP) and true negatives (TN), and the incorrect classifications, false positives (FP) and false negatives (FN). We want a (high) positive value, which indicates a greater association between the true negative/positive values than the false negative/positive values. Although other methods can be used to evaluate classifiers, e.g., the sensitivity, specificity, precision, and recall percentages, the coefficient may provide a more balanced evaluation [10]. Unlike percentage-based measures, it uses (and can be calculated solely from) the four values from a confusion matrix

		Predicted	
		FALSE	TRUE
Actual	FALSE	true negative (TN)	false positive (FP)
	TRUE	false negative (FN)	true positive (TP)

using the formula

$$\phi = \frac{\mathrm{TP} \times \mathrm{TN} - \mathrm{FP} \times \mathrm{FN}}{\sqrt{(\mathrm{TP} + \mathrm{FP})(\mathrm{TP} + \mathrm{FN})(\mathrm{TN} + \mathrm{FP})(\mathrm{TN} + \mathrm{FN})}} \tag{5.5}$$

It has a direct relationship with the χ^2 statistic for a 2×2 contingency table

$$\phi^2 = \frac{\chi^2}{n} \tag{5.6}$$

and this relationship can be used to test significance levels (with 1 degree of freedom).

We evaluated C4.5-built decision trees for the two most disruptive channel classes: rapid-spikes and shifting-mean. We subsampled each of the 1024 hand-classified 175 000-point traces ten times for each subsample size and technique. The subsample sizes included 250, 375, 500, 750, 1000, 2000, 3000, 4000, 6000, 8000, and 12 000 points.

5.1.3 Dip statistic

Given that histograms of shifting-mean traces often show more than one mode (e.g., Figure 5.1), our search for a statistic to capture this fact led us to the dip statistic [41]. Without making any assumptions about the underlying sample distribution, the dip statistic allows us to test for unimodality. It does this by computing "the maximum difference between the empirical distribution function and the unimodal distribution function that minimizes that maximum difference" [41].

For a given p-value, the dip value depends heavily on the number of samples. Hartigan and Hartigan [41] provide values for sample sizes up to 200, and Martin Maechler[1] extended this table for sample sizes up to 5000. Given that our complete traces consist of 175 000 points, we again extended it.

To make this extension, we used code written by Martin Maechler, which was originally based on the published Hartigan code [42]. Our random uniform values of the null distribution came from the GNU Scientific Library [34] with the `gsl_rng_ranlxd2` generator and unique seed values for each run. We calculated values for both 50 000 and 175 000 samples at 26 different significance levels.

We next augmented our training set with dip values. We also introduced a new Boolean feature named *modal* that we set to true when we rejected the null (unimodal) hypothesis, i.e., when the dip statistic exceeded the threshold for a given significance level. To determine the best significance level, we calculated the *modal* attribute for each level, and in each case, looked at the agreement between the shifting-mean and modal attributes. The previously introduced ϕ coefficient gave us an indication of the agreement, and we obtained the maximum value for it at the significance level of 1.

For each of our 1024 hand-classified traces, we computed subsamples using both even and Poisson sampling, as previously described. When adding our Boolean *modal* attribute to the subsamples, we interpolated for the smaller sample sizes using values from our dip value table at the chosen significance level.

The process of evaluating a classifier's performance on subsampled data is, in a sense, end-to-end. It begins with subsampling traces, which corresponds to the data collection that nodes would perform. The classifier then operates on the samples, and the process ends with the classifier producing a result: positive or negative for the particular pattern.

We applied our decision tree classifier to the subsamples, and we found that the tree incorporating the modal attribute (which is based on the dip statistic) improves the classification performance across all the tested subsample sizes (Figure 5.3). The shifting-mean channels tend to remain stable at each mean for a reasonable amount of time, and thus, even a small number of samples can capture the two modes. As it was to be expected, without the modal attribute the classifier was very sensitive to the sample size and generally performed poorly. We did not see a significant difference between even and Poisson sampling in either case.

[1] See http://www.cran.r-project.org/web/packages/diptest/.

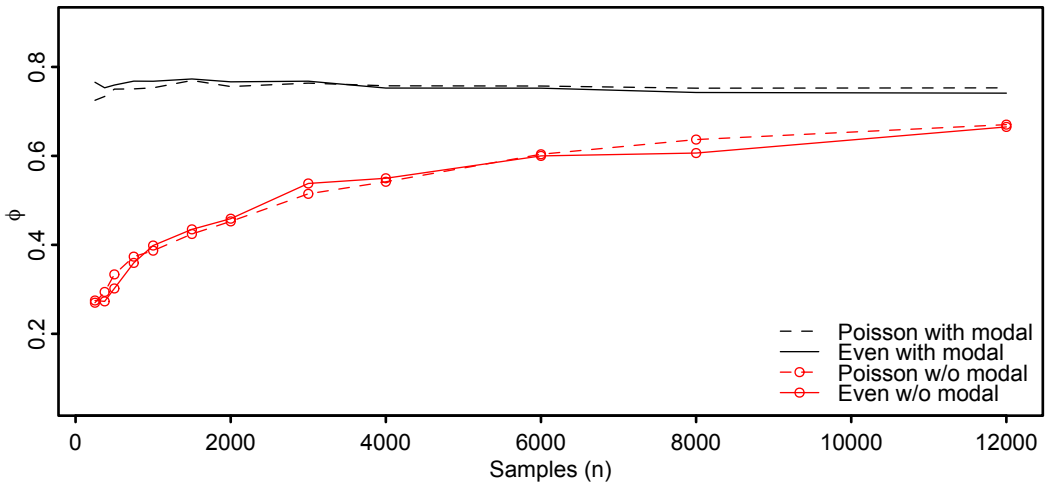


Figure 5.3: The effect of sampling technique and use of the modal attribute (derived from the dip statistic) on the classification of *shifting-mean* subsamples.

Beyond the classifier performance, we compared the shifting-mean decision tree both before and after adding the *modal* attribute. The tree without this attribute has 29 nodes, with 15 of them being leaf nodes (Figure 5.4). The tree that incorporates this attribute only has 15 nodes, with 8 of them being leaf nodes (Figure 5.5). This simpler tree has lower memory requirements and slightly faster classification performance. Also in the simpler tree, C4.5 selected the modal attribute for the first decision node, which lends further support to its effectiveness for detecting shifting-mean interference.

5.1.4 Lomb periodogram

In our plots of traces with rapid spikes, we noticed that the spikes appeared to have a very periodic nature (Figure 5.2). The first thought that occurred to us – Fourier transforms – does not apply when the spacing between samples is uneven. Given our desire to test Poisson sampling with its uneven spacing, we looked for alternative solutions. The Lomb periodogram is a technique of computing least-squares frequency analysis on unequally spaced data [50, 68]. In this subsection, we provide details for the algorithm because later, in Section 5.2.1, we will describe a simplification of it.

Given N data points, the periodogram's normalized form is defined by

$$P_N(\omega) \equiv \frac{1}{2\sigma^2} \left\{ \frac{\left[\sum_j (h_j - \bar{h}) \cos \omega(t_j - \tau)\right]^2}{\sum_j \cos^2 \omega(t_j - \tau)} + \frac{\left[\sum_j (h_j - \bar{h}) \sin \omega(t_j - \tau)\right]^2}{\sum_j \sin^2 \omega(t_j - \tau)} \right\} \tag{5.7}$$

where $\omega = 2\pi f$, f is the frequency, h_j and t_j represent the magnitude and time of sample j (respectively), $\bar{h}$ and σ^2 are the mean and variance of all the samples

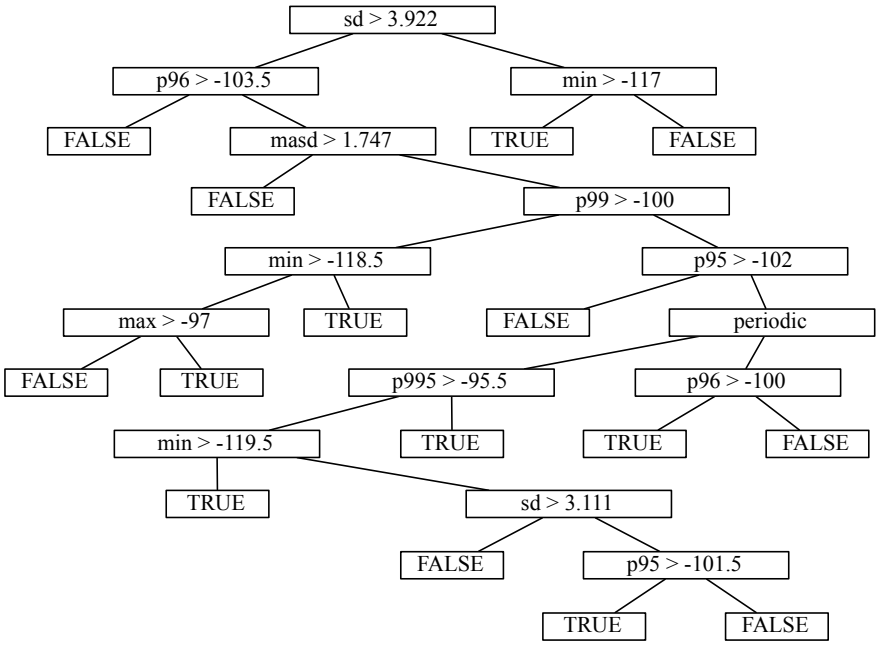


Figure 5.4: When we exclude the *modal* attribute, the C4.5 algorithm produces this tree to classify the shifting-mean channels. For each decision node, the left child is the false case. The Boolean value in the leaf nodes indicates membership in the shifting-mean class. This tree should be compared with Figure 5.5.

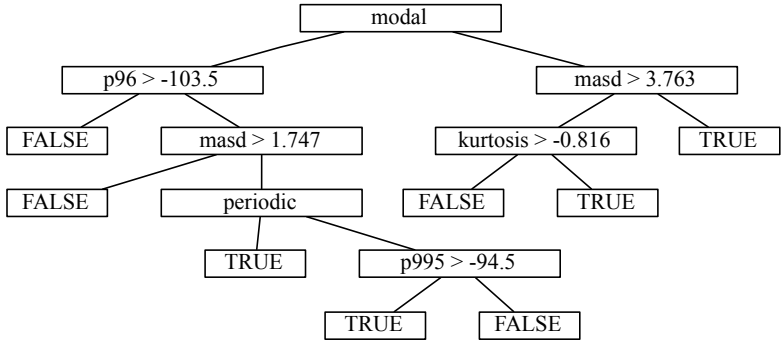


Figure 5.5: When we include the *modal* attribute, the C4.5 algorithm produces this tree to classify the shifting-mean channels. This tree should be compared with Figure 5.4.

(respectively), and τ is defined by the relation

$$\tan(2\omega\tau) = \frac{\sum_j \sin 2\omega t_j}{\sum_j \cos 2\omega t_j} \tag{5.8}$$

The intuition behind (5.7) is as follows: for each sample in the trace, multiply it by the sine and cosine waves modulated at the chosen frequency f. One of the two waves will capture any periodicity at f regardless of shifts in time. When the values $h_j - \bar{h}$ are randomly distributed about 0, i.e., the mean-adjusted signal level is essentially a Gaussian random variable with mean 0, this sum will tend to 0. When the values also modulate with the frequency, however, it will tend to a larger value. The final division at the end, by the sample variance, accounts for the width of the Gaussian distribution.

In our periodograms, we noticed that in addition to a peak at the fundamental frequency, additional peaks often occurred at multiples of that frequency. According to [58], the strong non-sinusoidal signal in our time series is responsible for these peaks, and the subsequent peaks are called harmonics.

We added a *periodic* attribute to our training set to summarize the periodogram in a single Boolean value. To determine when it should be true, we considered for each trace (a) the maximum periodogram value in one of the frequency ranges 2.5 to 3.5 Hz, 2.5 to 4.5 Hz, 2.5 to 5.5 Hz, and so on up 2.5 to 20.5 Hz and (b) the threshold that, when the above maximum periodogram value exceeds it, yields a maximum value for ϕ. Using a variant of simulated annealing implemented in R's `optim` function,[2] we identified a suitable threshold parameter for setting the periodic attribute when using the maximum frequency between 2.5 and 5.5 Hz.

As with the dip statistic, we subsampled our 1024 hand-classified traces, using both even and Poisson sampling, for the previously described subsample sizes and repetitions. Given that the periodogram is a sum across the (sub)sample, its magnitude depends on the number of samples in the sum. Therefore, when adding our new Boolean attribute to the subsamples, we had to scale the threshold proportionately.

We found that the periodic attribute derived from the Lomb periodogram produced significant gains in the rapid-spikes classifier performance (Figure 5.6). Given small sample sizes, classifiers both with and without the periodic attribute show low performance. As the sample size increases, the performance of the classifier incorporating the *periodic* attribute increases at a higher rate than the one without the attribute. Compared to the dip statistic, a larger sample size appears to be necessary for a response from the periodogram. Like with the dip statistic, we did not see a significant difference between even and Poisson sampling.

We compared the decision trees both with (Figure 5.7) and without (Figure 5.8) the *periodic* attribute. Prior to adding the attribute, the tree size is 67 with 34 leaf nodes, while with the attribute, the size is 15 with only 8 leaf nodes. It is evident that given the *periodic* attribute, we not only improve the classification performance but also obtain a simpler decision tree. Moreover,

[2] See `http://www.r-project.org` for more information.

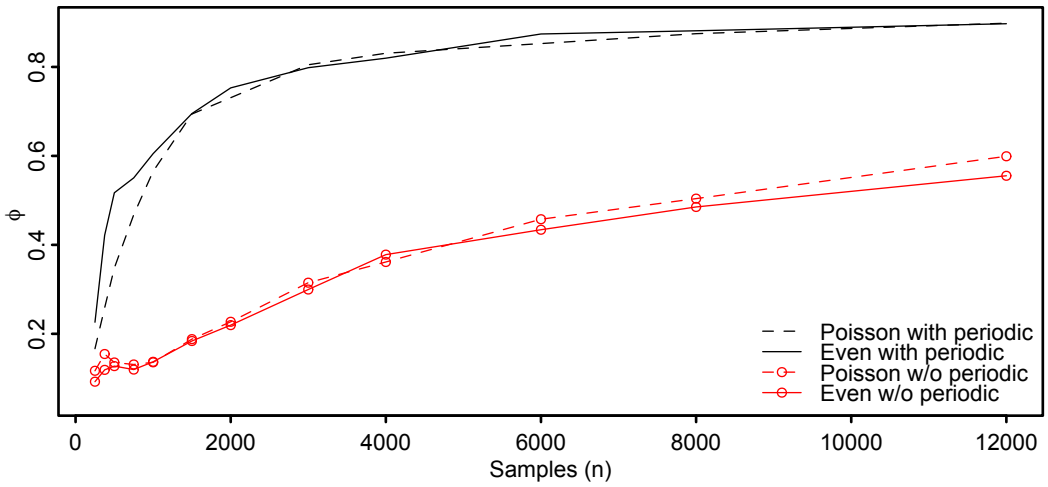


Figure 5.6: The effect of sampling technique and use of the periodic attribute (derived from the Lomb periodogram) on the classification of *rapid-spikes* subsamples.

as with the modal attribute, C4.5 selected the periodic attribute as the first decision node, which supports its use for detecting rapid-spikes interference.

5.2 Node-based realizations

In the previous section, we showed that the modal attribute (based on the dip statistic) and the periodic attribute (based on the Lomb periodogram) allowed the C4.5 algorithm to generate smaller decision trees that classified feature vectors with higher accuracy. When generating the earlier trees, the C4.5 algorithm had a wide range of attributes available to it, and the resulting trees included many of them. Computing those attributes on a wireless node, with the exception of the minimum, mean, and maximum, requires considerable memory and processing power. For in-node classification, we therefore explore smaller attribute sets to simplify the computations.

Recall that C4.5 builds trees from the top working down, and at each node, it selects the attribute that yields the purest children. In both the rapid-spikes and shifting-mean trees, our new attributes, periodic and modal, are the top attributes in their respective tree. For that reason, we first consider the performance of using just these attributes in a single-node decision tree.

We used C4.5 to generate a single-node decision tree based on the dip statistic alone, and it positively associated our modal attribute with the shifting-mean class as expected. Earlier, in Figure 5.3, we show that the dip statistic, when combined with other attributes, greatly improves performance for all our sample sizes. In this case, however, the dip statistic alone performs poorly compared with the full C4.5 tree, and this attribute should not be used alone to predict shifting-mean channels (Figure 5.9, dip only).

Given the poor performance of the dip statistic alone, we also consider combining it with other easily-generated attributes, i.e., minimum, mean, and max-

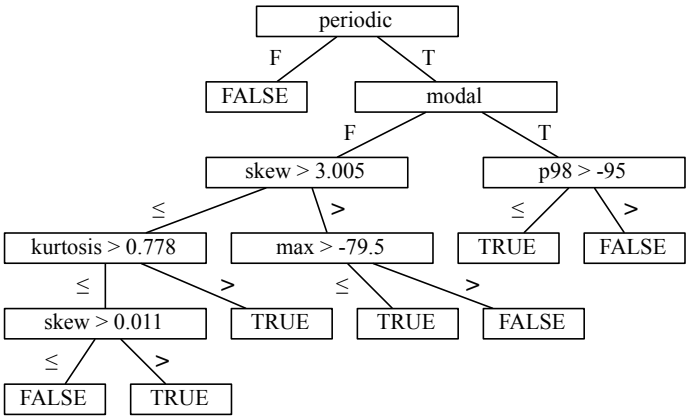


Figure 5.7: When we include the *periodic* attribute, the C4.5 algorithm produces this tree to classify the rapid-spikes channels. For each decision node, the left child is the false case. The Boolean value in the leaf nodes indicates membership in the rapid-spikes class. This tree should be compared with Figure 5.8, where we excluded the periodic attribute.

imum (Figure 5.9, dip+easy). In this case, we see improved performance at all sample sizes over the dip-only case, but for small sample sizes, the performance still significantly lags behind using the full attribute set; we do not find the dip+easy classifier's performance satisfactory until about 6000 samples. Given the high number of required samples, we abandon our current investigation of the dip+easy classifier. We therefore switch our focus to the rapid-spikes classifier.

Like with the dip statistic, we generated a single-node decision tree based on the Lomb periodogram alone (Figure 5.10). In this case, however, we observe that the single-decision tree performs very well for subsamples of at least 2000 observations. Apparently, this single attribute is responsible for most of the rapid-spike classifier's performance even at small sample sizes. In the next section, we focus on approximating the Lomb periodogram so that we can implement it within WSN-class devices. The actual implementation of the algorithm, however, will not occur until Chapter 6, at which point we investigate its use in identifying transmission opportunities in this class of channel.

5.2.1 `lombest`: an estimation of the Lomb periodogram

For a given frequency, the Lomb periodogram is defined by 5.7 and 5.8. In the exact case, the calculation

1. requires recording all of the samples, which is necessary to compute the mean and variance used in 5.7 and τ used in 5.8,
2. makes extensive use of the sine and cosine trigonometric functions,
3. uses floating-point values throughout, and
4. includes many multiplications and divisions (of floating-point numbers).

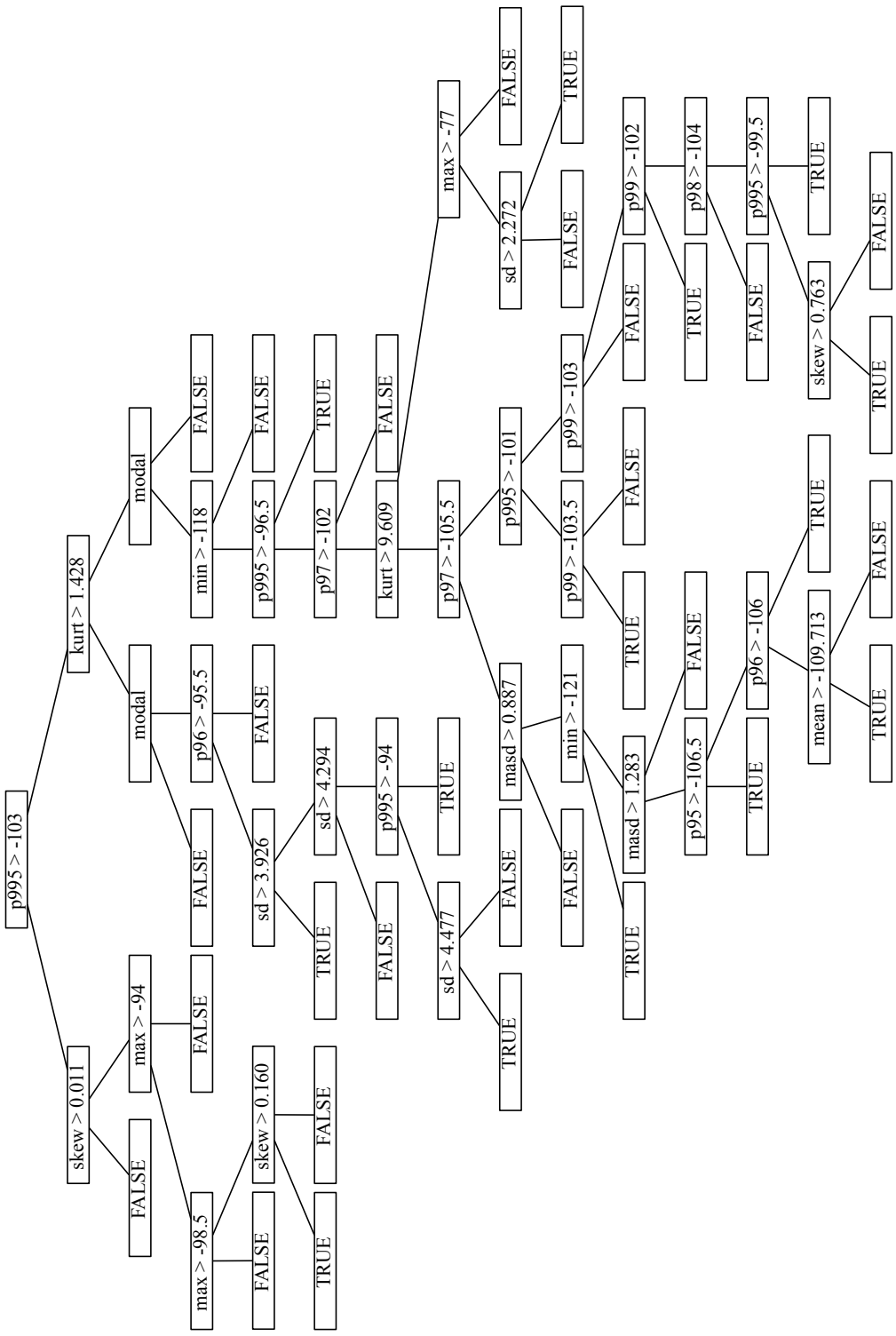


Figure 5.8: When we exclude the *periodic* attribute, the C4.5 algorithm produces this tree to classify the rapid-spikes channels. This tree should be compared with Figure 5.7, where we included this attribute.

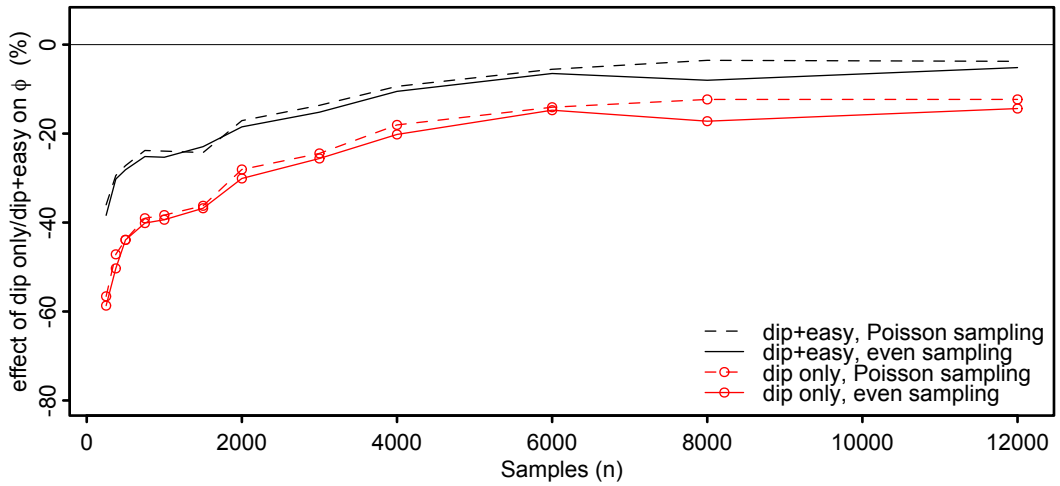


Figure 5.9: The effect of using (a) only the dip statistic (dip only) and (b) the dip statistic plus other easily-computed attributes (dip+easy) on ϕ for identifying *shifting-mean* channels.

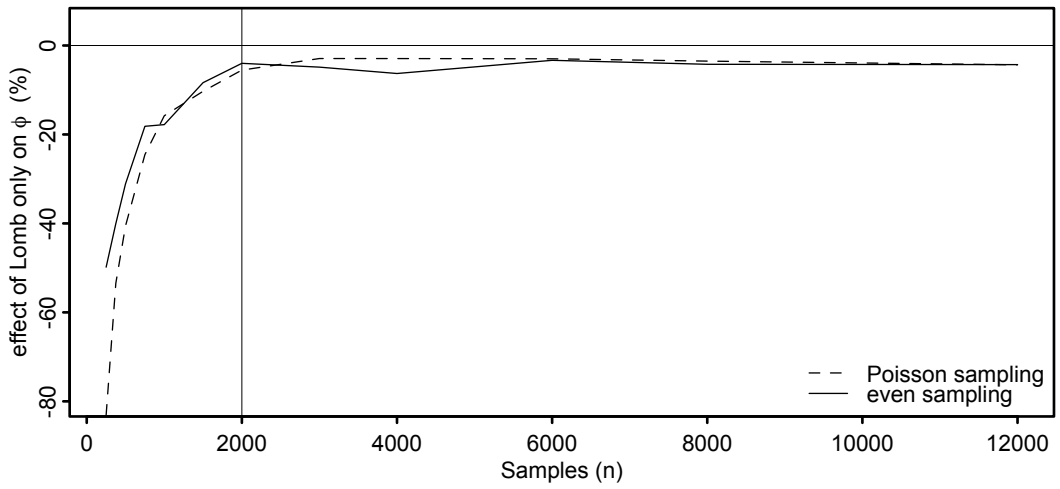


Figure 5.10: The effect of using only the Lomb periodogram on ϕ for identifying *rapid-spikes* channels. Note that after 2000 samples, the Lomb periodogram alone predicts the channels almost as well as with the full attribute set.

In the last case, note that one integer-float multiplication or division on the TI MSP430 uses over 400 instruction cycles[3], while a register add or shift operation requires only one cycle [88]. Given the limited memory and lack of hardware support for floating-point arithmetic and trigonometric functions in many microcontrollers, we need to consider approximations.

We aim for constant memory usage regardless of the sample size, and instead, we allow the usage to scale linearly with the number of analyzed frequencies. This setup allows us to collect a flexible number of samples, and at the same time, it encourages us to reduce the memory footprint by searching only frequencies of interest. To make the construction of the algorithm feasible within mote-class devices, we

- calculate and evolve an integer estimate of the mean,
- disregard the τ parameter, and
- quantize the sine and cosine waves with discontinuities selected to allow bitshifting.

The following paragraphs describe each compromise in detail.

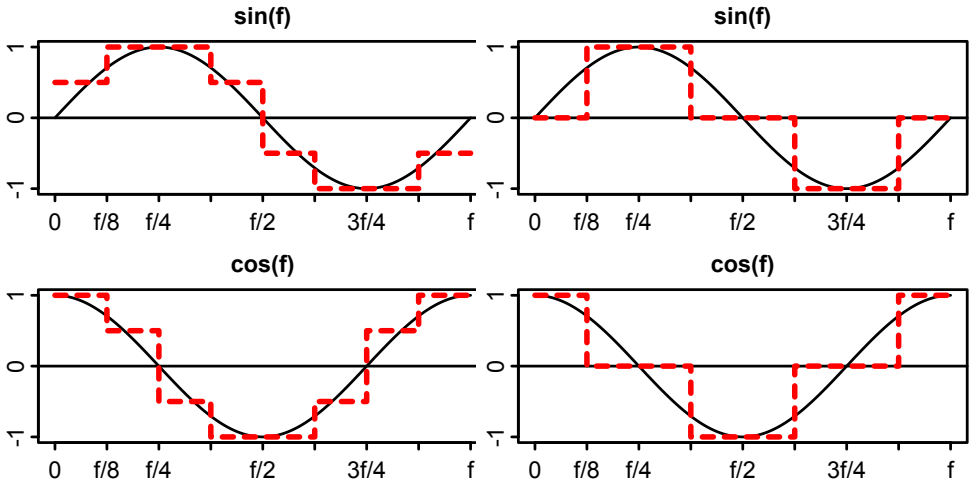
Note the use of the noise's mean $\bar{h}$ and variance σ^2 in 5.7, which in the exact case, requires the logging of all samples. Instead of taking this approach, we opt to (a) use an integer estimate of the mean that we calculate in advance using a small sample and (b) calculate the variance while collecting samples. To calculate the mean, we evaluated different sample sizes, and as expected, its value converges quickly as the sample size increases. At around 200 samples, we obtain a close approximation. Given a trace, we therefore use the first 10% for mean calculations and the remaining 90% for periodogram calculations. For example, at 2000 samples, the threshold for reasonable Lomb-only classification, the mean estimation uses 200 samples. To calculate the variance, we use floating-point numbers with one multiplication for each sample, and we perform the final divisions after accumulating this sum over the trace.

The mean plays a very important role in the algorithm, which assumes a random distribution of non-periodic samples around the mean. To reduce the risks associated with deriving the mean from a non-representative sample, we constantly adjust it: a sample above it increases it by 1 (and vice versa for samples below it).

The calculation of τ is another prerequisite for the exact periodogram. Lomb introduced this parameter to facilitate the statistical description of the least-squares spectrum [50]. For our approximation, however, we do not need such rigour, and intuitively, the periodogram changes little with shifts in time. We confirmed our suspicion by comparing several periodograms both before and after the removal of τ: we observed very little difference between the two cases.

The sines and cosines in the exact formulation oscillate at the frequency of interest. When calculating the periodogram value for a particular frequency, we first identify each sample's location in the sine and cosine waves and then multiply the sample's magnitude by the corresponding values in those waves.

[3]This cycle count assumes the C library included with IAR Embedded Workbench, version 3.41A.



(a) 4-level approximation (b) 3-level approximation

Figure 5.11: Alternative approximations of the sine and cosine functions used when calculating the Lomb periodogram.

We make a number of approximations here, both with identifying the sample's location in the wave and with the values of the waves.

To identify a location in a wave, we keep an array of time-stamps, one for each frequency of interest, that indicates when the next wave begins. For each new sample, we update this array, which requires an integer comparison for each frequency and possibly a few additions. By knowing when the next wave begins for a given frequency, we can perform subtractions and additions to identify the offset into the current wave. This technique allows us to avoid the alternative – the modulo operation – which would be more expensive.

By quantizing the wave and making the discontinuities occur at opportune locations (bitshifts of the frequency), we can easily compute the offset into a wave and avoid divisions. We experimented with two rather crude (but effective) approximations of the sine and cosine functions (Figure 5.11).

In the 4-level approximation, we use seven comparisons to determine the correct offset. Given an offset, we add/subtract the whole or (bitshifted) half mean-adjusted sample to a sum depending on the sine or cosine amplitude at that offset. We record two integer sums for each denominator ($\sum_j \cos^2 \omega(t_j)$ and $\sum_j \sin^2 \omega(t_j)$): one for whole numbers and one for quarters. Using this approximation requires 6 short (2-byte) integers per frequency of interest.

In the 3-level approximation, we use four comparisons to determine the correct offset rather than seven. Given an offset, we need only add/subtract whole mean-adjusted samples, and we now only encounter zero and one in the denominator. In this case, we only require 4 short integer sums per frequency.

In Figure 5.12, we compare the classification accuracy of the original and 4- and 3-level trigonometric approximations. Recall that the Lomb-only classifier begins to perform comparably to that of the full attribute set at 2000 samples. When using between 2000 and 4000 samples, the performance of our sine and cosine approximations perform reasonably well when compared with the origi-

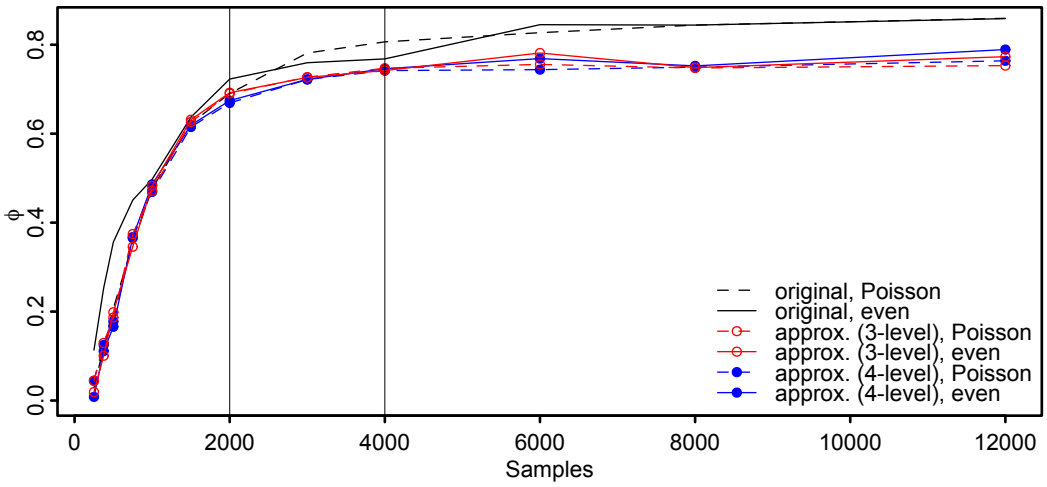


Figure 5.12: The performance of classifying *rapid-spikes* channels when using only the Lomb periodogram to produce the attribute. This graph compares using the original Lomb periodogram against approximations based on our 3- and 4-level trigonometric approximations. At 2000 samples, our Lomb-only rapid-spikes classifier performs nearly as well as the full classifier. After 4000 samples, the performance when using the approximation-based attribute remains steady.

nal. After 4000 samples, the performance of the original periodogram continues to slightly increase while the approximations remain constant. The 4- and 3-level approximations perform similarly across all sample sizes. Overall, the approximations perform very well when compared with the exact periodogram, with the 3-level being preferred given its lower requirements.

5.3 Summary

In this chapter, we described key work towards enabling WSN devices to characterize the interference in their environment. We consider observed patterns and macro-level descriptions of them (i.e., without knowing whether the interferers are adhering to a particular modulation and coding protocol, or even if they happen to just be noise sources). We focus on the most problematic classes of interference (in terms of their impact on packet loss rates) and develop classification mechanisms for them. Our main concern in this chapter is to generate accurate classification outcomes given little computation and energy resources, as is usually the case in wireless sensor networks.

In our work, we considered the energy and space constraints of a WSN node when selecting a classifier. We focused on decision trees, since after training, the classification of new cases is simple. In all of our experiments, we considered two different sampling techniques, even and uneven/Poisson sampling, and we had expected to find improved performance with the latter. Our results, however, showed no advantage when using Poisson sampling, and in hindsight, we are not surprised. Given the frequency of our sampling, it would be very

unlikely for them to synchronize with the rapid spikes and consistently miss the impulses.

We investigated the classification of samples according to distinct channel patterns: shifting-mean and rapid-spikes. Each of these can be exploited by cognitive radio avoidance strategies: in the former, changing channels or increasing the transmission power and in the latter, carefully timing transmissions. We found two features that aided in recognizing these patterns. For the shifting-mean channels, we exploited the sample's multi-modal distribution by using the dip statistic. For the rapid-spikes channels, we exploited their periodic nature by calculating the Lomb periodogram. In both cases, the new attributes improved the classification performance for all subsample sizes. We also observed that both of these new attributes reduced the size of their respective decision trees, particularly in the *periodic* case.

We then explored using one-node decision trees to classify the shifting-mean and rapid-spikes channels. The dip statistic alone was a poor predictor of shifting-mean channels. We managed to improve its performance by adding other easily calculated attributes (minimum, mean, and maximum), but it needed a rather large number of samples before its performance satisfied us. The Lomb periodogram, when used alone, performed quite well, particularly when collecting more than 2000 samples. Finally, we showed a simplification of the Lomb periodogram that could be implemented within the limited resources of a WSN-class device.

Given the positive results that we have presented here, the next chapter explores the *how* question – how we can use these channel classifications. Using the rapid-spikes classifier, we develop a pattern-aware medium access control (MAC) protocol that carefully schedules transmissions to avoid expected interference impulses.

Chapter 6

Exploiting Channel Classification

In Chapter 4, we described the interference and noise patterns that we encountered in our indoor urban wireless sensor network (WSN), and in Chapter 5, we explored classifying the two most disruptive of these patterns, shifting-mean and rapid-spikes. After showing that our classifiers performed well given the full attribute set, we reduced the available attributes to simplify the calculations and make them feasible for use by wireless sensor nodes. As we expected, these simplifications impacted the performance of our classifiers, but for the rapid-spikes one, we could still obtain very good results using only a single attribute. We then simplified the computation of this attribute to make it suitable for the limited resources of a wireless sensor node.

In this chapter, we describe the migration of our stand-alone classifier to our WSN programming paradigm and then explore the avoidance of the impulsive (rapid-spikes) interference within the framework of WSN devices. We show the effect of varying the packet arrival rate and packet length on the packet reception rate and latency. For these experiments, we isolate the variables of interest and eliminate a huge number of unknown and uncontrollable variables found in real environments by using simulation. In Section 6.1, we describe our extension to SIDE that generates impulsive interference. After modelling the interference, we incorporate the classifier and a proof-of-concept MAC into a WSN application (Section 6.2) and present the results from several simulations (Section 6.3). The positive simulation results led us to integrate the MAC into our EMSPCC11 wireless sensor nodes, and Section 6.4 presents the results from a small-scale deployment using hardware rather than simulation. Finally, in Section 6.5, we summarize our results.

Note that we are not the first researchers to incorporate interference into a WSN simulator: those working on closest-fit pattern matching (CPM) [49, 78] also use real-world observations to guide their channel modelling. Instead of focusing on specific patterns, they developed a modelling approach that initially replays a recorded trace and then estimates future points based on computed probabilities. Unfortunately, their probabilistic approach could easily lose the key periodic attribute that we are trying to recognize and subsequently exploit.

6.1 Interference modelling

In this section, we describe an extension to SIDE that allows users to define external impulsive interference. Its user-visible component consists of new attributes and tags for use within the existing XML configuration. Internal to the simulator, we implemented interference generation by adding a new node type, `Interferer`, and writing new processes that run on those nodes to produce the specified pattern. In Chapter 3, we described our development environment; in this section, we only describe our additions to it that allow us to model the interference.

Additional tags and attributes added to an existing XML (extensible markup language) configuration file provide the user-specified interference configuration. A new `interferers` attribute for the network tag indicates the number of interferers in the environment, e.g.,

```
<network nodes="40" interferers="3">
```

Within the `<network>` tag, a single `<interferers>` tag identifies the section for interferer-specific settings, akin to the existing `<nodes>` tag. Within this new `<interferers>` section, the user can define the parameters for each interferer, e.g.,

```
<interferer number="0" type="impulsive">
  <location type="random">170.0 170.0</location>
  <pattern>
    R 0.245 s         ; random delay
    P                 ; start periodic portion
    O 0.0 dBm 3 dB    ; on at 0.0 dBm with 3 dB sd
    T 0.005 s         ; delay
    F                 ; off
    T 0.245 s         ; delay then implicit jump to P
  </pattern>
</interferer>
```

In our current implementation, the only valid interferer type is `impulsive`, but in the future, we could add additional types. Internally, each interferer becomes an object within the simulation, much like what already happens for a node. These new objects execute a process specific to their associated interference type: for this work, we implemented an `Impulsive` process to simulate the pattern of impulsive interference specified within the `<pattern>` tag.

The body of the `<pattern>` tag essentially provides a script for the `Impulsive` process to follow. When SIDE parses the pattern, it extracts commands and arguments from the possibly commented description. It creates two arrays for these extracted values: one for the single-character commands and one for the double-valued arguments. For an impulsive interferer, SIDE supports the commands shown in Table 6.1. Essentially, the `Interferer` process interprets (in a fetch-decode-execute style) the command sequence provided in the specification block. Upon reaching the end of the command list, an implicit jump occurs to the command immediately following the `P` command, if specified; otherwise, it jumps to the start of the script.

Table 6.1: Commands supported by the `Impulsive` process.

Cmd.	Args.	Description
`R`	1	Delay the process for a random duration between 0 and the value specified in the argument (in seconds).
`T`	1	Delay the process for the specified duration (in seconds).
`O`	2	Generate interference at the specified power level (in dBm) with the specified standard deviation (in dB).
`F`	0	Stop the generation of interference.
`P`	0	Mark the start of the periodic portion of the pattern.

For the `<location>` tag, the attribute `type` with the value `"random"` causes SIDE to generate a new location every time the simulator starts (assuming a new seed for the random number generator). It uses the specified coordinates (e.g., `170.0 170.0`) as upper bounds for the random values.

We placed a number of synchronized impulsive interferers in a virtual environment, and using our earlier sampling application [15], collected a number of virtual traces (e.g., Figure 6.1, bottom). With very little tweaking, we were able to make the simulated traces match, in essence, the real traces. Upon close inspection, there were slight differences, e.g., the simulated traces lacked some random non-periodic components, and with a little more work, we could include these in our model as well. That said, the existing model's detail suffices for testing the classification and medium access control techniques in the following sections.

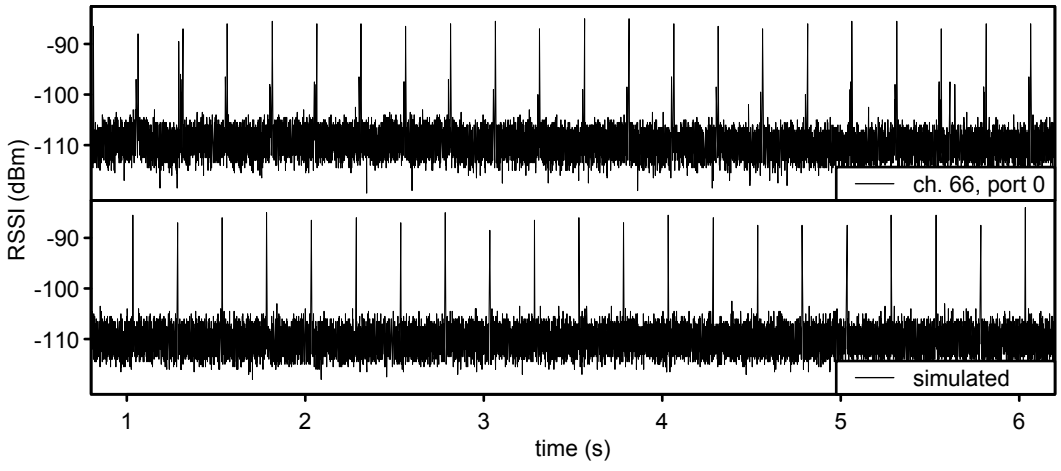


Figure 6.1: An actual trace (top) plotted with a simulated trace (bottom). We used the same application to collect both traces.

6.2 Implementing the classifier and a MAC protocol

PicOS includes a central driver, named the Versatile NETwork Interface (VNETI), that acts as the mediator between three primary components. Figures 3.1 (p. 27) and 3.2 (p. 29) show its location within PicOS and VUE^2, respectively, and Figure 6.2 shows additional VNETI components. The three primary components are

1. the application programming interface (API),
2. network protocol plug-ins, and
3. hardware drivers.

VNETI's primary responsibility in this framework is buffer and queue management. The following subsections describe it (and its three components) in greater detail so that readers can better understand our placement of the classifier and MAC within this framework.

6.2.1 VNETI

Networks are traditionally built around a stack of layers, where each layer provides a set of services via a clearly defined interface [86]. By isolating each layer and assigning it clear responsibilities, this approach allows for an overall reduction in design complexity at the cost of

- increased non-volatile memory usage for storing the additional function implementations,
- increased random access memory (RAM) usage for making function calls between layers and storing local variables at individual layers,
- increased processor usage for traversing the levels of abstraction intervening on the way from general-purpose high-level interfaces down to heterogeneous protocols and devices, and
- increased communication for transmitting data that is functionally duplicated in multiple layers, e.g., checksums.

In general-purpose computing environments, where the machines and operating systems are rather diverse and powerful, the costs are negligible and such layering has proven quite effective. On the other hand, these costs tend to outweigh the benefits in networks consisting of generally homogeneous (networking) hardware with scarce memory and limited processing capabilities.

The purpose of the Versatile NETwork Interface (VNETI) is to provide a simple collection of APIs, independent of the underlying I/O driver implementation, which, in addition to enabling a rapid deployment of networked application for microcontrollers, would make it easy to develop test beds using emulated I/O interfaces. To avoid protocol layering problems on small footprint solutions, the presented interface is essentially layer-less and its semi-complete generic functionality can be redefined by plug-ins. Also, the actual implementation of the physical interface can be encapsulated as a relatively simple and easily exchangeable module. This modularization allows us to easily compile a single PicOS application for a number of different hardware platforms. In a

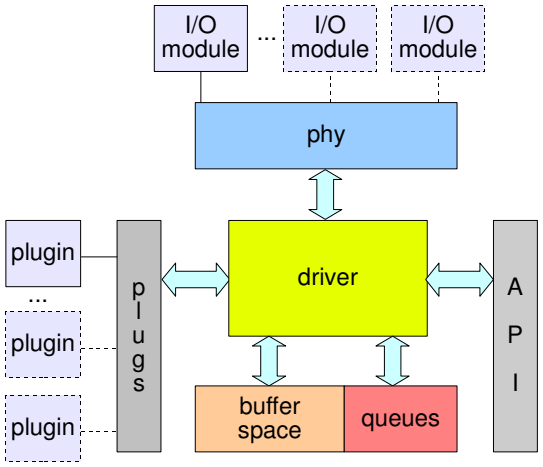


Figure 6.2: The Versatile NETwork Interface component in PicOS. This component serves as a mediator between the application programming interface (API), protocol plug-ins, and physical device drivers.

drastic departure from the layered approach, the plug-ins facilitate modularity and incorporate functionality that would, conceptually, span across many layers in a traditional layered design. For example, there is no restriction preventing plug-ins from consulting the packet's payload as well as its headers. Multiple plug-ins and physical interfaces can coexist within the same system configuration.

Figure 6.2 shows the internal structure of VNETI. In essence, VNETI implements (a) transparent management of buffer (packet) storage organized into a dynamic number of queues with timeouts definable on a per-packet basis, (b) multiple application access points (roughly equivalent to connections or sessions), and (c) a unified set of functions for interfacing plug-ins and physical modules. It acts as a mediator between the physical I/O modules, protocol plug-ins, and the application.

Physical network interface (phy)

The phy (physical) interface provides a standard set of APIs for attaching device drivers to VNETI. These drivers typically add support for RF networking devices; however, other I/O devices, e.g., the UART, can also be accessed via VNETI as honorary networking devices. The interface assumes that information written to/received from the device is packetized, in the sense that it is extracted and written in chunks obeying some specific requirements regarding their minimum/maximum size.

A phy module (device driver) registers itself with VNETI by calling the function

```
int tcvphy_reg (int phy, ofun_t ps, int info);
```

where `phy` is a logical (unique within the application) numerical identifier of the module, `ps` is a function that provides hooks for setting and querying some

standard options that all drivers must provide, and `info` is a globally unique information attribute used to identify the device handled by the driver. The function returns an event identifier corresponding to the event that will be triggered whenever the outgoing queue of packets associated with the driver becomes non-empty.

VNETI assigns a queue of outgoing packets to each registered phy module, and three functions allow the module to access this queue:

```
address tcvphy_top (int phy);
address tcvphy_get (int phy, int *len);
void tcvphy_end (address pkt);
```

The first function returns a pointer to the first (topmost) packet in the queue (or NULL if the queue is empty), while the second extracts the first packet from the queue. In both cases, the `phy` argument identifies the phy module, which should match the identifier assigned during the earlier registration phase. The phy module calls the third function when the packet, pointed to by the argument, has been transmitted, i.e., it is no longer needed by the driver.

The organization of a typical driver's transmission thread (FSM) is an event loop in which it examines the outgoing queue. If the queue is non-empty, the thread extracts and transmits the first packet; otherwise, it waits for the event whose identifier was returned by `tcvphy_reg`. Figure 6.3 shows a skeletal version of such an FSM, and we will reference this figure later when we discuss our implementation. The function call `start_transmission` represents the physical action of submitting the packet to the RF device for transmission, and `xmit_done` stands for the event triggered by the device when the transmission has been completed; it is typically signalled by an interrupt.

When sent over an RF channel, a packet is never physically addressed or encapsulated in any particular way, even if the device implements support for point-to-point transmission. At the phy level, packets are always broadcast and their contents are considered raw, i.e., the entire packet is treated as a sequence of bytes to be made available to the application or, more specifically, to VNETI plug-ins.

A driver also has a reception thread that typically waits for a hardware interrupt indicating a reception, reads received packets from the device, and provides those packets to VNETI. Alternatively, a single driver may combine both of these threads into a single thread. In either case, upon receiving a packet, the driver calls the function

```
int tcvphy_rcv (int phy, address p, int len);
```

which accepts the `phy` number, a buffer with the just-received packet, and its length in bytes. In contrast to the packet pointers commonly handled by VNETI, which typically point to packet buffers that can be put in queues, the second argument of `tcvphy_rcv` points to the raw sequence of bytes that have arrived from the device. After the driver calls this function, VNETI presents the newly received packet to the chain of plug-ins for the first step of its formal processing.

```
...
int event_id;
...
fsm driver {
  int len;
  address pkt;
  state LOOP:
    if ((pkt = tcvphy_get (0, &len)) == NULL) {
      when (event_id, LOOP);
    } else {
      start_transmission (pkt, len);
      when (xmit_done, DONE);
    }
    release;
  state DONE:
    tcvphy_end (pkt);
    proceed LOOP;
}
...
  event_id = tcvphy_reg (0, my_opts, 0xFECA);
...
```

Figure 6.3: The skeletal version of a physical I/O driver.

Application programming interface (API)

A workable VNETI setup involves at least one physical I/O module (phy) and at least one plug-in. Application interactions through VNETI deal with sessions which are logical entities with the flavor of socket file descriptors from UNIX. In contrast to such a UNIX file descriptor, however, a VNETI session need not (and usually does not) represent a communication peer. Instead, at the highest level, a session is an identifier that refers to some specific way of handling packets sent out or received by the application. The need for multiple sessions stems from the fact that the application may require diverse ways of handling different kinds of packets, in which case, it can have different sessions associated with a single networking interface.

An application establishes a session by performing three steps in sequence:

1. Initializing the phy. This is usually accomplished by invoking a function associated with the specific device driver, e.g.,

   ```
   phys_cc1100 (0, 62);
   ```

   which initializes the CC1100 transceiver driver as phy number 0 with a maximum packet length of 62 bytes. The function carries out the driver-specific initialization and invokes `tcvphy_reg`, where it registers itself with VNETI.
2. Configuring one or more plug-ins. This is accomplished by invoking

   ```
   int tcv_plug (int pl, const tcvplug_t *pl);
   ```

   which takes a logical plug-in ID as its first argument and a structure of

plug-in functions (described later) as its second. The plug-in ID fulfils a role similar to the phy ID for `tcvphy_reg`.

3. Opening a session by executing

   ```
   int tcv_open (word st, int phy, int pl);
   ```

   The function takes a state number, phy ID, and plug-in ID, and returns a new session ID. The state number is needed in those situations when the function may block, which depends on the plug-in. For example, setting up an elaborate session may involve exchanging packets with other nodes.

Once a session has been created, the application can acquire packets from it and/or write packets to it using the functions

```
address tcv_rnp (word st, int ses);
address tcv_wnp (word st, int ses, int len);
```

The second argument of both functions is the session ID.

For each session, VNETI assigns an incoming queue where it stores packets that are ready to be retrieved by the application. The application can acquire them with the first function, `tcv_rnp`, which will block when the queue is empty. Given its initial state argument, the scheduler will resume the invoking FSM in the indicated state when a packet arrives in the queue.

An application can use the second function to write a packet to the session, i.e., send it out, which happens in two stages. First, the application calls `tcv_wnp` to acquire a packet buffer of the specified length and associated with the indicated session. The function will block if no memory is available, and the scheduler will restart the FSM at the specified state when enough memory is freed elsewhere. Once the application obtains a buffer, it can fill the buffer with the required content. When done, the application calls

```
void tcv_endp (address packet);
```

to terminate the packet which, in this case, means that the packet is ready for transmission. The same function (`tcv_endp`) called for a packet retrieved with `tcv_rnp` deallocates the packet buffer, which an application will do when it has no further use for its contents.

An opened session can be closed by executing `tcv_close(s)`, where `s` is the session ID. Closing a session deallocates all packets and queues related to it.

Plug-in interface (plugs)

A data structure containing six function pointers describes a plug-in; its definition is

```
typedef struct {
   int (*tcv_ope) (int phy, int ses);
   int (*tcv_clo) (int phy, int ses);
   int (*tcv_rcv) (int phy, address buf, int len, int *ses);
   int (*tcv_out) (address pkt);
   int (*tcv_xmt) (address pkt);
   int (*tcv_tmt) (address pkt);
} tcvplug_t;
```

The first two plug-in functions, tcv_ope and tcv_clo, handle the plug-in-specific actions related to the session open and close operations (tcv_open and tcv_close). Their first argument is the phy ID (provided to tcv_open), and the second is the session ID (allocated by tcv_open).

The remaining four functions allow packets to interact with the plug-in, i.e., they are invoked when VNETI needs a plug-in-specific action (or merely a decision) regarding a packet reaching some meaningful stage of processing. Each of these functions returns a so-called disposition code representing an action or decision, and possible values for the code include

TCV_DSP_PASS meaning skip or do nothing, depending on the context.

TCV_DSP_DROP meaning that the packet should be dropped and its buffer deallocated and returned to the free memory pool.

TCV_DSP_RCV meaning that the packet should be queued for reception at the session with which it is associated. A packet may be classified as urgent,[1] in which case it will be queued at the front of the session's queue; otherwise, it will be queued at the end.

TCV_DSP_RCVU meaning that the packet should be marked as urgent and queued for reception at the session (necessarily at the front of the queue).

TCV_DSP_XMT meaning that the packet should be queued for transmission by the physical module with which it happens to be associated. If the packet is urgent, it will be queued at the front of the module's outgoing queue; otherwise, it will be queued at the end.

TCV_DSP_XMTU meaning that the packet should be marked as urgent and then queued for transmission by the respective physical module.

Function tcv_rcv is an exception among the four packet-related plug-in functions; it operates on raw packet contents (buf) rather than a packet buffer (as the remaining three functions). The function is invoked by tcvphy_rcv to (a) determine whether the plug-in should claim the newly received packet and (b) assign the packet to a session (the session ID is returned via the fourth argument). Value TCV_DSP_PASS returned by tcv_rcv is interpreted as an indication that the present plug-in does not want to claim the packet, i.e., the packet does not fall under its jurisdiction. Another way to express that would be to say the packet does not belong to the protocol handled by the plug-in, except that we prefer to avoid drawing unnecessary boundaries or assigning things rigidly to non-existent drawers. Note that the function makes this decision based on the packet's entire content (there are no predefined fields in the packet whose meaning would be universal). Consequently, the interpretation of such a decision may be more general. For example, multiple plug-ins may implement the same protocol (whatever that means) and operate as a chain of rules whose purpose is to diversify the processing of packets depending on some differences in their contents.

[1]This is a buffer attribute used internally by VNETI. Packets as such have no explicit predefined attributes stored in their contents. Thus, the (internal) urgent attribute is never transmitted along with the packet, but is assumed to be derivable from its contents by the plug-in.

If the application defines multiple plug-ins associated with the same phy (see function tcv_plug), all of them are scanned upon the reception of every packet via the phy, and the first plug-in whose tcv_rcv returns something different from TCV_DSP_PASS claims the packet, i.e., the scanning stops there. For the purpose of this scanning (which is the only situation when that matters), the plug-ins are assumed to be arranged in the reverse order of their association by tcv_plug, i.e., the last-associated plug-in is scanned first. Thus, the first plug-in can be viewed as a fall-back (or default) plug-in. If no plug-in claims the packet (all instances of tcv_rcv return TCV_DSP_PASS), the packet is dropped. The effect is the same as if one of those functions returned TCV_DSP_DROP.

Any buffered packet, i.e., any packet handled by VNETI following the reception/acceptance stage represented by tcv_rcv, is always assigned to a specific session. For a packet received from the network, this assignment is done by tcv_rcv. For a packet created by the application (e.g., with tcv_wnp), the assignment is clear from the very beginning.

Note that tcv_rcv has no obligation to assign the packet to the reception queue of the session (which would happen if the function returned TCV_DSP_RCV or TCV_DSP_RCVU). In particular, it may decide to drop the packet altogether or even direct it to the transmission queue (by returning TCV_DSP_XMT or TCV_DSP_XMTU), in which case the received packet will be immediately queued for retransmission by the node without ever arriving at the application. The plug-in functions have access to several operations of VNETI, which allow them to create new packets, assign them to sessions, clone packets, and so on. Thus, it is easy, for example, to pass a received packet to the application and simultaneously queue a (possibly modified) copy for retransmission.

Regarding the remaining plug-in functions, tcv_out is called for a packet that has just been submitted by the application, typically by tcv_wnp[2] (or rather by tcv_endp following tcv_wnp). Function tcv_xmt is invoked for a packet that has just been transmitted by the phy, an event that occurs when the phy driver calls tcvphy_end.

A plug-in function may set up a timer for any packet that has been stored in one of VNETI's buffers. This timer is in addition to the packet's normal path through the system, which means that a packet may be present in one of the standard queues (e.g., in the phy transmission queue waiting for transmission) while also waiting for its timer to go off. When the timer goes off, tcv_tmt will be called for the packet. As with the other packet-related plug-in functions, the value returned by tcv_tmt indicates the packet's subsequent fate.

6.2.2 Classifier integration

The division of VNETI into three clear interfaces allows us to implement the classifier (and MAC) transparent to the application. As described in the preceding section, the plug-in interface provides an abstraction for packets. The classifier and the MAC, on the other hand, must interact with the hardware and both have a number of precise timing constraints. Therefore, it is most appropriate to implement our work within an I/O module, as shown in Figure 6.2

[2]There is an urgent variant of the function called tcv_wnpu.

and exemplified in Figure 6.3. For the results presented for the simulator, we implemented both components in VUE^2's emulated radio driver, and both components make full use of the PicOS finite-state machine paradigm.

To classify channels with periodic short-duration impulsive interference, we implement the approximated least-squares spectral analysis (LSSA) technique described in Section 5.1.4. In the transceiver's transmit FSM, we introduce three new states to accommodate the classifier

> `CLS_INIT` to initialize the variables required for classification. After initializing them, this state immediately advances to `CLS_MEANEST`.
>
> `CLS_MEANEST` to obtain a single RSSI sample for inclusion in the mean estimate. It remains in this state for 200 iterations prior to transitioning to `CLS_SAMPLE`. This number of iterations proved reasonable in our early tests. The delay between each iteration is uniformly randomly distributed between 0 and 7 ms.
>
> `CLS_SAMPLE` to obtain a single RSSI sample for calculating the LSSA. It remains in this state for 5000 iterations (approximately 17.5 s) and then transitions to the (regular) MAC state. The delay between each iteration is uniformly randomly distributed between 0 and 7 ms. We used more iterations than the 2000-4000 identified in Chapter 5 simply as a precaution.

The complete classification process lasts just over 18 s during which we prevent nodes from communicating. If the application wishes to transmit packets during the process, they are simply queued within VNETI until the nodes complete the classification process. It is implied that in a real deployment, the classification task is to be executed occasionally to assess the new levels and periods of any periodic impulse interference.

We produce a rough estimate of the classification cost considering (a) the energy consumption of the microcontroller and transceiver in various states and (b) the time spent in each state during classification (from measurements). According their respective data sheets,

- an active MSP430F1611 (4.5 MHz) consumes 2.25 mA,
- an idle CC1100 consumes 1.6 mA,
- a CC1100 in receive mode (915 MHz, 38.4 kbaud) consumes 15.2 mA, and
- a CC1100 in transmit mode (10 dBm) consumes 28.9 mA.

Assuming three volts, a node collects 200 samples at 319 μs per sample and consumes 3.3 mJ for the initial mean estimation. The classifier should achieve reasonable performance with 1800 samples at 583 μs per sample, with the node consuming 54.9 mJ.[3] Finally, a node must make some final calculations for the periodogram: in our case, nodes tested for 21 frequencies, took 23 ms, and consumed 0.2 mJ. For the remaining 17 s elapsed between samples, the microcontroller can sleep (with negligible energy consumption) and the transceiver can remain idle, consuming 81.6 mJ. This total cost of 140.1 mJ is comparable to transmitting 52 average-sized Smart Condo packets at 10 kbps.

[3]Although our actual implementation used 5000 samples, our earlier work suggested that 2000 samples (200 for the mean estimation and 1800 for the periodogram) should suffice.

6.2.3 Pattern-aware medium access control (PA-MAC)

The output from the classifier indicates the presence of periodic short-duration impulsive interference along with its frequency (from those tested). Our proof-of-concept pattern-aware MAC (PA-MAC) then uses this output in its attempt to steer transmissions around the impulses. The periodic interference provides well-defined timing points around which nodes anchor transmissions, with some back-off, of course, as we will later describe. Stretching definitions a bit, the interference becomes a means for implicit synchronization of the MAC transmissions across nodes.

Although our technique responds to the interference by navigating packets around impulses, other possible approaches exist. For example, a transceiver could use a forward error correction technique such as a repetition code, convolutional code, or turbo code [79]. That said, our technique (PA-MAC) does not consider coding techniques.

In our approach, we make observations about the interference at a transmitting node and assume that they also hold for the receiving node, i.e., we assume a significant amount of correlation in the interference between nodes. Particularly in dense deployments, we have found this assumption to hold, e.g., we observed significant correlation in the traces collected in the Smart Condo (Figure 6.4). Moreover, other researchers have observed significant correlations in packet losses [83]. Even in larger environments, this assumption may hold given either a particularly strong interferer or a collection of correlated interferers.

To implement PA-MAC, we introduce one further state to the transceiver's transmit FSM, `MAC_SEARCH`, and use it to track the impulse instants. Initially after the classification, and then again regularly after each transmission window, the process will enter this state to sample the channel and search for the next impulse. Successfully finding an impulse causes the thread to (a) set a timer to mark the end of the next transmission window, i.e., the expected arrival of the next impulse and (b) delay for the expected duration of the currently identified impulse and then transition to the thread's preexisting primary state (`XM_LOOP`). Once in the main loop, the driver will retrieve outgoing packets from VNETI as they become available and transmit them until the expiration of the first timer. At that point, the process reenters `MAC_SEARCH` where it attempts to track the next impulse.

Without PA-MAC, multiple independent transmitting nodes have little opportunity to synchronize: the random back-offs effectively resolve contention. With PA-MAC, however, our regular tracking of the interference introduces a new opportunity for nodes to synchronize, which could ultimately cause a number of nodes to transmit at the same time. In our initial experiments with PA-MAC, we overlooked this possibility and experienced a high number of packet losses. Upon investigating those losses, the reason became blatantly obvious: a number of nodes would transmit immediately after an impulse, all at the same time. We eliminated this point of contention by introducing an additional random back-off, and we immediately saw the benefits in our results.

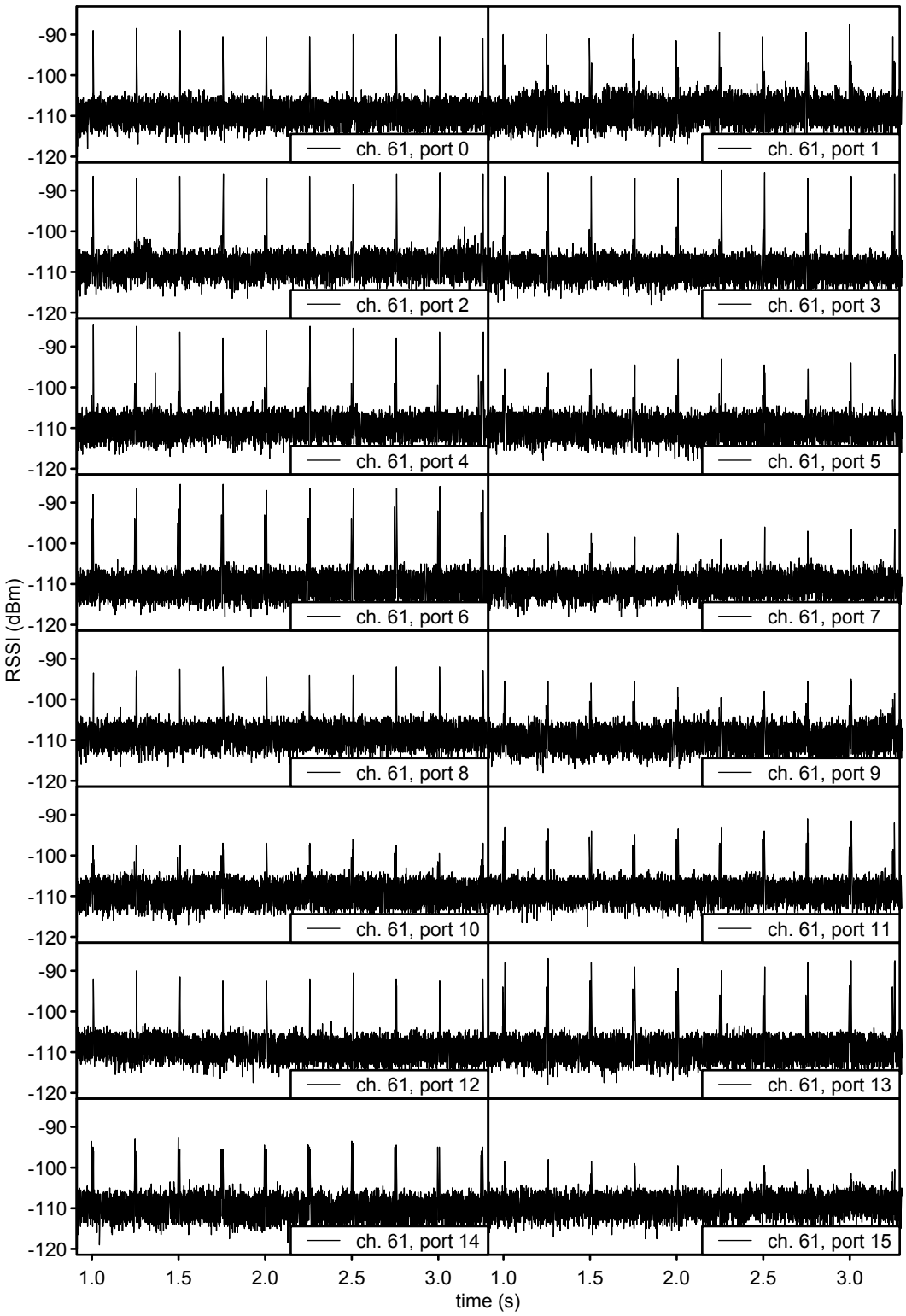


Figure 6.4: Traces collected on channel 61 (916.197 MHz) from 16 different locations simultaneously. Note that the impulses occur at all of the receivers.

6.3 Simulation results

We evaluated the pattern-aware MAC within the interference-generating SIDE simulator. We used its built-in shadowing channel model, and we tweaked its parameters to represent our hardware.

We compare PA-MAC against the existing PicOS non-persistent carrier sense multiple access (CSMA) MAC that listens before talking (LBT) and resolves contention using a random back-off. Such CSMA-family protocols are quite common for WSNs; e.g., TinyOS, a particularly popular WSN operating system, now uses a CSMA protocol named B-MAC [67]. We decided not to compare against a MAC that incorporates collision avoidance, i.e., CSMA/CA, given the additional overhead of its request-to-send and clear-to-send packets.

We include results for both single- and multi-hop random topologies. For each tested configuration, we average the measurements from 100 different topologies, each with its own traffic pattern, and plot the results with 95% confidence intervals.

6.3.1 Single-hop

The single-hop configurations consist of 19 source nodes, one destination node, and two interferers, and the simulator places them all randomly within an 18 m $\times$ 18 m field. Since the model neither includes obstructions nor considers radio irregularity [102], these dimensions guarantee that the destination is within the transmission range of every source node. Each node transmits at a rate of 10 kbps and a power of -20 dBm. The interferers introduce 5 ms pulses of impulsive interference at 4 Hz and -30 dBm, and both interferers are synchronized.

We first evaluated the effect of varying the packet length on the latency and packet reception rates (PRRs) (Figure 6.5). Note that the destination node does not acknowledge received packets, and nodes make no attempt to retransmit lost packets. We measure latency from the application perspective: the time that elapses between VNETI receiving the packet from the application (at the transmitter) and the application receiving the packet from VNETI (at the receiver). Note that the effect of varying the packet length is similar to the effect of varying the impulse frequency. In these tests, nodes generate new packets according to an exponential distribution with a mean of 20 s to reduce (if not practically eliminate) the effect of congestion. For each run of the simulation, we generate 469 s of transmissions and allow the simulator to run for 600 s to ensure that all source nodes empty their transmission queues.

When increasing the length, the PRR decreases for all configurations and the latency increases (as expected). In terms of PRRs, PA-MAC performs similarly to the quiet configuration because it successfully steers the transmissions around the interference. To obtain these PRRs, it delays transmissions that may collide with interference, and the latency graph reflects this behaviour. The traditional LBT MAC's PRRs suffer at a greater rate than the other two configurations as more packets are lost to collisions than simply the link's non-zero bit error rate. The traditional LBT MAC shows higher latency than what

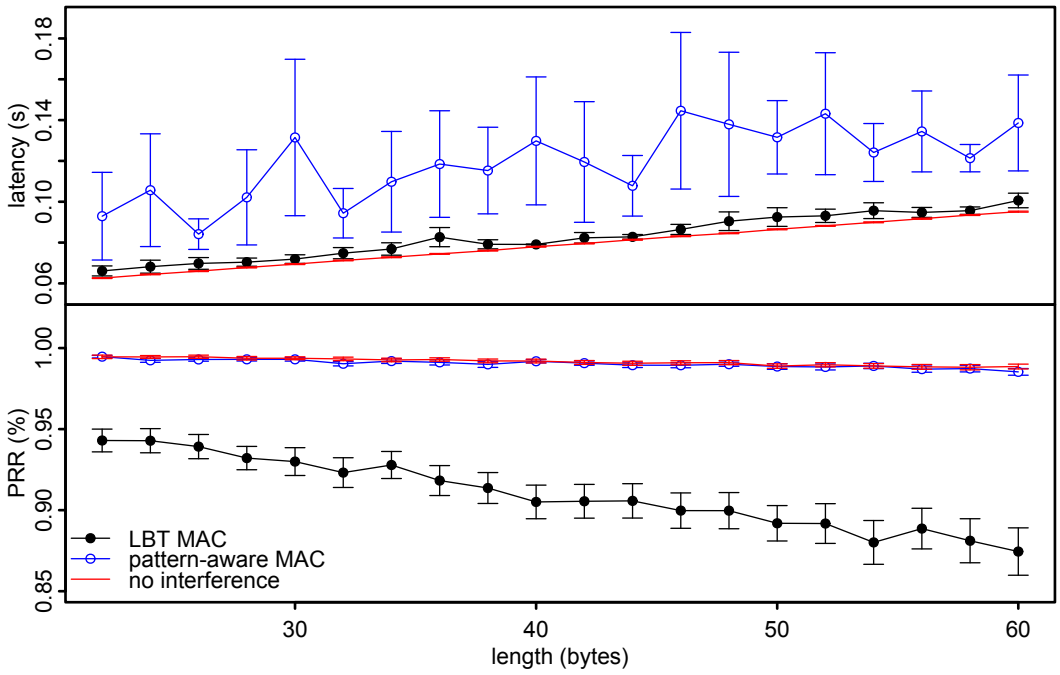


Figure 6.5: In a dense single-hop network, the effect of varying the packet length on the latency (top) and packet reception rate (bottom).

we observe with the quiet channel, demonstrating that it also yields occasionally to interference because it senses the medium as being busy.

We also evaluated the effect of varying the per-node packet inter-arrival rate on the latency and PRRs (Figure 6.6). In these experiments, we set the packet length to its maximum (60 bytes) in order to accentuate the effect of varying the inter-arrival rate.

Under high congestion, where the mean per-node packet inter-arrival rate μ is less than 1.3 s, the packet reception rates drop significantly for all methods in this dense network, and the LBT MAC and PA-MAC perform very similarly in terms of their PRRs. Under particularly high levels of congestion ($\mu < 1.1$ s), nodes must drop packets as their transmission queues fill; these full queues cause the substantial increase in the latency and the sudden drop in the reception rates for all techniques at 1.1 s. To ensure that nodes can empty their queues before the simulator terminates, we increase the duration of each simulation to 700 s from the previous 600 s, while still generating 469 s of transmissions. At lower levels of congestion, the PA-MAC tends towards the performance of the interference-free configuration.

6.3.2 Multi-hop

The multi-hop configurations consist of 39 source nodes, one destination node, and three interferers, and the simulator places them all randomly within a 170 m $\times$ 170 m field. As with the single-hop scenario, each node transmits at a rate of 10 kbps and a power of -20 dBm. In this case, the three interferers

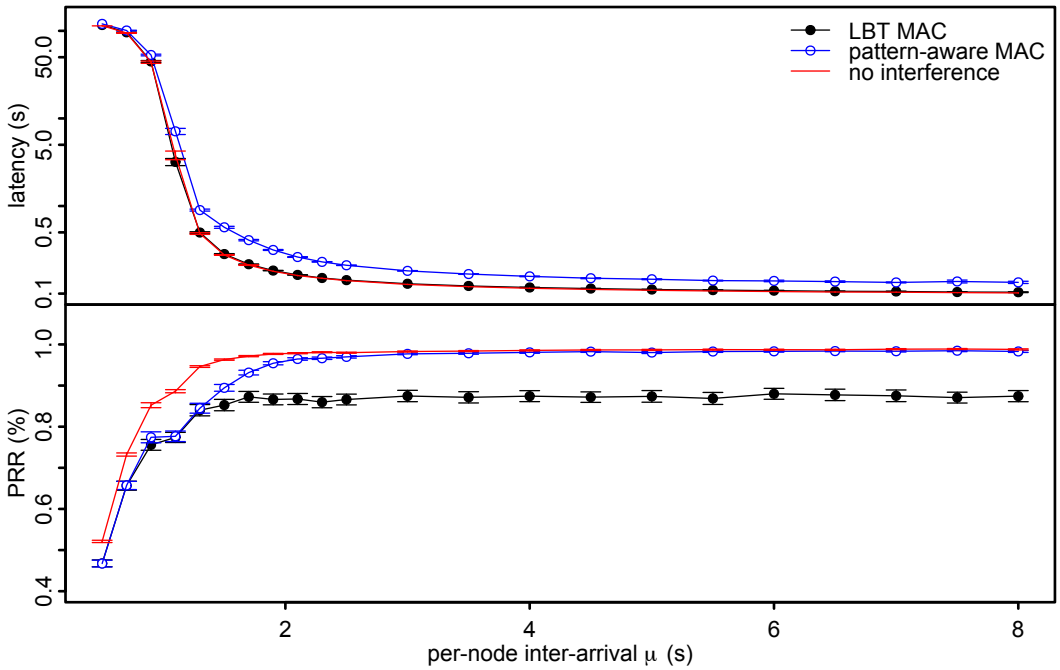


Figure 6.6: In a dense single-hop network, the effect of varying the per-node packet inter-arrival rate on the latency (top) and packet reception rate (bottom).

produce the same synchronized interference pattern as in the single-hop case, but transmit at 0 dBm rather than -30 dBm to better ensure that interference reaches the extents of the environment.

The nodes all use the tiny ad hoc routing protocol, TARP [61], to deliver packets to the destination. TARP is a light-weight on-demand routing protocol that quickly converges to the shortest path in static networks. It does not inflate the overall traffic needed to support it because it lacks explicit control packets and requires minimal control information in the packet header. Although our test application only demands one-way communication, the destination sends short 14-byte replies to each source node for the benefit of the routing protocol. These acknowledgements allow the routing protocol to limit the extent of its flooding, which should, in turn, reduce the congestion in the network. Note that communication continues to be unreliable, and nodes make no attempt to retransmit lost packets in the absence of an acknowledgement.

In order to use TARP within a dense network of this size, we tweaked a number of its parameters. We increased the size of its *duplicate discard* cache from 10 to 100 and its *shortest path discard* cache from 20 to 200. We also enabled its broader definition of a duplicate packet where those sequence numbers "in the shadow of" (a few behind) a cached number are considered to be duplicates.

Given random node locations, we need to take precautions to ensure that each source node has a path to the destination node. Immediately after generating a random layout, we extended the simulator to search for a path from every source to the single destination while ensuring that each hop is less than

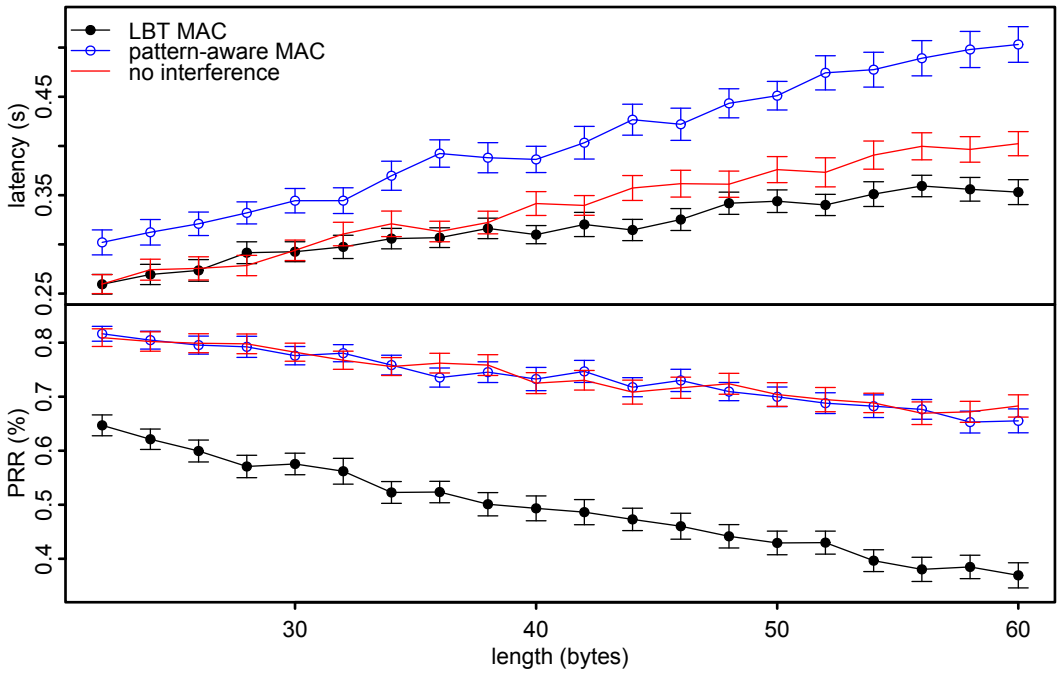


Figure 6.7: In a connected multi-hop network, the effect of varying the packet length on the latency (top) and packet reception rate (bottom).

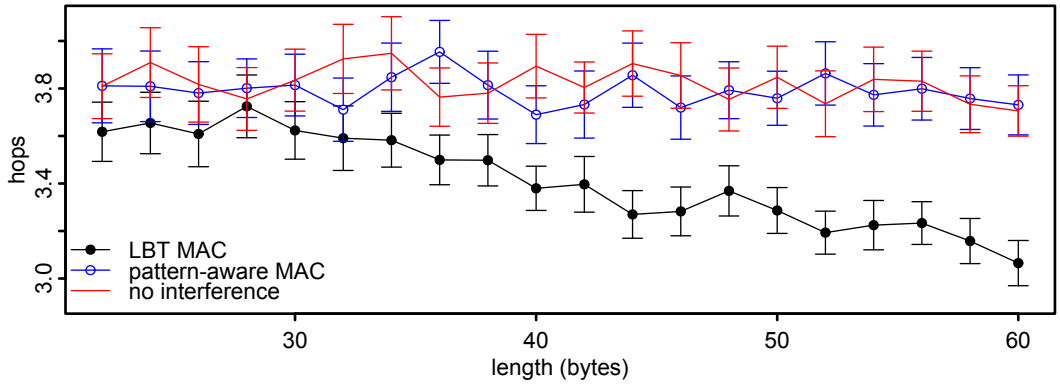


Figure 6.8: In a connected multi-hop network, the effect of the packet length on the mean number of hops.

the maximum transmission range. If the procedure finds a disconnected node, the simulator regenerates the placement of all nodes until every node is connected.

As in the single-hop case, we first evaluate the effect on varying the packet length on the latency and PRRs (Figure 6.7). To reduce congestion in the multi-hop environment given the high initial number of retransmissions, we lower the packet generation rate to follow an exponential distribution with a per-node packet inter-arrival rate of 200 s. Given the lower generation rate, we generate 1969 s of input and allow the simulator to run for 2100 s before terminating.

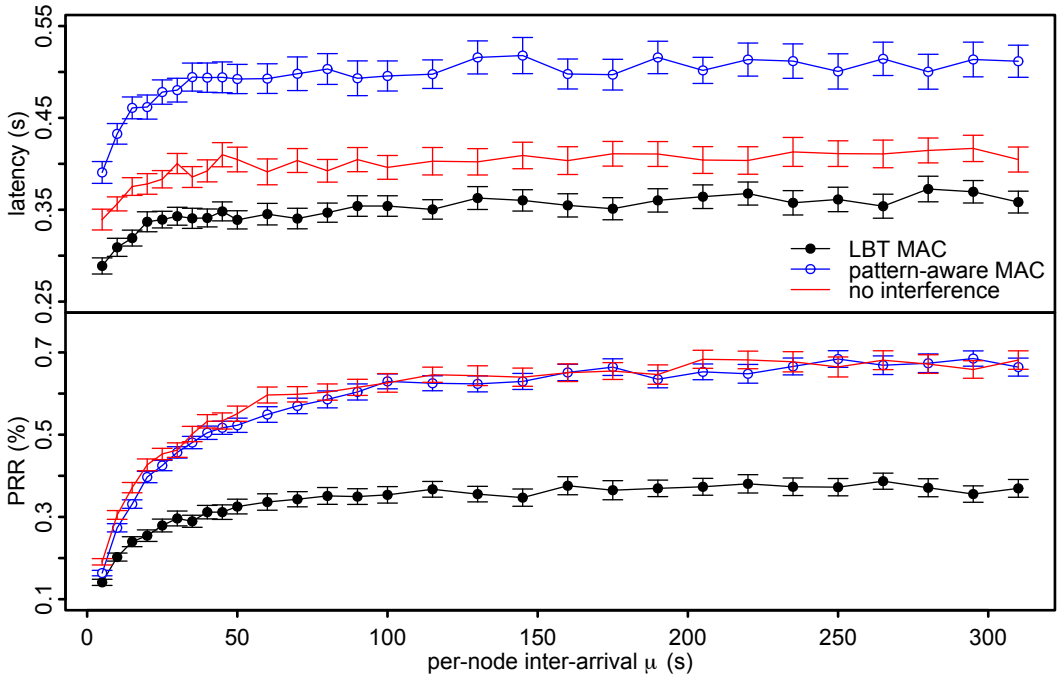


Figure 6.9: In a connected multi-hop network, the effect of varying the per-node packet inter-arrival rate on the latency (top) and packet reception rate (bottom).

As with the single-hop case, we notice decreasing PRRs and increasing latencies as the packet length increases, and PA-MAC again follows the PRR of the quiet configuration. However, unlike in the single-hop case, we notice that the quiet configuration no longer provides the baseline for delay. To explore this phenomenon, we investigate the path lengths compared to the packet lengths (Figure 6.8).

Since the network is static, we expected the path lengths to remain constant regardless of the packet lengths. However, we notice that as the packet length increases, the average number of hops decreases for the LBT MAC. The significant number of packet losses cause this behaviour: packets are more likely to be lost on the long paths, and these lost packets will not factor into the latency calculations.

The final graph for our simulation results shows the effect of varying the per-node packet inter-arrival rate on the latency and PRRs (Figure 6.9). In these experiments, we set the packet length to its maximum (60 bytes) in order to accentuate the effect of varying the inter-arrival rate.

Here, the PRR follows a similar trend to the single-hop case just at significantly lower levels. Unlike with the single-hop case, the latency curve increases (rather than decreases) as we slow the rate of packet introductions. Previously in the single-hop case, the transmit queues filled with packets, which led to greater latency. In this case, however, less congestion results in an increased number of successful transmissions along the long paths, which subsequently increases the latency.

In summary, these simulation results demonstrate the possible benefits of

Table 6.2: The CC1100 configuration for PA-MAC experiments.

Setting	Value
base frequency	904.000 MHz
channel spacing	199.951 kHz
receive filter bandwidth	270.833 kHz
modulation	2-FSK
data rate	9.993 kbps
transmission power	-30 dBm
channel	63
carrier frequency	916.597 MHz

using interference in a constructive manner rather than simply ignoring its existence. After adding our impulsive-interference classifier and pattern-aware MAC protocol to nodes, we saw immediate benefits. Naturally, more elaborate schemes could be devised. For example, it could be possible to construct a self-organizing TDMA-like MAC protocol around the impulsive interference and use the interference as the basis for its synchronization.

6.4 Hardware results

Although our wireless simulator has historically shown itself to be reasonably accurate, nothing inspires confidence quite like an implementation on real hardware. We had previously claimed that both the simplified impulsive-interference classifier and PA-MAC were suitable for wireless nodes, but we did not substantiate those claims. In this section, we describe experiments that we performed after testing both the classifier and PA-MAC on EMSPCC11 wireless sensor nodes. Given that PA-MAC depends on the classifier, our experiments consider their combined performance.

The EMSPCC11 devices from Olsonet Communications use a TI MSP430F1611 microcontroller (4.5 MHz, 10 KB of RAM, and 48 KB of flash memory) and flexible TI CC1100 transceiver. We set the transceiver's data rate to 10 kbps to match our simulations and resemble the 9.6 kbps rate that we used in our DM2200-based Smart Condo network. Table 6.2 summarizes the CC1100 configuration used in these experiments.

PA-MAC with an impulsive classifier requires the presence of periodic impulsive interference; to run our experiments, we needed a predictable and long-term source of such interference. Unfortunately, the unknown source of the Smart Condo's impulsive interference makes its long-term behaviour unpredictable. Therefore, we used extra hardware to generate interference with similar characteristics to our earlier observed pattern, and we then conducted experiments in an environment with this interferer.

Using a DM2200 – the same type of node used in our earlier Smart Condo work – we generated interference at 916.5 MHz. Although the hardware and operating system impose a number of restrictions on transmission parameters,

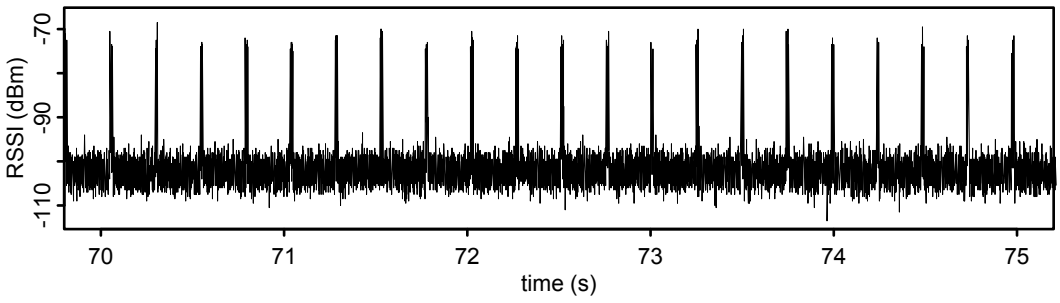


Figure 6.10: A trace of the received signal strength indicator showing the impulsive interference generated by a DM2200.

e.g., the frequency, power level, and minimum/maximum packet duration, the device is still flexible enough for our purposes. We created a simple application for the node to periodically transmit 8-byte packets at 9.6 kbps. Note that this size is the smallest possible while still using an official version of PicOS, and due to the DC-balanced encoding used by PicOS, each byte actually becomes 12 bits just prior to transmission. Each transmission includes preamble (28 bits) and a start symbol (16 bits), and considering these overheads in addition to the DC-balanced 8-byte payload, creates a transmission with a duration of around 15 ms. The node makes this transmission with an output power of 10 dBm, which creates significant interference at short distances.

To reproduce *impulsive* interference at 4 Hz, we separate each such transmission by 244 ms using a PicOS `delay` request.[4] When we first used our interferer, we noticed that the transmissions would sometimes be spaced irregularly. We discovered that transmissions, from an unknown source or even our own nodes, would cause an interferer's listen-before-talk (LBT) MAC to sometimes delay transmitting.

After disabling the LBT MAC functionality, we separated an interferer and EMSPCC11 by $\sqrt{0.75^2 + 0.75^2} = 1.06$ m to collect a trace to check the received power levels and verify its behaviour (Figure 6.10).[5] In this trace, the strong spikes around 25 dB above the noise floor originate from the interferer. At these distances, the interference transmitted at 10 dBm should easily overpower any of the transmissions at -30 dBm. Our plot starts at 70 s in order to avoid showing a coexisting (but weak) shifting-mean pattern that also occurred in the trace.

In setting up our experiment, we connected each WSN node to a computer using the same serial interface that we used in Chapter 4. By opting for a wired connection, we could use a command interface to flexibly guide experiments. Each node accepts commands, which may have one or two arguments, and issues a positive or negative response (Table 6.3).

With PA-MAC and the command interface programmed into the wireless sensor nodes, we then developed some back-end support to script our exper-

[4]The PicOS application issues a delay request for 250/1024 s.

[5]In the experiments that we describe later (Figure 6.11), we placed the receiving node 1.06 m from the interferer and each transmitting node.

iments. One such program, `genActual.pl`, generates one input file (*node* script) per node. For both the receiver and transmitters, the script issues commands from Table 6.3 to configure the (a) packet size (bytes), (b) channel number, (c) status of PA-MAC (enabled or disabled), and (d) status of TARP forwarding (enabled or disabled). For a receiver, the script then instructs the node to enter the receive state. For a transmitter, it generates packet transmission commands for a predetermined duration (seconds) and separates each command by a random exponential delay. The application also accepts a seed value to initialize the random number generator.

A *shell* script ultimately controls an experiment, where an experiment consists of transmitting packets under different node configurations for a number of repetitions. A short Perl script, `genActualSet.pl`, generates this *shell* script which, for each configuration, (a) calls `genActual.pl` to produce a *node* script for each involved node, (b) time-stamps the log file, (c) sends each generated *node* script to the appropriate node, (d) waits for the nodes to finish executing the supplied script, and (e) appends the resulting individual log files to a single large log file. When running our tests, the highest-level program (`genActualSet.pl`) always generates one configuration (e.g., a single size, arrival rate, or some other variable) first without and then with PA-MAC enabled.

Figure 6.11 shows the topology that we used for these experiments. During the experiments, we did not control the environment, so interference could have appeared and disappeared from external sources without our knowledge. In all cases, we placed the nodes directly on the floor, and no obstructions were within the square.

In order to investigate the protocol with different levels on congestion, we

Table 6.3: The commands implemented in the PicOS application for running PA-MAC experiments.

Cmd	Args	Description
l	x	set the packet length to x, but restrictions on the packet length will possibly increase this value
c	$0 \leq x \leq 255$	set the channel to x
x	$0 \leq x \leq 7$	set the power level to x, which ranges from 0 (-35 dBm) to 7 (about 10 dBm)
m	0/1	enable (1) or disable (0) PA-MAC
T	0/1	enable (1) or disable (0) forwarding using the Tiny Ad hoc Routing Protocol (TARP)
n	$x\ y$	collect $x \times y$ received signal strength indicator (RSSI) samples
r	N/A	start receiving packets (continues until the (a) transmit or (b) reset command)
q	N/A	stop all processes, disable the radio, erase any queued packets, re-enable the radio
t	x	transmit a packet to node ID x

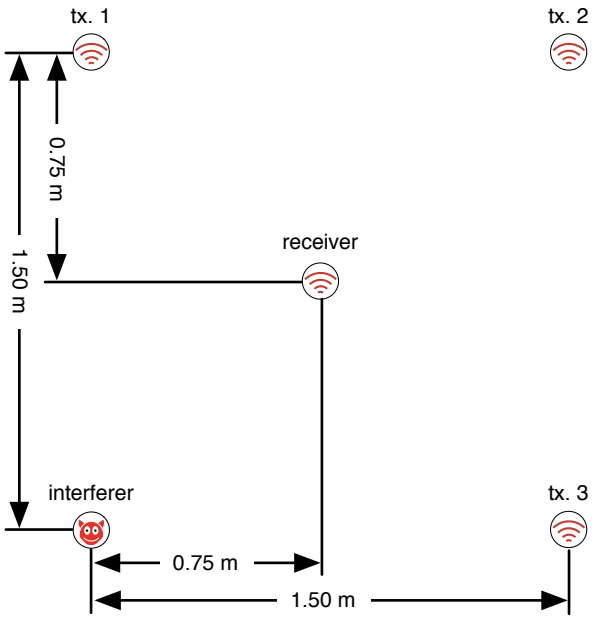


Figure 6.11: The low-density single-hop network that we used when determining the effect of PA-MAC on packet reception rate and latency. The error bars indicate the 95% confidence interval.

ran a series of experiments where we varied the per-node inter-arrival time μ while holding the packet length constant at 60 bytes (Figure 6.12). Under high congestion (smaller values of μ), we observed much higher latencies because nodes would locally queue packets if they were unable to transmit them fast enough. At the fastest arrival rates, nodes would receive more packets than they could queue and would drop some of the packets. In configurations with and without PA-MAC, the packet reception rates (PRR) show significant decreases under highly congested conditions (smaller μ) because of this dropping behaviour. In all cases, the performance of PA-MAC exceeded LBT MAC. If we disregard the dropped packets, the average PRR for the LBT MAC was 0.724, and for PA-MAC, 0.932; on average, PA-MAC improved the PRR by 28.7%.

Now with μ, the per-node inter-arrival rate, fixed at 0.3 s, we ran a series of experiments where we varied the packet length (bytes) to show the actual effect on latency and PRRs (Figure 6.13). The packets have payloads that range from 8 to 46 bytes, and we use a minimum length of 8 so that packets can accommodate some test-related data. The transceiver actually transmits 14 extra bytes with each packet: the headers station ID (2 bytes) and TARP (8 bytes) and the trailers status (2 bytes) and CRC (2 bytes). Therefore, the actually transmitted packet lengths range from 24 to 60 bytes.

In this experiment, the PRR of LBT MAC and PA-MAC changes at different rates as the packet length increases, with the PRR of the LBT MAC degrading faster. At all lengths, PA-MAC's PRR exceeds that of the LBT MAC: across all of the tested lengths, the average LBT MAC PRR was 0.791, while the average PA-MAC PRR was 0.958. On average, PA-MAC improved the PRR by 21.2%.

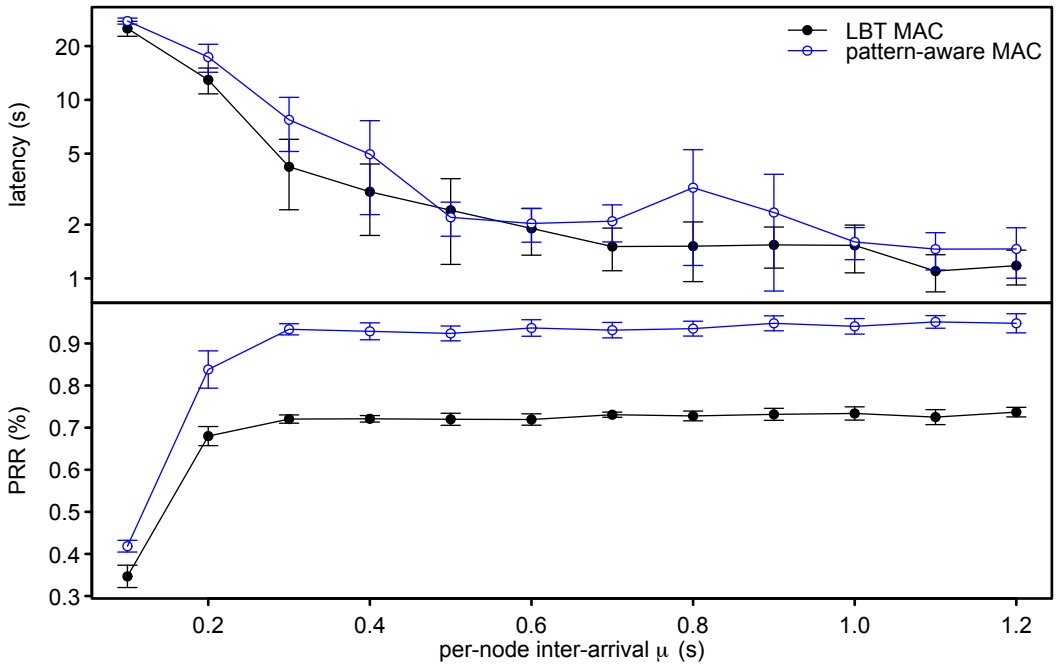


Figure 6.12: In a low-density single-hop network, the effect of varying the per-packet inter-arrival rate on the packet reception rate and the latency.

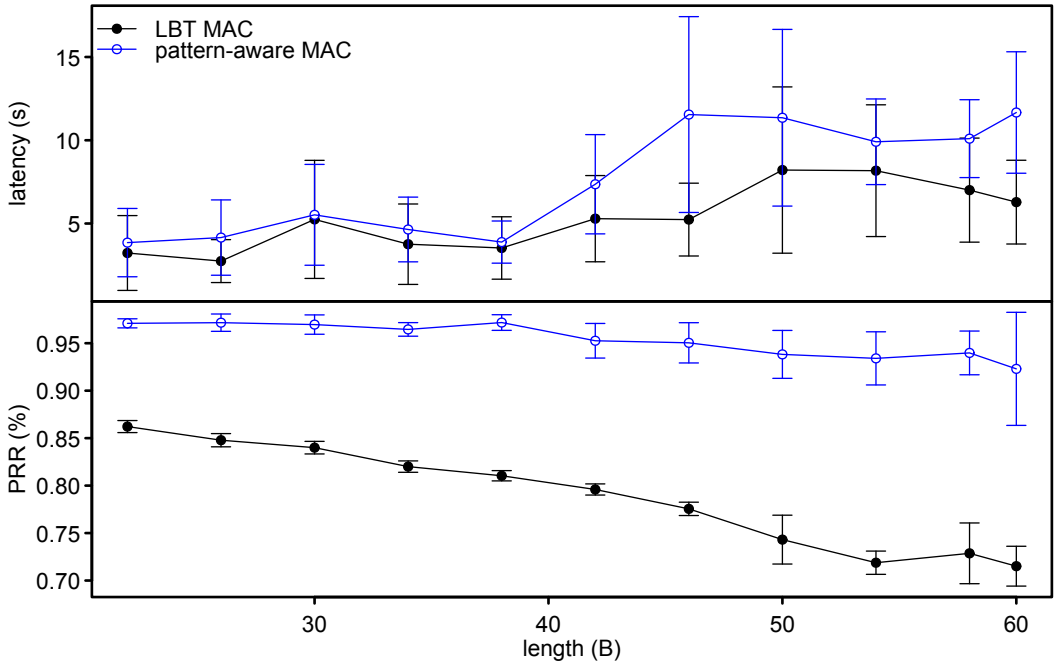


Figure 6.13: In a low-density single-hop network, the effect of varying the packet length on the packet reception rate and the latency.

Note that at all packet sizes, the network remained reasonably congested as evidenced by the rather high latencies.

Given all of the LBT MAC versus PA-MAC configurations that we tried, PA-MAC consistently showed an ability to improve PRRs. In the worst case, we saw a 12.6% improvement when transmitting our smallest tested packet size (24 bytes). In the best case, we saw a 31.2% improvement when transmitting large 60-byte packets at a low inter-arrival rate.

6.5 Summary

In this chapter, we extended the SIDE simulator with a flexible interface for the production of impulsive interference. By extending its existing configuration files, we outlined new syntax that allows users to describe a wide range of impulsive interference patterns.

We incorporated the classifier and a proof-of-concept pattern-aware MAC (PA-MAC) into SIDE's emulated radio driver. After simulating a variety of different configurations, we found that PA-MAC could improve the packet reception rates in both single- and multi-hop environments at the cost of increased latency.

Finally, we tested both the classifier and PA-MAC using actual WSN hardware (EMSPCC11). To perform the experiments, we needed a predictable long-term source of periodic impulsive interference, so we developed an application for the DM2200 – a radio incompatible with the EMSPCC11 – to generate it. After building a flexible command interface into the EMSPCC11 and some back-end software necessary for running the experiments, we evaluated the performance of PA-MAC under a variety of different conditions. In all cases, PA-MAC improved the packet reception rates significantly: we saw PRR improvements ranging from 12.6% to 31.2%.

Although our work here focused on the rapid-spikes pattern, we are optimistic that our approach to classify and subsequent avoid interference can apply to other patterns. The Lomb periodogram proved to be a particularly effective technique in identifying the rapid-spikes pattern, and given another pattern, another attribute may similarly capture its essence. If a classifier can recognize a sufficiently predictable pattern, like our rapid-spikes pattern, then it follows that a MAC protocol can avoid its interference.

Chapter 7

Conclusions

In this chapter, we summarize our work and main contributions (Section 7.1), identify some opportunities for future research (Section 7.2), and finally make some concluding remarks (Section 7.3).

7.1 Summary

In this thesis, we explored the development of wireless sensor networks for urban environments. We began our journey by distancing ourselves from many of the mainstream assumptions for wireless sensor networks, and instead, we sought those appropriate for this class of environment. Our appropriate assumptions include

- the presence of high levels of dynamic non-Gaussian noise and interference,
- the deployment of networks in stages so that they can evolve with user needs,
- the existence of extrinsic, application-specific constraints on the placement of nodes,
- the presence of abundant power sources that are available to at least a subset of nodes,
- the use of predominantly single-hop communication between a sensor node and its associated collection point (sink), and
- the requirements for both robustness and reliability, which stem from potential healthcare applications.

From these assumptions, we identified several relevant research topics: accurate emulation, site surveys, and protocol development.

Urban WSNs and software development

With a strong desire to develop and deploy our own urban wireless sensor network (WSN), we began working on the Smart Condo [11–13, 85], a prototype independent-living environment. Located within Telus Centre, a building on the University of Alberta campus that provides both classroom and office space, we deployed a wireless sensor network within an 80 m^2 room. The deployed

urban wireless sensor network used the RF Monolithics DM2200 wireless module [76], which incorporates a Texas Instruments MSP430F148 microcontroller and RF Monolithics TR8100 transceiver.

Given this hardware, we customized it for the Smart Condo application. We physically attached a number of peripherals to the wireless modules (nodes): passive-infrared motion sensors, switches, tactile pressure sensors, and accelerometers. We developed software for the nodes using the PicOS operating system (OS), a small foot-print OS that supports the organization of an embedded reactive application's multiple activities on a microcontroller with limited resources, and we integrated our reliable (application-layer) communication protocol. With this hardware and software in place, we deployed our network.

Although the network met its design goals, we noticed higher end-to-end latencies that we anticipated. Upon investigation, we discovered high packet loss rates that were not necessarily correlated with distance – a clear violation of a particularly common mainstream assumption. In searching for the underlying cause of the losses, we encountered high levels of interference in traces of the received signal strength indicator (RSSI). For comparison purposes, we explored two other environments: a 321-unit apartment building and the Computing Science Centre at the University of Alberta. In the apartment building, we also encountered significant interference, and in contrast, the Computing Science Centre was relatively quiet during our experiments.

Data collection and pattern overview

Given that we observed particularly strong interference within Telus Centre, we returned to this environment to better explore it. The transceiver used in the Smart Condo network, the TR8100, is a rather simple device that receives on only a single channel and produces its RSSI output as an analog signal that cannot be easily or accurately converted to a standard unit for received energy (dBm). To better understand the environment, we deployed a more sophisticated – but still WSN-class – node, the EMSPCC11, that incorporates the Texas Instruments CC1100 transceiver. This device provides greater flexibility for exploring the environment since it

- communicates on a user-selected frequency between 904 and 954 MHz with a channel spacing of 199.9512 kHz and receive filter bandwidth of 101.5625 kHz (configurable),
- produces a digital RSSI output that is easily converted to dBm with $1/2$ dB resolution, and
- generates that RSSI output rapidly at $12\,695.31$ Hz.

With this new hardware in place, we developed software and a framework for high-frequency sampling of the environment.

To accommodate the data collection, we connected the nodes to a single computer using

- USB cables that adapt a node's TTL-level serial output to USB (FTDI TTL-232R-3V3),

- USB extension cables (Assman Electronics), and
- powered 7-port USB hubs (Digitus DA-70227).

We wrote a PicOS application for the nodes to sample the transceiver's RSSI value as quickly as possible. For the single computer, we wrote an application in C that opens all of the USB ports simultaneously and reads the results as they are produced by the nodes. Within this program, we took a number of steps to reduce latencies, since the FTDI adapter provides only a 256-byte buffer – enough to buffer measurements for approximately 51 ms.

After collecting traces on the CC1100's 256 channels at roughly 5000 Hz – a higher rate for WSN-class devices than we have found in the literature – we plotted these time series data. Within our plots, we observed a number of reoccurring patterns.

1. The *quiet* class lacks any indication of interference.
2. The *quiet-with-spikes* class has infrequent (less than 1 Hz) and apparently random spiky noise/interference impulses.
3. The *quiet-with-rapid-spikes* class has frequent and periodic spiky impulses that are likely to impact packet loss rates.
4. The *high-and-level* class shows constant interference with a very narrow distribution.[1]
5. The *shifting-mean* class has apparently random shifts of the mean that produce a multi-modal distribution of the samples.

To better understand the distribution of patterns, we manually identified the predominant pattern in each trace. Those channels falling within the unlicensed industrial, scientific, and medical (ISM) band commonly exhibited infrequent and frequent impulsive interference. Although we also saw some shifting-mean patterns within the ISM band, they were more frequent on licensed channels, where we can attribute the pattern to telecommunications equipment, i.e., pagers. Within the non-ISM band, we noticed a much higher occurrence of quiet channels.

Given our preliminary classification of channels, we briefly explored techniques for automatically classifying a channel given a trace. We computed a variety of summary statistics for each trace, and using Weka 3.6.2, we applied a Bayesian network classifier to them. This classifier performed best on the quiet and high-and-level traces (as expected), but we were also satisfied with its performance for the other classes.

Subsampling and classification

Motivated by these promising results for classifying traces, we then revisited classification. This time, we abandoned the Bayesian network classifier to focus on a technique more suitable for the limited resources of a wireless sensor node: decision trees. We focused on those patterns most detrimental to packet loss rates: the quiet-with-rapid-spikes and shifting-mean traces. For each of these patterns, we identified statistics to aid in their classification, the Lomb periodogram and the dip statistic, respectively.

[1] We later determined that this interference originates at the node itself.

To improve the suitability of the technique for wireless nodes, we investigated the classifier given smaller sample sizes by subsampling our existing traces. To this end, we explored two techniques for subsampling: even and uneven/Poisson subsampling. The former technique is the easiest to implement, but the latter technique could prevent regularly missing the interference pattern during sampling.

Using the ϕ coefficient, we evaluated the performance of our classifier using various sample sizes and selectively restricting the attributes available to it. Both of the new features, the Lomb periodogram and the dip statistic, significantly improved the performance of their respective classifier for all sample sizes. We also noticed a significant reduction in the size of the generated decision trees when using these statistics. These smaller trees, which make fewer comparisons, would consume less space when stored within a node.

These reductions in the decision tree sizes motivated us to explore a classifier based on the new statistics alone – essentially a decision tree with a single decision node. In the shifting-mean case, with a decision node based on the dip statistic alone, the classification performance deteriorated significantly. We tried adding some further easily-computed statistics to the classifier, but again, the performance was inadequate without a large enough sample size. In the rapid-spikes case, with a decision tree based on the Lomb periodogram alone, the classifier performed extremely well given a sample size of at least 2000.

The promising results for the Lomb periodogram motivated us to simplify the algorithm for wireless sensor nodes. To this end, we

1. calculated and evolved an integer estimate of the mean,
2. disregarded its τ parameter, and
3. quantized the sine and cosine waves with discontinuities selected to allow for bit-shifting.

We tried both a 3- and 4-level approximation of the trigonometric functions, and in both cases, our simplification performed well. Given the lower memory and computation requirements of the 3-level case, we adopted that version for our subsequent experiments.

Evaluation using the simulator and hardware

Having developed a technique for identifying frequent short-duration impulsive interference, we then focused on putting such characterizations to constructive use to improve packet loss performance. To this end, we devised a proof-of-concept pattern-aware medium access control (PA-MAC) protocol to schedule transmissions between impulses. To evaluate this technique, we set out to test the MAC while varying certain variables and holding all other variables constant. Recall that we discovered early in our research that urban environments are highly complex and interference is dynamic: this realization led us to simulation.

To evaluate our MAC within a simulator, we had to first extend the simulator to generate impulsive interference. Given the close relationship between PicOS (our OS of choice) and the SIDE simulator, we quickly adopted SIDE as our simulator and began extending it to produce impulsive interference.

When simulating a network using SIDE, an XML file describes the nodes within the network. When the simulator starts, it parses this file and creates an object for each node; this object descends from the `Station` class. For our extension, it was natural to follow the existing semantics: we added new tags and attributes to the XML file, and the parsing of these attributes creates objects that also descend from the `Station` class. Within the XML file, we define clear and flexible syntax for describing the interference generated at an impulsive interferer.

With the simulator now generating interference, we integrated the classifier and PA-MAC protocol into the virtual radio driver. We explored packet reception rates and latency in both single- and multi-hop networks under varying loads (packet arrival rates) and varying packet lengths. After running these simulations, we confirmed that PA-MAC could improve the packet reception rates in both single- and multi-hop environments at the cost of increased latency.

Since nothing inspires confidence quite like a hardware implementation, we then tested both the classifier and PA-MAC within the EMSPCC11 wireless sensor node. In order to run our experiments, we needed a predictable source of impulsive inference, so we created one using a DM2200 node. We then set up a small single-hop network and confirmed that the classifier could recognize impulsive interference and PA-MAC could improve packet reception rates even when deployed on real hardware.

7.2 Future work

Our research has exposed some important and previously overlooked topics for future work with wireless sensor networks. Here we briefly describe the most promising directions.

1. Multi-hop deployments: When we evaluated both the classifier and PA-MAC using the EMSPCC11, we only deployed a single-hop network. In future work, we could evaluate the same software/hardware in multi-hop topologies.
2. Other environments: In this work, we obtained detailed samples from a single environment: the Smart Condo. From this environment, we identified a number of reoccurring patterns in our time-series data. It could be valuable to explore additional urban environments and determine the prevalence of the patterns that we encountered as well as identify some as-of-yet unknown patterns.
3. Hardware correlation: Our work did not explore the effect of individual node characteristics while sampling. The RF tract on a wireless sensor node is very sensitive, and even small differences such as the amount of solder applied, can effect its performance. Although we are certain of the correlation in the interference we observed at our nodes, we have not considered how the levels may differ between nodes.
4. Extensions to PA-MAC: Our proof-of-concept pattern-aware medium access control technique provides a number of directions for further work.

Although we have shown the benefit of it, a more complete protocol could overcome a number of its weaknesses:

(a) nodes individually classify a channel when they could consult their neighbours,
(b) nodes should reevaluate the channel when they observe that the environment has changed,
(c) when nodes join the WSN, communication must pause so that they can evaluate the channel, and
(d) the protocol does not leverage the possibility of heterogeneous nodes, where some nodes could have unlimited power.

Further development of the protocol in these directions could allow us to relax our correlation assumption and make the technique more useful in practical deployments.

7.3 Concluding remarks

Our work highlights the importance of considering real-world observations in research. By actually working with hardware and carefully observing its behaviour, we were able to identify a research problem overlooked by many others. Our work describes some important considerations for developing wireless sensor networks for dynamic urban environments.

Bibliography

[1] D. Aguayo, J. Bicket, S. Biswas, G. Judd, and R. Morris. Link-level measurements from an 802.11b mesh network. *SIGCOMM Comput. Commun. Rev.*, 34(4):121–132, 2004.

[2] E. Akhmetshina, P. Gburzynski, and F. Vizeacoumar. PicOS: A tiny operating system for extremely small embedded platforms. In H. R. Arabnia and L. T. Yang, editors, *Embedded Systems and Applications*, pages 116–122. CSREA Press, 2003.

[3] I. Akyildiz, W.-Y. Lee, M. Vuran, and S. Mohanty. A survey on spectrum management in cognitive radio networks. *Communications Magazine, IEEE*, 46(4):40–48, Apr. 2008.

[4] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: A survey. *Computer Networks*, 38(4):393–422, Mar. 2002.

[5] J. Al-Karaki and A. Kamal. Routing techniques in wireless sensor networks: A survey. *IEEE Wireless Communications*, 11(6):6–28, Dec. 2004.

[6] A. Amanna and J. Reed. Survey of cognitive radio architectures. In *Proceedings of the IEEE SoutheastCon*, pages 292–297, Mar. 2010.

[7] T. Arampatzis, J. Lygeros, and S. Manesis. A survey of applications of wireless sensors and wireless sensor networks. In *Proceedings of the 13th Mediterranean Conference on Control and Automation*, pages 719–724, Limassol, Cyprus, June 27-29, 2005.

[8] A. Arora, P. Dutta, S. Bapat, V. Kulathumani, H. Zhang, V. Naik, V. Mittal, H. Cao, M. Demirbas, M. Gouda, Y. Choi, T. Herman, S. Kulkarni, U. Arumugam, M. Nesterenko, A. Vora, and M. Miyashita. A line in the sand: A wireless sensor network for target detection, classification, and tracking. *Computer Networks*, 46(5):605–634, 2004. Military Communications Systems and Technologies.

[9] N. Baccour, A. Koubaa, M. Ben Jamaa, H. Youssef, M. Zuniga, and M. Alves. A comparative simulation study of link quality estimators in wireless sensor networks. In *MASCOTS '09: IEEE International Symposium on Modeling, Analysis Simulation of Computer and Telecommunication Systems*, pages 1–10, Sept. 2009.

[10] P. Baldi, S. Brunak, Y. Chauvin, C. Andersen, and H. Nielsen. Assessing the accuracy of prediction algorithms for classification: An overview. *Bioinformatics*, 16(5):412–424, 2000.

[11] N. M. Boers, D. Chodos, P. Gburzynski, L. Guirguis, J. Huang, R. Lederer, L. Liu, I. Nikolaidis, C. Sadowski, and E. Stroulia. *E-Health, Assistive Technologies and Applications for Assisted Living: Challenges and Solutions*, chapter The Smart Condo Project: Services for Independent Living. IGI Global, 2010.

[12] N. M. Boers, D. Chodos, J. Huang, P. Gburzynski, I. Nikolaidis, and E. Stroulia. The Smart Condo: Computing in service of assisted living. Poster at the second University of Alberta Festival of Teaching. Edmonton, Jan. 27, 2009.

[13] N. M. Boers, D. Chodos, J. Huang, E. Stroulia, P. Gburzynski, and I. Nikolaidis. The Smart Condo: Visualizing independent living environments in a virtual world. In *PervasiveHealth '09: Proceedings from the 3rd International Conference on Pervasive Computing Technologies for Healthcare*, London, UK, Apr. 2009.

[14] N. M. Boers, P. Gburzynski, I. Nikolaidis, and W. Olesinski. Developing wireless sensor network applications in a virtual environment. *Telecommunication Systems*, 45(2-3):165–176, 2010.

[15] N. M. Boers, I. Nikolaidis, and P. Gburzynski. Patterns in the RSSI traces from an indoor urban environment. In *CAMAD '10: IEEE 14th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks*, Coconut Creek, FL, Dec. 3-4, 2010.

[16] P. Bose, P. Morin, I. Stojmenović, and J. Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. *Wirel. Netw.*, 7(6):609–616, 2001.

[17] R. R. Bouckaert. *Bayesian Network Classifiers in Weka*. University of Waikato, Hamilton, New Zealand, July 2, 2007.

[18] I. Budiarjo, M. K. Lakshmanan, and H. Nikookar. Cognitive radio dynamic access techniques. *Wirel. Pers. Commun.*, 45(3):293–324, 2008.

[19] H. Celebi and H. Arslan. Enabling location and environment awareness in cognitive radios. *Comput. Commun.*, 31(6):1114–1125, 2008.

[20] Center for Universal Design. *The Principles of Universal Design, Version 2.0*. North Carolina State University, Raleigh, NC, 1997.

[21] A. Chandra. Measurements of radio impulsive noise from various sources in an indoor environment at 900 MHz and 1800 MHz. In *13th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, volume 2, pages 639–643, Sept. 2002.

[22] J. Cheng and R. Greiner. Learning Bayesian belief network classifiers: Algorithms and system. *Advances in Artificial Intelligence*, pages 141–151, 2001.

[23] Chipcon. Note on CC1100 device discrepancies. Technical documentation, application note, Sept. 6, 2005.

[24] D. Clark. The design philosophy of the DARPA Internet protocols. *Computer Communication Review*, 18(4):106–114, 1988.

[25] H. Cramér. *Mathematical Methods of Statistics*. Asia Publishing House, Bombay, 1962.

[26] Crossbow Technology. MICA2: Wireless measurement system. Data sheet, 2003.

[27] W. Deisman, P. Derby, A. Doyle, S. Leman-Langlois, R. Lippert, D. Lyon, J. Pridmore, E. Smith, K. Walby, and J. Whitson. A report on camera surveillance in canada: Part one. Technical report, Surveillance Camera Awareness Network (SCAN), Jan. 2009.

[28] I. Demirkol, C. Ersoy, and F. Alagoz. MAC protocols for wireless sensor networks: A survey. *IEEE Communications Magazine*, 44(4):115–121, Apr. 2006.

[29] J. Do, D. Akos, and P. Enge. L and S bands spectrum survey in the San Francisco Bay area. In *PLANS 2004: Position Location and Navigation Symposium*, pages 566–572, 2004.

[30] D. Douglas and T. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122, 1973.

[31] K. Fall, W. Hong, and S. Madden. Custody transfer for reliable delivery in delay tolerant networks. Technical Report IRB-TR-03-030, Intel Research, July 25, 2003.

[32] U. M. Fayyad and K. B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. *International Joint Conference on Artificial Intelligence*, 13(2):1022–1027, 1993.

[33] N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classifiers. *Machine Learning*, 29(2):131–163, 1997.

[34] M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, P. Alken, M. Booth, and F. Rossi. *GNU Scientific Library Reference Manual*. Network Theory Ltd., 3 edition, 2009.

[35] P. Gburzynski. *Protocol Design for Local and Metropolitan Area Networks*. Prentice Hall PTR, Upper Saddle River, NJ, 1995.

[36] P. Gburzynski. *SIDE/SMURPH: A Modeling Environment for Reative Telecommunication Systems (Version 3.2)*. Olsonet Communications, June 9, 2010.

[37] P. Gburzynski, B. Kaminska, and W. Olesinski. A tiny and efficient wireless ad-hoc protocol for low-cost sensor networks. In *DATE '07: Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1557–1562, San Jose, CA, 2007. EDA Consortium.

[38] P. Gburzynski and I. Nikolaidis. Wireless network simulation extensions in SMURPH/SIDE. In *WSC '06: Proceedings of the 2006 Winter Simulation Conference*, Monterey, California, Dec. 2006.

[39] A. Goldsmith. *Wireless Communications*. Cambridge University Press, 2005.

[40] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: An update. *SIGKDD Explorations*, 11(1), 2009.

[41] J. A. Hartigan and P. M. Hartigan. The dip test of unimodality. *The Annals of Statistics*, 13(1):70–84, 1985.

[42] P. M. Hartigan. Algorithm as 217: Computation of the dip statistic to test for unimodality. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 34(3):320–325, 1985.

[43] H. Hashemi. The indoor radio propagation channel. *Proceedings of the IEEE*, 81(7):943–968, 1993.

[44] C. Hsin and M. Liu. Network coverage using low duty-cycled sensors: Random & coordinated sleep algorithms. In *IPSN '04: Proceedings of the Third International Symposium on Information Processing in Sensor Networks*, pages 433–442, New York, NY, USA, 2004. ACM Press.

[45] S. B. Kotsiantis. Supervised machine learning: A review of classification techniques. *Informatica: An International Journal of Computing and Informatics*, 31(3):249–268, Oct. 2007.

[46] D. Kotz, C. Newport, R. Gray, J. Liu, Y. Yuan, and C. Elliott. Experimental evaluation of wireless simulation assumptions. In *Proc. of the 7th ACM Intl. Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pages 78–82. ACM Press New York, NY, USA, 2004.

[47] D. Kotz, C. Newport, R. S. Gray, J. Liu, Y. Yuan, and C. Elliott. Experimental evaluation of wireless simulation assumptions. Technical Report 2004-507, Dartmouth Computer Science, June 2004.

[48] S. Kotz, N. L. Johnson, and C. B. Read, editors. *Encyclopedia of Statistical Sciences*. Wiley-Interscience, 2 edition, 2006.

[49] H. Lee, A. Cerpa, and P. Levis. Improving wireless simulation through noise modeling. In *IPSN '07: Proceedings of the 6th International Conference on Information Processing in Sensor Networks*, pages 21–30, New York, NY, USA, 2007. ACM.

[50] N. R. Lomb. Least-squares frequency analysis of unequally spaced data. *Astrophysics and Space Science*, 39:447–462, Feb. 1976.

[51] S. Lühr, G. West, and S. Venkatesh. Recognition of emergent human behaviour in a smart home: A data mining approach. *Pervasive and Mobile Computing*, 3(2):95–116, 2007.

[52] G. Lukachan and M. Labrador. SELAR: Scalable energy-efficient location aided routing protocol for wireless sensor networks. In *29th Annual IEEE International Conference on Local Computer Networks*, pages 694–695, Nov. 2004.

[53] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *WSNA '02: Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications*, pages 88–97, New York, NY, USA, 2002. ACM Press.

[54] G. Mao, B. Fidan, and B. D. Anderson. Wireless sensor network localization techniques. *Computer Networks*, 51(10):2529–2553, 2007.

[55] B. W. Matthews. Comparison of the predicted and observed secondary structure of T4 phage lysozyme. *Biochimica et Biophysica Acta (BBA) - Protein Structure*, 405(2):442–451, 1975.

[56] Metrix Communication LLC. Metrix Mark II Kit [MET-KIT-MARK-II] - $550.00: Metrix Communication LLC. http://www.metrix.net/metrix-mark-ii-kit-p-14.html, 2009.

[57] R. Musaloiu-E. and A. Terzis. Minimising the effect of WiFi interference in 802.15.4 wireless sensor networks. *International Journal of Sensor Networks*, 3(1):43–54, 2008.

[58] H. J. Newton. Lecture 2: The periodogram. Lecture notes, STAT 685, Texas A&M University, 2007.

[59] P. Oehen and B. Plattner. DSNAnalyzer: Backend for the deployment support network. Master's thesis, Swiss Federal Institute of Technology (ETH) Zurich, 2006.

[60] J. Oetting. A comparison of modulation techniques for digital radio. *IEEE Transactions on Communications*, 27(12):1752–1762, Dec. 1979.

[61] W. Olesinski, A. Rahman, and P. Gburzynski. TARP: A tiny ad-hoc routing protocol for wireless networks. In *ATNAC '03: Proceedings of Australian Telecommunications Networks and Applications Conference*, Melbourne, Australia, Dec. 8–10, 2003.

[62] Olsonet Communications Corporation. Platform for R&D in sensor networking. `http://www.olsonet.com/Documents/emspcc11.pdf`, 2008.

[63] Olsonet Communications Corporation. *VUE*^{2} *(Version 0.9)*. Ottawa, ON, Canada, Sept. 2010.

[64] R. Orpwood, T. Adlam, N. Evans, J. Chadd, and D. Self. Evaluation of an assisted-living smart home for someone with dementia. *Journal of Assistive Technologies*, 2(2):13–21, 2008.

[65] J. Pan and W. J. Tompkins. A real-time QRS detection algorithm. *IEEE Transactions on Biomedical Engineering*, BME-32(3):230–236, Mar. 1985.

[66] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[67] J. Polastre, J. Hill, and D. Culler. Versatile low power media access for wireless sensor networks. In *SenSys '04: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, pages 95–107, New York, NY, USA, 2004. ACM.

[68] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, Cambridge, MA, 2 edition, 1992.

[69] J. G. Proakis. *Digital Communications*. McGraw-Hill, New York, 4 edition, 2001.

[70] J. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[71] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2010. ISBN 3-900051-07-0.

[72] A. Räisänen and A. Lehto. *Radio Engineering for Wireless Communication and Sensor Applications*. Artech House Publishers, 2003.

[73] T. S. Rappaport. *Wireless Communications: Principles and Practice*. Prentice Hall PTR, 1st edition, 1996.

[74] T. Rein and D. Thiele. Energy and time efficient link-quality estimation for wireless sensor networks. Master's thesis, Swiss Federal Institute of Technology (ETH) Zurich, 2007.

[75] RF Monolithics, Inc. TR8100: 916.50 MHz hybrid transceiver. Data sheet, 2006.

[76] RF Monolithics, Inc. DM2200-916VM: 916.50 MHz Transceiver Module. http://www.rfm.com/products/pdf/dm2200-916vm.pdf, Apr. 2008.

[77] M. S. Roden. *Analog and Digital Communication Systems*. Prentice-Hall, Englewood Cliffs, NJ, 2 edition, 1985.

[78] T. Rusak and P. Levis. Physically-based models of low-power wireless links using signal power simulation. *Computer Networks*, 54(4):658–673, 2010.

[79] D. Schmidt, M. Berning, and N. Wehn. Error correction in single-hop wireless sensor networks: A case study. In *DATE '09: Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1296–1301, 3001 Leuven, Belgium, Belgium, 2009. European Design and Automation Association.

[80] D. Son, B. Krishnamachari, and J. Heidemann. Experimental study of concurrent transmission in wireless sensor networks. In *SenSys '06: Proc. of the 4th Intl. Conference on Embedded Networked Sensor Systems*, pages 237–250, New York, NY, USA, 2006. ACM.

[81] K. Srinivasan, P. Dutta, A. Tavakoli, and P. Levis. Understanding the causes of packet delivery success and failure in dense wireless sensor networks. In *SenSys '06: Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, pages 419–420, New York, NY, 2006. ACM.

[82] K. Srinivasan, P. Dutta, A. Tavakoli, and P. Levis. An empirical study of low-power wireless. *ACM Transactions on Sensor Networks*, 6(2):16:1–16:49, 2010.

[83] K. Srinivasan, M. Jain, J. I. Choi, T. Azim, E. S. Kim, P. Levis, and B. Krishnamachari. The κ-factor: Inferring protocol performance using inter-link reception correlation. In *MobiCom '10: Proceedings of the Sixteenth Annual International Conference on Mobile Computing and Networking*, pages 317–328. ACM, 2010.

[84] K. Srinivasan and P. Levis. RSSI is under appreciated. In *EmNets '06: Proceedings of the Third ACM Workshop on Embedded Networked Sensors*, 2006.

[85] E. Stroulia, D. Chodos, N. M. Boers, J. Huang, P. Gburzynski, and I. Nikolaidis. Software engineering for health education and care delivery systems: The Smart Condo project. In *SEHC '09: Proceedings from the 31st International Conference on Software Engineering*, Vancouver, Canada, 2009.

[86] A. S. Tanenbaum. *Computer Networks*. Prentice Hall PTR, Upper Saddle River, NJ, 4th edition, 2003.

[87] A. Terzis, R. Musaloiu-E., J. Cogan, K. Szlavecz, A. Szalay, J. Gray, S. Ozer, C.-J. M. Liang, J. Gupchup, and R. Burns. Wireless sensor networks for soil science. *International Journal of Sensor Networks*, 7:53–70, February 2010.

[88] Texas Instruments. Efficient multiplication and division using MSP430. Application Report SLAA329, Sept. 2006.

[89] Texas Instruments. *Data sheet for CC2420: 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver*. Dallas, Texas, Mar. 20, 2007.

[90] Texas Instruments. RSSI interpretation and timing. Design Note DN505, SWRA114B, Oct. 22, 2007.

[91] Texas Instruments. Data sheet for CC1100: Low-power sub-1 GHz RF transceiver, Oct. 2009.

[92] D. J. Vergados, N. A. Pantazis, and D. D. Vergados. Energy-efficient route selection strategies for wireless sensor networks. *Mob. Netw. Appl.*, 13:285–296, August 2008.

[93] M. Vieira, J. Coelho, C.N., J. da Silva, D.C., and J. da Mata. Survey on wireless sensor network devices. In *ETFA '03: Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation*, volume 1, pages 537–544, Sept. 2003.

[94] B. Wang and K. J. R. Liu. Advances in cognitive radio networks: A survey. *IEEE Journal of Selected Topics on Signal Processing*, 5(1):5–23, Feb. 2011.

[95] K. Whitehouse, C. Sharp, E. Brewer, and D. Culler. Hood: A neighborhood abstraction for sensor networks. In *Proceedings of the 2nd International Conference on Mobile Systems, Applications, and Services*, pages 99–110. ACM New York, NY, USA, 2004.

[96] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[97] A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *SenSys '03: Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, pages 14–27, New York, NY, USA, 2003. ACM.

[98] N. Xu, S. Rangwala, K. K. Chintalapudi, D. Ganesan, A. Broad, R. Govindan, and D. Estrin. A wireless sensor network for structural monitoring. In *SenSys '04: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, pages 13–24, New York, NY, USA, 2004. ACM.

[99] T. Yucek and H. Arslan. A survey of spectrum sensing algorithms for cognitive radio applications. *IEEE Communications Surveys Tutorials*, 11(1):116–130, First quarter 2009.

[100] M. Z. Zamalloa and B. Krishnamachari. An analysis of unreliability and asymmetry in low-power wireless links. *ACM Transactions on Sensor Networks*, 3(2), 2007.

[101] J. Zhao and R. Govindan. Understanding packet delivery performance in dense wireless sensor networks. In *SenSys '03: Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, pages 1–13, New York, NY, USA, 2003. ACM Press.

[102] G. Zhou, T. He, S. Krishnamurthy, and J. A. Stankovic. Impact of radio irregularity on wireless sensor networks. In *MobiSys '04: Proceedings of the 2nd International Conference on Mobile Systems, Applications, and Services*, pages 125–138, New York, NY, USA, 2004. ACM.

[103] G. Zhou, T. He, S. Krishnamurthy, and J. A. Stankovic. Models and solutions for radio irregularity in wireless sensor networks. *ACM Transactions on Sensor Networks*, 2(2):221–262, 2006.

[104] M. Zuniga and B. Krishnamachari. Analyzing the transitional region in low power wireless links. In *SECON '04: First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks*, pages 517–526, Oct. 2004.

Appendix A

Sample VUE^2 Configuration

```
<network nodes="17">
  <grid>0.1m</grid>
  <tolerance quality="2">1E-4</tolerance>

  <channel>
    <shadowing bn="-110.0dBm" syncbits="8">
      RP(d) = received power at distance d
      XP  = transmitted power
      X   = lognormal random Gaussian component
      ======================================================
      RP(d)/XP [dB] = -10 x 5.1 x log(d/1.0m) + X(1.0) - 33.5
      ======================================================
    </shadowing>

    <cutoff>-115.0dBm</cutoff>

    <ber>
      Interpolated ber table:
      ======================
      SIR       BER
      50.0dB      1.0E-6
      40.0dB      2.0E-6
      30.0dB      5.0E-6
      20.0dB      1.0E-5
      10.0dB      1.0E-4
       5.0dB      1.0E-3
       2.0dB      1.0E-1
       0.0dB      2.0E-1
      -2.0dB      5.0E-1
      -5.0dB      9.9E-1
    </ber>

    <frame>12 16</frame>
    <rates>9600</rates>
    <rssi>0 -202.0 255 53.0</rssi>

    <power>
      0    -30.0dBm
      1    -15.0dBm
```

```
    2    -10.0dBm
    3     -5.0dBm
    4      0.0dBm
    5      5.0dBm
    6      7.0dBm
    7     10.0dBm
  </power>
</channel>

<nodes>
  <defaults>
    <memory>2048 bytes</memory>

    <radio>
      <power>7</power>
      <preamble>32 bits</preamble>

      <lbt>
        delay           8msec
        threshold         -109.0dBm
      </lbt>

      <backoff>
        min           8msec
        max           303msec
      </backoff>
    </radio>

    <leds number="3">
      <output target="socket"></output>
    </leds>

    <!-- note that by not specifying the debouncing parameters,
         we've effectively disabled debouncing -->
    <pins total="12" adc="1" counter="3" notifier="4" dac="0">
      <input source="socket"></input>
      <output target="socket"></output>
      <values>000000000000</values>
    </pins>

    <uart rate="38400" bsize="12">
      <input source="socket"></input>
      <output target="socket"></output>
    </uart>

    <preinit tag="DATECODE" type="lword">2008120100</preinit>
    <preinit tag="HID" type="lword">3133857453</preinit>
    <preinit tag="MHOST" type="word">1</preinit>
    <preinit tag="NID" type="word">85</preinit>
    <preinit tag="SENS_0" type="lword">0</preinit>
    <preinit tag="SENS_1" type="lword">0</preinit>
  </defaults>

  <node number="0" type="bridge" start="on">
```

```
  <location>8.4 6.8</location>
  <preinit tag="HID" type="lword">0xBACA0001</preinit>
</node>

<node number="1" type="sensor" start="on">
  <location>1.1 3.5</location>
  <preinit tag="HID" type="lword">0xBACA0002</preinit>
  <preinit tag="SENS_0" type="lword">4</preinit>
</node>

<node number="2" type="sensor" start="off">
  <location>10.5 2.7</location>
  <preinit tag="HID" type="lword">0xBACA0003</preinit>
  <preinit tag="SENS_0" type="lword">4</preinit>
</node>

<node number="3" type="sensor" start="off">
  <location>6.3 2.1</location>
  <preinit tag="HID" type="lword">0xBACA0004</preinit>
  <preinit tag="SENS_0" type="lword">4</preinit>
</node>

<node number="4" type="sensor" start="off">
  <location>8.7 2.3</location>
  <preinit tag="HID" type="lword">0xBACA0005</preinit>
  <preinit tag="SENS_0" type="lword">4</preinit>
</node>

<node number="5" type="sensor" start="off">
  <location>10.6 3.0</location>
  <preinit tag="HID" type="lword">0xBACA0006</preinit>
  <preinit tag="SENS_0" type="lword">4</preinit>
</node>

<node number="6" type="sensor" start="off">
  <location>3.4 5.1</location>
  <preinit tag="HID" type="lword">0xBACA0007</preinit>
  <preinit tag="SENS_0" type="lword">3</preinit>
</node>

<node number="7" type="sensor" start="off">
  <location>6.5 1.9</location>
  <preinit tag="HID" type="lword">0xBACA0008</preinit>
  <preinit tag="SENS_0" type="lword">3</preinit>
</node>

<node number="8" type="sensor" start="off">
  <location>8.0 0.1</location>
  <preinit tag="HID" type="lword">0xBACA0009</preinit>
  <preinit tag="SENS_0" type="lword">5</preinit>
</node>

<node number="9" type="sensor" start="off">
  <location>8.5 3.4</location>
```

```
      <preinit tag="HID" type="lword">0xBACA000A</preinit>
      <preinit tag="SENS_0" type="lword">6</preinit>
    </node>

    <node number="10" type="sensor" start="off">
      <location>6.8 0.3</location>
      <preinit tag="HID" type="lword">0xBACA000B</preinit>
      <preinit tag="SENS_0" type="lword">3</preinit>
    </node>

    <node number="11" type="sensor" start="off">
      <location>5.9 3.6</location>
      <preinit tag="HID" type="lword">0xBACA000C</preinit>
      <preinit tag="SENS_0" type="lword">3</preinit>
    </node>

    <node number="12" type="sensor" start="off">
      <location>5.3 4.6</location>
      <preinit tag="HID" type="lword">0xBACA000D</preinit>
      <preinit tag="SENS_0" type="lword">3</preinit>
    </node>

    <node number="13" type="sensor" start="off">
      <location>6.8 3.4</location>
      <preinit tag="HID" type="lword">0xBACA000E</preinit>
      <preinit tag="SENS_0" type="lword">3</preinit>
    </node>

    <node number="14" type="sensor" start="off">
      <location>6.6 3.7</location>
      <preinit tag="HID" type="lword">0xBACA000F</preinit>
      <preinit tag="SENS_0" type="lword">3</preinit>
    </node>

    <node number="15" type="sensor" start="off">
      <location>8.2 6.3</location>
      <preinit tag="HID" type="lword">0xBACA0010</preinit>
      <preinit tag="SENS_0" type="lword">3</preinit>
    </node>

    <node number="16" type="sensor" start="off">
      <location>6.5 0.5</location>
      <preinit tag="HID" type="lword">0xBACA0011</preinit>
      <preinit tag="SENS_0" type="lword">3</preinit>
    </node>
  </nodes>
</network>
```