N° d'ordre:

THÈSE

Présentée devant

L'UNIVERSITÉ DE BORDEAUX
École Doctorale de Mathématiques et Informatique

pour obtenir le grade de :

DOCTEUR DE L'UNIVERSITÉ DE BORDEAUX
Mention INFORMATIQUE

par

Martin PERES

Laboratoire d'accueil : LABRI
Équipe de recherche : PROGRESS

Titre de la thèse :

A Holistic Approach to Green Networking in Wireless Networks : *collaboration among autonomic systems as a mean towards efficient resource-sharing*

Directrice de thèse: **Francine KRIEF**
Co-encadrant de thèse: **Mohamed Aymen CHALOUF**

Soutenue le 19 décembre 2014 devant la commission d'examen:

M. :	Raymond	NAMYST	Pr. à l'Université de Bordeaux, France	Président
MM. :	Ahmed	KARMOUCH	Pr. à l'Université d'Ottawa, Canada	Rapporteurs
	Jean-Marc	PIERSON	Pr. à l'Université Paul Sabatier, France	
MM. :	Guillaume	FERRÉ	MCF à l'Université de Bordeaux, France	Examinateurs
	Raymond	NAMYST	Pr. à l'Université de Bordeaux, France	
	Christian	TOINARD	Pr. à l'INSA Centre Val de Loire, France	
MM. :	Francine	KRIEF	Pr. à l'Université de Bordeaux, France	Encadrants
	Mohamed Aymen	CHALOUF	MCF à l'Université de Rennes 1, France	

Abstract

The last twenty years saw the emergence of wireless systems in everyday's life. They made possible technologies such as mobile phones, WiFi or mobile Internet which are now taken for granted in today's society. The environmental impact of Information and Communications Technology (ICT) has been raising exponentially to equate the impact of the airline industry. The green computing initiative has been created in response to this observation in order to meet the 15%-30% reduction in green-house gases by 2020 compared to estimations made in 2002 to keep the global temperature increase below 2°C. In this thesis, we studied power-saving techniques in wireless networks and how they interact with each others to provide a holistic view of green networking. We also take into account the radio frequency resource which is the most commonly used communication medium for wireless systems and is becoming a scarce resource due to our society's ever-increasing need for mobile bandwidth. This thesis goes down the network stacks before going up the hardware and software stack. Contributions have been made at most layers in order to propose an autonomic wireless network where nodes can work collaboratively to improve the network's performance, globally reduce the radio frequency spectrum usage while also increasing their battery life.

Keywords Green networking; autonomic computing; power management; distributed systems; run-time decision engines; reputation management.

Contents

List of Figures

List of Tables

This page has been intentionally NOT left blank.

Acknowledgements

Firstly, I would like to thank my thesis directors Francine Krief and Mohamed Aymen Chalouf for introducing me to interesting challenges, providing me with their continuous feedback, giving me the means to carry out my research in good conditions and giving me great ideas on how to improve my work. I would also like to thank Professor Ahmed Karmouch for having me at the University of Ottawa and supervising my research there.

Not all the technical work found in this thesis is the result of my sole work. I would thus like to thank all the people I worked with during the ANR projects Diaforus and Licorne but would like to thank in particular Romain Perier for working for almost a year with me on the Diaforus project. Not only was he of great help while designing many of the algorithms found in Diaforus, but he also implemented most of them along with implementing Diase, the visualisation GUI we created. I would also like to thank Guillaume Ferré and his students for our work on the physical layer using software-defined radios which was really helpful for the Licorne project.

Likewise, I would like to thank all the non-academic colleagues I worked with and especially the Nouveau developers for teaching me their understanding of NVIDIA GPUs and for the numerous discussions we had together either online or in real life. Among many others, I would especially like to thank Ben Skeggs (the Nouveau maintainer), Marcin Kościelnicki (Envytools maintainer), Christoph Bumiller, Ilia Mirkin and Roy Spliet. I would also like to thank my Google Summer of Code (2013 & 2014) student Samuel Pitoiset who has been working with me on reverse engineering performance counters on some NVIDIA cards and who started implementing them in Nouveau. I would also like to thank Marcin Slusarz for our discussions relative to Nouveau and his proof-reading of my hardware-related chapter.

I would also like to thank my other colleagues and fellow PhD students of the University of Bordeaux and ENSEIRB-MATMECA. However, I would like to thank in particular Vincent Autefage, Tom Bouvier and Mathieu Barjon not only for our long discussions, brainstorming on distributed networking and our long debates on computer science and mathematics, but also for becoming dear friends. I would also like to thank in particular all the PhD students of the CVT room and its neighbouring room before thanking all the other PhD students, research associate and professors I met during my time in Bordeaux. Outside my work environment, I would like to thank all the members of the *LaBx*, the hackerspace of Bordeaux, for sharing their

entire knowledge openly. From this group, I would particularly like to thank Jonathan BISSON, Thibault Melsbach, Richard Moglia and Zener.

I would also like to thank my family and non-computer-scientist friends who supported me during my greater-than-usual isolation and lack of communication without holding grudges. This includes all my Erasmus/exchange students friends I met where I studied or during my trips. I am really grateful to having all of them in my life.

This thesis has been written with a lot of background music which I am sure contributed greatly to delivering this thesis in time. This includes many progressive and symphonic rock bands such as Dream Theater, Epica, Genesis, Nightwish, Pink Floyd, Riverside, Rush, Transatlantic, Yes and all the musical projects of Arjen Anthony Lucassen. I would also like to thank the highly-energetic bands such as Arch Enemy, Gojira and Pantera for keeping me awake better than any cup of coffee could ever do.

Glossary

ADC An Analog-to-Digital Converter (ADC) converts a voltage into a digital representation. This representation is bound by the number of bits of the ADC, for instance, a 8-bit ADC can only output a representation of the voltage between 0 and 255. The correspondence between the representation and the actual voltage depends is linear between the ADC's V_{min} (usually 0 V) and V_{max} (usually called V_{ref}). It's reverse operation is done by a Digital-to-Analog Converter (DAC) device. 45, 46, 48, 56, 84

CPU Central Processing Unit, the main processor of a computer. The CPU is meant to execute user-provided applications and the one controlling the other processors present on the computer. xxvi–xxviii, 1, 5, 25, 49, 56–58, 85, 92, 93, 95, 96, 100, 106, 110, 115, 117–121, 123, 124, 133–138, 149–152, 157, 160, 163–165

CR A Cognitive Radio (CR) is radio communicating with other Cognitive Radios to form a Cognitive Radio Network (CRN). xxv, xxvi, 45, 47, 48, 51–55, 58–65, 68, 71

CRN A Cognitive Radio Network (CRN) is a network composed of intelligent radios that are capable of understanding their Radio Frequency environment. They opportunistically find the frequency bands that are not in used by the rightful owners (primary users) and share them among so-called secondary users. An additional goal of cognitive radios is to keep the network of secondary users connected at all time and guarantee the QoS requirements of the applications running on top of it. 48, 58, 60, 72

daemon A daemon is a program running in the background of an Operating System and which does not interact directly with the user. 138

DVFS Dynamic Voltage/Frequency Scaling is a power-saving technique for processors that consists in dynamically selecting the most efficient performance level of the processor, based on its load. xxvii, 85, 89, 94, 100–105, 107, 110, 114, 116, 160, 165

FFT A Fast Fourier Transform (FFT) allows converting a signal from the time domain to the frequency domain. 48, 49, 57, 197

GPGPU General-Purpose computing on Graphics Processing Units. Using a GPU to perform computations that were usually handled by the CPU. OpenCL is the only standard for writing GPGPU applications that work on Intel, AMD and NVIDIA GPUs. However, NVIDIA's API called CUDA was introduced first and performs better on NVIDIA GPUs which explains why so many applications prefer using this API. 86, 88, 96, 97, 100, 110, 117, 119

GPU Graphics Processing Unit. GPUs were introduced as 2D and 3D graphics accelerators for CPUs and are programmable using OpenGL or Direct3D. Recently, they became more general purpose (GPGPU) with the introduction of CUDA or OpenCL. xxvi, xxviii, 1, 4, 6, 76–80, 85–90, 93–107, 110, 117–120, 126–128, 138, 150–152, 157, 160, 163–165, 169, 170

HW Hardware. 122, 124, 125, 128–132, 134, 135, 138

I2C Inter-Integrated Circuit. I2C is a serial communication bus invented by Philips in 1982 in order to add low-speed peripherals to traditional computers. 36, 78, 86, 103

interrupt An interrupt is an electrical signal sent by a piece of hardware to a processor to signal an event that requires the processor's immediate attention. Interrupts introduced event-driven processing instead of polling inputs constantly, thus freeing resources for more useful purposes and/or lowering the power consumption by putting the processor to sleep. 78, 88, 90, 95

PLL A Phase-Locked Loop takes a clock as an input and generates an output clock being a fractional multiple of the input clock. 46, 79, 89, 90

QoS Quality of Service aims at guaranteeing the overall performance of a network on any combination of the following typical metrics: throughput, transmission delay, availability, jitter, error rates. QoS is critical for real-time applications such as audio and/or video streaming but also data logging. 2–6, 16, 73, 110, 114, 120, 122, 125, 130, 141, 144, 148, 152, 156, 158–160, 164

RAM Random Access Memory (RAM) is a storage device that allows data to be written or read from anywhere in memory with no performance cost, contrarily to hard-disk drives which have a non-uniform access time. RAM is volatile, which means that the data is lost after a power-down or a reset, contrarily to Read-Only Memory (ROM). There are two kinds of RAM, static and dynamic. Dynamic RAM (DRAM) requires less transistors than static RAM (SRAM) but it requires to be periodically refreshed. xxvii, 11, 15, 25, 86, 89, 116, 121, 135, 138, 156, 159, 187

RF A Radio Frequency (RF) signal is an electromagnetic signal oscilating between 3 kHz and 300 GHz. xxviii, 2, 3, 6, 8, 19, 44, 46, 47, 52, 64, 67, 73, 78, 124, 130–132, 147, 148, 153

ROM Read-Only Memory. Device that stores data that cannot be modified or is generally not meant to be modified. ROMs are often used for storing firmwares. 11, 25, 156

RX Reception of a message. 12, 13, 15, 47, 48, 59

SNR Signal-to-Noise Ratio (SNR) is a measure that compares the strength of the wanted signal compared to the unwanted signals (background noise + other signals). It is usually expressed in decibels (dB). 9, 11, 48, 49, 52, 70, 71, 78, 196

SW Software. 45, 48

TDP The Thermal Design Power (TDP) is the maximum thermal power the processor's heatsink is required to be able to dissipate. xxvii, 91, 95, 96, 100, 103, 107, 150

TX transmission of a message. 12, 47, 48, 58, 70

WSN Wireless Sensor Network. Network composed of multiple inexpensive wireless nodes that are potentially distributed over a large area to sense their environment. A WSN is ad-hoc and often used in scientific or industrial applications. xxxi, 2, 11, 13, 22–26, 28, 29, 33, 37–39, 42, 163

Chapter 1

General introduction

Contents

1.1 General Context

The last twenty years saw the emergence of wireless systems in everyday's life. This trend is mostly due to the increasing need to acquire data in more-and-more remote areas and also to the convenience mobile devices provide to end users.

Most wireless systems have communication capabilities and are powered by a battery. As such, they have a maximum usage time before needing to swap or recharge the battery. This time depends on the capacity of the battery and the power consumption of the system.

The general public first got introduced to mobile devices in 1973 when presented with the first hand-held mobile phone. At the time, the battery-life was about 20 minutes and recharging it took 10 hours [18]. Since then, mobile computing started thriving with the introduction of laptops which had a very-limited battery-life at first but evolved to nowadays cover more than a typical work-day. The convergence between laptops and mobile phones gave birth to smartphones and tablets. These devices feature very-powerful CPUs and GPUs along with 4G and WiFi network interfaces, allowing the device to access the Internet almost-everywhere, with a home-like bandwidth.

The general public is not the only user of wireless devices. One of the most widely-known industrial and scientific application for them is weather monitoring and forecasting. Wireless and autonomous stations have been deployed all around the world to collect weather data and transmit it via the cellular network or a satellite communication. As this system is supposed to be streaming data periodically, they cannot be powered with a battery because they have a fixed capacity and swapping them would require a costly human intervention. Recharging the batteries can however be done automatically by harvesting the surrounding energy such as the ambient light, vibrations or radio signals by respectively using a solar panel, a piezzo-electric element or an antenna.

Although a weather monitoring system can be qualified as a Wireless Sensor Network (WSN), it is a very simple one because each system/node can reach the telecommunication infrastructure directly. When the spatial density of nodes increases, it becomes more cost-effective to make the nodes communicate with each others to route the sensed information to a special node with long-distance communication capabilities. This way, nodes can be fitted with inexpensive short-range communication radios which lower the power consumption and the size of each node. As the power consumption is lower, these devices can be powered with less bulky and less expensive batteries which drives the cost down even more. WSN deployments can be used for crops monitoring for precision agriculture where only the dry parcels get water. In the same fashion, temperature, humidity, O_2 and CO_2 levels can be monitored in Greenhouses and warn farmers when any reading is out of the expected ranges. Some WSN are composed of mobile sensors. When mounted on drones, they can be controlled remotely and can be relocated dynamically unlike the cheaper versions that are usually mounted on throw-away balloons. The main example is weather monitoring with weather balloons.

Mobile or static WSN commonly use the Radio Frequency (RF) spectrum to communicate. The RF environment is a shared communication medium that can be used by multiple devices to communicate with one another. Multiple communications can happen at the same time if the devices are aware of their RF environment or if an exploitation license has been given to guarantee the exclusive usage of part of the RF spectrum to the entity controlling this device. Failing in sharing the RF spectrum can have a strong impact on communications which may result in a Quality of Service (QoS) lower than needed by the devices and their users. Other possibilities for wireless communications include sound and light but they usually consume more power and are slower than using RF communications.

Various degrees of autonomy can be found in wireless devices when it comes to their internal management, location management for drones and the usage of the RF environment. Wide-scale distributed systems management are problematic to manage, especially with the ever-growing complexity of every device that composes it. An increased autonomy of the device in order to achieve self-configuration, self-optimisation, self-protection and self-healing drastically reduce the amount of configuration needed and can result in a longer-lived system thanks to local decision-making requiring less control messages and enabling better power management.

1.2 Problems studied in this thesis

In this thesis, we present an hollistic approach to wireless systems on the topic of power management and efficient sharing of the RF spectrum for communications. The study spans from the hardware to the applications and goes through every network layer of the TCP/IP model.

1.2.1 Decentralising data processing

Our propositions to lower power consumption of wireless networks are aimed towards decentralising decision-making in order to increase the autonomy of individual nodes. With all the information needed locally, the device can self-optimise to lower its power consumption within the boundaries set by the needed QoS.

This autonomy can also drastically reduce the number of messages, increase the reliability of the network through automatic re-organisation and can improve auditability in certain circumstances.

This thesis studies the challenges associated with decentralisation in wireless networks and their mitigation. These problems are mainly demonstrated through an heterogeneous and redundant wireless sensor network aimed towards physical-intrusion detection.

1.2.2 Efficient spectrum sharing

Spectrum-sharing is usually done by a regulation entity who attributes licences to companies or usages. These licences allow their recipient to use a band of the RF spectrum with a given transmission technology. Some bands do not require a licence but still require to comply with some rules.

While these regulations statically make sure that the needed QoS by every company is met at any point of the territory, it often results in so-called white spaces when a company does not use use its licence everywhere on the territory. Detecting those white-spaces and sharing them with other so-called secondary users can be used to increase the bandwidth of the dynamic spectrum-sharing users. A user performing this collaboration is called a cognitive radio.

This thesis proposes a set of protocols to achieve a large scale cognitive network and studies the added-constraints onto the hardware, operating system of all the devices of this cognitive network. These protocols work better when using software-defined radios for which we also propose a software infrastructure.

1.2.3 Hardware performance and power consumption

When a device knows all its applications' QoS constraints, it can self-optimise according to them. However, there are no easy hardware dials to modulate performance or power consumption. The only hardware parameters that can be tweaked to affect performance are the voltage and the clock frequency of one or several processors found in the device.

In a modern processor, the relation between clock frequency and performance cannot always be foreseen statically. This is why modern processor have special blocks, called performance counters to provide performance introspection that allows detecting when a processor is under-utilised or when it is currently the processing bottleneck. Dynamic Voltage/Frequency Scaling (DVFS) is the process that is commonly used to vary the level of performance based on this load indication.

Calculating the power consumption based on performance is also not always possible. Trying to limit the power consumption of a processor, no matter its impact on performance, requires hardware and software co-design in order to achieve good efficiency in all scenarios. This is made more difficult for researchers because most hardware companies consider power management as an industrial secret.

In this thesis, the origin of power consumption is studied starting from the transistors up to the whole device. We also propose how to modulate performance and/or power consumption in order to meet the higher-level requirements set by the applications. The result are demonstrated on an NVIDIA GPU which is considered to provide the most power-efficient, power-aware and performance-aware hardware. This work highly relies on clean-room reverse engineering and is used in the Linux open source driver for NVIDIA cards, called Nouveau.

1.2.4 Making run-time power management decisions

Radios are usually most energy- and spectrum-efficient at their maximum bitrate. However, the propagation channel or the recipient(s) may not allow the usage of this setting. The most suitable settings should thus be selected on both the emitter and the receiver in order to allow the communication to happen in the most efficient way.

Wireless devices like smartphones are often composed of multiple radios. These radios are not entirely flexible, allow only one communication at a time on a selected number of frequency bands and have a restricted choice for the physical layer parameters. This means that depending on the user's activity, some radios may be better-suited and more power-efficient than others. The cost of switching from one radio to another can be prohibitive and needs to take into account the mobility of the user, and his/her current activity to avoid doing un-necessary switches that may have a performance or a power cost for no benefit. The decision is almost entirely mission-dependent and cannot be statically written once for all because it requires power and performance models of the embedded radios and application's QoS.

A decision engine, coupled to MPTCP, could deliver seamless hand-hovers between radios while providing the lowest possible power consumption and the performance needed to fulfil the QoS.

Likewise, some systems are composed of multiple heterogeneous cores which have different power consumption and performance characteristics. Selecting the most appropriate

ones at run time can save a substantial amount of power while also increasing the user's satisfaction by providing the necessary performance.

This thesis proposes a decision engine that is compatible with real-time requirements in order to be usable at run time. It also is generic-enough to allow smartphones developers to model the performance and power models of their radios and CPU cores to provide a decision that increases the battery life while also increasing the QoS.

1.2.5 Creating a green network for disaster relief services

Disaster scenarios usually involve the loss of a lot of infrastructures which can make it difficult for rescue teams to coordinate efficiently by exchanging data to prioritise rescue missions based on their urgency.

In the future, such coordination could be obtained through using tablets that would allow rescuers to communicate by voice and video, annotate with the work that needs to be done on a map of the area updated in real time. Finally, it could also show the position of other rescuers to allow a rescuer to call for help.

The rescuer's communication system could not rely on existing infrastructures because they may have been damaged by the disaster. Instead, rescuers should be able to set-up a network very easily that would re-use the white spaces to provide as much bandwidth and the lowest latency possible across a wide area. To do so we propose a very flexible relay infrastructure that may be composed of both terrestrial and aerial relays that uses software-defined radios and our signalling protocols to provide a constant connectivity at the needed QoS. The selection of the white space that should be used for the communication can also be done using our run-time decision engine.

Guaranteeing the availability of the tablets and the connectivity of the network requires modulation power consumption and performance across the network. Using our hardware contribution, we can detect an unsustainable power consumption in the hardware, attribute it to an application or a communication and potentially shut it down. This would allow every node in the network to guarantee its availability for a given amount of time while gracefully lowering its performance when its power consumption is too high.

Wireless sensor networks could also be deployed very easily to monitor the air for dangerous gases and make sure people keep out of dangerous areas. Our decentralised data processing contributions would allow for an automatic response to an intrusion (sounding an alarm). It would also increase the life of the network and allow it to run for several months while providing excellent debugging capabilities.

Such a scenario demonstrates how the contributions of the thesis can work together to provide complex cognitive behaviour at every layer and on most nodes of the network.

1.3 Outline of this thesis

In Chapter 2, we introduce the concept of green computing and green networking. We then describe how radio-frequency-based communications operate and how multiple users can share this radio frequency spectrum before talking about the existing type of wireless deployment. We then introduce different ways of saving power by changing the way computers communicate and by changing the way their hardware operates. Finally, we introduce the concept of autonomic computing which aims at creating self-managing components in order to reduce both the operational complexity and the power consumption of the network as a whole.

In Chapter 3, we propose a way to dramatically increase the lifetime of a surveillance network by implementing reasoning capabilities in wireless sensor nodes and letting them collaboratively detect meaningful events before alerting the network administrator. This proposition heavily relies on a data-centric network communication paradigm and on the fact that events should be spatio-correlated. This proposition is however so successful in lowering the number of communications that it becomes impossible for the network operator to detect any problem in the network, nodes or their sensors. To improve the introspection capabilities of the system, we propose multiple power-efficient techniques.

In Chapter 4, we study the problems associated with sharing the radio frequency (RF) spectrum among multiple users and propose using software-defined radios to create a cognitive radio network. The cognitive behaviour of this network allows nodes to understand their RF environment and react to it to guarantee the needed QoS for the applications. We then propose and evaluate two signalling protocols that enables node discovery, time synchronisation and sleep scheduling while also allowing them to find the least crowded frequency bands and sending/receiving multiple transmissions concurrently.

In Chapter 5, we study existing hardware through reverse engineering the power management capabilities of the most power-efficient hardware, NVIDIA GPUs. We then study the impact of multiple factors such as voltage, frequency and temperature on power consumption and then show ways to change these parameters to adapt to the current application running in order to lower the power consumption without negatively affecting performance. The way NVIDIA manages these parameters is then studied under various circumstances. Finally, we show how NVIDIA GPUs are already autonomous-ready and have been improving in this direction for multiple years.

In Chapter 6, we introduce the need for run-time power management by demonstrating the power saving achievable by having a dynamic policy. We then propose a generic run-time decision engine for power management which we evaluate in two scenarios: network interface selection and performance level selection. We then compare the overhead of our proposition compared to the most common language used in the literature.

In Chapter 7, we showcase how our different contributions can work together to create a suitable network for a disaster relief scenario. We finally conclude the thesis in Chapter 8 and present our future work and perspectives.

Chapter 2

State of the art

Contents

Energy is the source of every change in our universe, from particle physics to cosmological changes. Without energy, no work is possible.

At a human scale, the availability of energy has been directly linked to economical growth [19]. This growth comes from an increase in added value which can be only be obtained by work. Cheap energy means cheap work which can thus lead to an increased added value for the same final price and also economic growth.

Although improvements in the efficiency of our machines and in our way of transporting energy has made it seem like economic growth and energy consumption were decoupling [20], energy usage keeps on rising.

As our society is increasingly reliant on Information and Communications Technology (ICT), it's energy usage increases by around 10% each year and was already consuming 10% of the electricity worldwide [21] in 2007, equating to 2% of the total energy usage. In 2008 in France, a report [22] estimated that 55 to 60 TWh per year were consumed by end applications which amounts to 13.5% of France's electricity consumption. These figures do not take into account the production, transport, and recycling costs [23].

Energy does not only have a monetary cost, it also has an environmental cost as generating this energy released 0.82 Gtons of CO_2 in the atmosphere in 2007 and is expected to reach 1.43 GTons in 2020. This represents 2% of the total emission around the world putting ICT's emission at the same level as the airline industry [24]. CO_2 is a Green-House Gas and as such, should see its emission reduced by 15%–30% before 2020 in the European Union to keep the global temperature increase below 2°C [25].

The "green IT" initiative is an effort to reduce the footprint of ICT in the environmental pollution. We can say that green IT is about efficient usage of resources at any stage of the life-cycle of the ICT hardware, including its production, transport, usage and its recycling. Multiple techniques can be applied to reduce the impact of ICT, some are presented in [23] [25]. Green IT may also be called green computing or green networking.

In this chapter, we present an overview of the different techniques proposed to lower the energy consumption in green wireless networks. This review is done at every layer because an optimisation at one layer can be detrimental to other layers and result in a negative result. We present this holistic approach in different sections for the ease of readability. We first introduce in Section 2.1 how multiple nodes can communicate wirelessly using the radio frequency spectrum without interfering with each others. We then introduce in Section 2.2 different power- and spectrum-efficient ways for multiple nodes to communicate together. In Section 2.3, we focus on the hardware used by the wireless nodes to improve its power efficiency. Finally, in Section 2.4, we introduce the vision of autonomic computing which aims at lowering the maintenance cost of a complex system while also improving its performance and power-efficiency.

2.1 Wireless communications

Wireless communication technologies are usually based on electromagnetic radiations with a frequency ranging from 3kHz to 300 GHz, also known as radio frequencies. They are different from wired communication because the communication medium has a much-higher attentuation and is shared among every wireless radios. Another difference is that radios need to cope with echos in the signal (multipath effect).

2.1.1 Transmitting information wirelessly

Digital information can be sent over the radio frequency (RF) spectrum using different modulation techniques. The signal to be transmitted could be encoded on the amplitude of a sine wave, by shifting the sine wave's frequency or by changing the phase of the sine wave. It is also possible to combine Amplitude-Shift Keying (ASK) with the Frequency-Shift Keying (FSK) or Phase-Shift Keying (PSK). For instance, the Quadrature Amplitude Modulation (QAM) modulation combines the ASK with the PSK. It is a very common modulation for high-speed digital communications. Many other modulation schemes and variants of them are presented in [26].

In the same way oral communication uses phonems that assemble into words and sentences, digital communications use symbols to carry the information. A symbol can

encode one or more bits. For instance, the 2-PSK has two possible states which allow encoding one bit of information while a 64-QAM can encode 8 bits per symbol. The symbol rate is linearly linked to how much spectrum is being used for the communication [27]. It is thus more spectrum-efficient to encode as many bits per symbol as possible. However, the number of bits per symbol is limited by the Signal-to-Noise Ratio (SNR) found at the receiver [28].

2.1.2 Sharing the spectrum

The radio frequency spectrum can be shared by multiple radio transceivers by transmitting at different frequencies (FDMA/OFDMA), at different time (TDMA), using different codes (CDMA) or at a different locations (SDMA). These techniques are all presented in [29] and can manage to share the spectrum among multiple nodes without harmful interference.

SDMA reduces the chances of two nodes interfering by greatly limiting the interference area. However, it still spreads all the energy over a wide area. The company Artemis proposed a way to concentrate all the energy right at the antenna of the receiver and track it continuously thanks to their pCell technology [30].

Other techniques require some sort of synchronisation. Indeed, TDMA requires an accurate time synchronisation across all the nodes of the network to create short time slots. Short time slots are necessary to get a small network latency. All the nodes should also be given a slot which usually requires centralisation. Likewise, making sure that no node outside the network is using the same channel or CDMA code requires some form of centralisation or signalling.

A purely-local solution exists under the name Carrier Sense Multiple Access with Collision Detection (CSMA/CD). It has been introduced in IEEE 802.3 and consists in sensing the communication medium to check that no communication is currently happening before transmitting. The problem with this technique is that a node may not detect a transmission received by another node further away and start transmitting. The receiver node would then be unable to properly receive any of the two inbound transmissions. Both interfering transmissions are said to have collided.

CSMA/CD has been designed for wired networks where every network interface can detect when a collision happened. In wireless networks, the attenuation of the communication medium is much higher and if a network is spread over a relatively large area, it is unlikely that every network node will be able to hear the transmissions of every other node of the network. To avoid this “hidden node problem”, Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) has been introduced in IEEE 802.11 to reduce the number of collisions in wireless networks. This technique allows reserving the network around receiving nodes by having them emit a Clear-To-Send (CTS) frame to reserve the channel before receiving a transmission. This CTS frame is sent in response to a Ready-To-Send (RTS) frame sent by the source node. An overview of the behaviour is available in Figure 2.1. Both CSMA/CD and CMSA/CA are defined in [29].

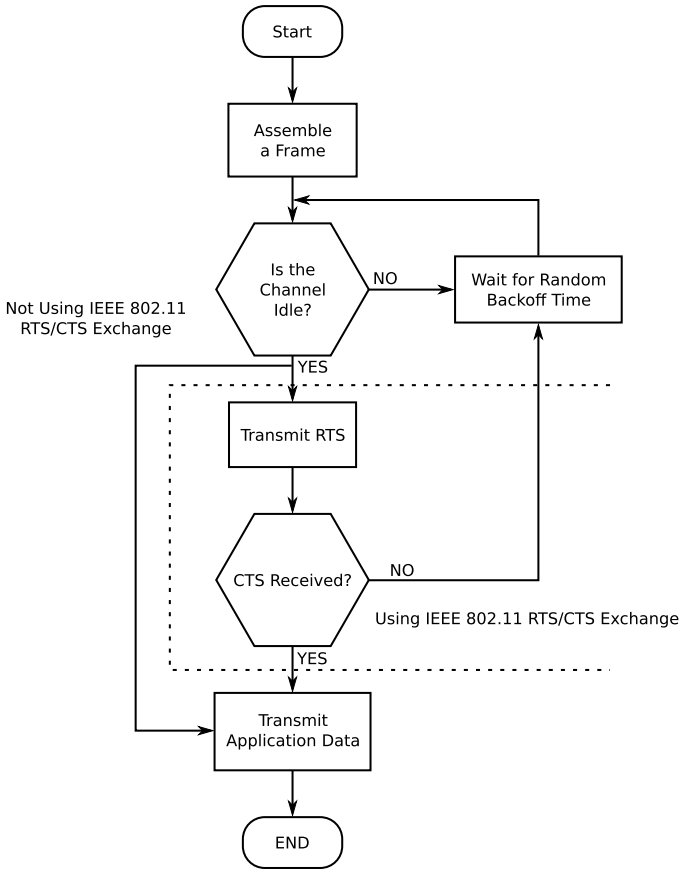


Figure 2.1: Simplified overview of the Carrier Sense Multiple Access with Collision Avoidance technique. Source [1].

The frequency at which a transceiver is allowed to radiate (transmit power) is currently state-regulated and licenses are given or sold for some technologies, regions and/or a company. By licensing frequency bands to different technologies and different companies, the state grants a monopoly that allows companies to guarantee the availability of their services operating on the licensed frequency band. So-called unlicensed bands where the state allows individuals to radiate without authorisations come with limitations such as the maximum emission power, the duty cycle (how often is the transceiver allowed to communicate) and how much spectrum is it allowed to use (linear with the bitrate). This allows individuals to compute a statistical probability of message loss if an estimate of the amount of wireless devices in the direct vicinity of the transceiver can be obtained.

Such a fixed spectrum allocation results in a sub-optimal usage of the spectrum as the licensed users, also called primary users, may not use the spectrum at all time and everywhere. To fix this problem, the research community proposed to develop cognitive radio (CR) networks that would allow a CR to re-use spectrum allocated to primary users as long as it does not interfere with any of the primary users of the band [31].

2.1.3 Applications for wireless communications

Wireless communications are very useful to lower the cost of deployment of a wired communication channel. It can be very efficient to provide Internet access to isolated homes in cities or to coastal islands. In both cases, relays communicating in a line-of-sight fashion enable the use of directional antennas which increase the SNR and provide a higher bandwidth while the lower interference on surrounding radios both increases the reusability of the spectrum while also diminishing the average latency of the network by lowering the likeliness of having collisions.

Wireless communications are also very useful for mobile users. Thanks to the GSM, 3G and 4G infrastructure, users can phone each others from almost anywhere in populated areas around the globe. Smartphones may also use this infrastructure to provide mobile access to the Internet even when their users are moving.

These architectures are however extremely inefficient. Indeed, to produce 40 W of radiated power, a 3G station consumes around 500 W. As the coverage of such station is usually limited to a few kilometres, the global power consumption of the complete infrastructure is non-trivial. Improving the efficiency of the 3G stations could thus yield substantial savings when multiplied by the number of antenna found throughout the world. Multiple techniques to do so have been proposed in [23].

Wireless Sensor Networks (WSNs) use wireless communications to route the data collected by their sensors to the outside of the sensor network. Wireless links allow them to be easily deployed which results in a lower cost compared to traditional wired networks. WSN are already used in several fields such as monitoring air-quality [32], seismic activity [33], forest fires [34], structural integrity of a building [35] and also area intrusion detection [36].

Wireless sensor nodes are small nodes characterised by their very low processing power and storage capacity (both ROM and RAM). They are also characterised by their very limited energy resources. Despite these constraints, sensors are usually expected to be secure and serve data with a relatively low latency and operate over multiple months or years without swapping their batteries.

Wireless meters are also an increasingly popular use of wireless sensor networks. They are meant to speed up the collection of the meters' reading by allowing a read-out of all the meters of an area to be collected by simply getting close to one of them [37]. This is thanks to the fact that these so-called smart meters create a mesh network that allows routing the readings of every meter of the network to the person in charge of retrieving them. A sensor fitted with a GSM connection can even be used to periodically collect the readings and forward it to a server of the company for billing. This continuous reading however poses privacy problems [38].

2.2 Green wireless networks

In the previous section, we introduced how wireless nodes can communicate with each others and which kind of applications make use of these capabilities. In this section, we study techniques commonly used in green wireless networks to lower the power consumption.

2.2.1 The overhearing problem

To be available, a radio has to be in receive (RX) mode. In wireless sensor networks, a radio being in the RX mode consumes roughly the same power as when it is in the emission (TX) mode [39]. However, very few messages are actually exchanged in such networks which yields a very poor average energy usage per exchanged message. To reduce the average power consumption of the nodes, radios should spend most of their time in sleep mode where they barely consumes any power.

Ideally, the radio should only transit from the sleep to RX state when a message is about to be sent. Any time spent with the radio in the RX mode while not receiving any information, called overhearing time, is a complete waste of energy. The problem is that a radio cannot receive a message or signal when it is in sleep mode so it has to have another way to know when a message is about to be sent.

Preambles

One solution to fix this overhearing problem for mostly-idle networks is to put the burden of synchronisation on the emitter by introducing very long preambles before the actual message. For instance, radio A is willing to send a message to radio B. Radio A sends a 100 ms-long preamble before sending its message. Radio B spends most of its time in sleep mode but wakes up every 90 ms for a few milliseconds to check if a preamble is currently happening. If there is none, then it can go back to sleep mode for another 90 ms. If there is a preamble, the radio waits until its end before receiving the message. This allows receiving radios to spend most of their time in sleep mode unless necessary.

This preamble solution has been successfully used by the wave2m [37] technology to drastically lower the power consumption of the wireless sensor nodes. In [40], the authors proposed an improved solution that broadcasts multiple times a very short message containing the destination radio. This allows other radios to go back to sleep earlier as they do not have to receive the message header to know if they are the intended recipient. The authors also proposed that the receiving node should acknowledge the preamble before the emitter sends the complete message. This allows to lower the spectrum usage if the receiver is unavailable. The authors managed to perform the check for the preamble on the receiver side in 0.38 ms. This figure is so low that it enables multi-channel capabilities which could be used to improve the throughput in the network while also reducing the chances of collisions.

The length of the preamble needs to be adjusted depending on the average frequency at which messages are sent in the network. For instance a preamble of 100 ms is fine if a

message is sent every minute in average. It is however much too long if a message is sent every second as radios would start spending a non-trivial amount of time in RX mode (up to 10%). Likewise, 100 ms would be too short if a message is sent in average every hour as radios would have to wake up constantly and would have a very low probability of receiving a message. The preamble length should thus be set according to the expected average frequency at which messages are sent in the network and the maximum latency acceptable by the application running on the network.

Time synchronisation

Another solution to the overhearing problem is for nodes to agree to only send information periodically, at the start of every second for instance. Radios would only need to be put in the RX mode at the beginning of every second and keep it long-enough to check that no radio in its surrounding started emitting a message.

The problem of such solution is that the clock of each radio has a slightly different rate because the crystal used is usually not very accurately cut. This means that even if the radios were synchronised at one point, their clocks would drift apart and the network wouldn't be connected anymore. Even if the crystals were exactly the same on all the radios, the temperature difference between the radios caused by different light and wind exposition would result in a frequency difference.

This clock drift can be identified and corrected in software by exchanging messages with other nodes. In [41], the authors propose a synchronisation mechanism that works for single hop networks. For large networks, other algorithms can be used to keep the clocks in sync [42][43].

If the sensor nodes hosting the radios are equipped with a GPS, it is also possible to use the time broadcasted by the GPS satellites as a time reference to fix the clock drift of the radios. The time accuracy of such clocks is 50 ns which is more than enough for our needs. GPS receivers are however power-hungry and should not be enabled at all time.

Once nodes are synchronised, it is possible to implement TDMA in the network. Provided a good assignation of time slots, the advantages of TDMA are that no collisions can happen and radios can sleep when it is not their turn to communicate. A distributed TDMA slot assignment algorithm for WSNs has been proposed in [44].

2.2.2 Efficient routing algorithms

There are different topologies of network, as can be seen in Figure 2.2. The Mesh topology is usually found in wireless ad hoc networks as mesh nodes are connected together without a central hierarchy. The other topologies are usually found in infrastructure networks where a hierarchy is needed.

A network is said to be ad hoc when it does not rely on any infrastructure to operate and has been created with little to no planning. Such network requires its nodes to

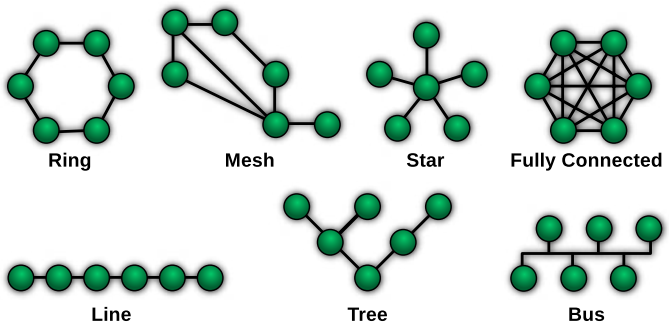


Figure 2.2: Example of network topologies. Source [2].

forward data for other nodes. The route taken by data depends on the source, the destination and the routing algorithm used.

Multiple routing algorithms have been proposed in the literature. Some of them find routes on-demand (reactive) while some are pro-active and maintain up-to-date routing tables. The main advantage of the first is that it is able to cope with a highly-dynamic network but this comes at the price of a high latency for the first communication. On the contrary, pro-active routing provide a low routing latency but require a lot of background traffic to keep the routing tables up to date in a mobile situation.

Pro-active routing algorithms such as OLSR (Optimized Link State Routing Protocol) [45] or B.A.T.M.A.N. (Better Approach To Mobile Ad-hoc Networking) [46] are best suited for low-latency applications exchanging a lot of data in order to make the necessary background traffic insignificant.

Reactive routing algorithms such as AODV (Ad hoc On-Demand Distance Vector Routing) [47] are best-suited for extremely mobile applications not minding a potential high-latency delivery time due to the on-demand nature of the algorithm. Such algorithms are much more efficient than pro-active ones when the application running on the network does not exchange a lot of data as no background traffic is necessary.

Some routing algorithms can fall in the following categories:

- Data-centric: Data is interpreted and/or fused by the routing process;
- Hierarchical: Data follows a strict hierarchy of nodes to reach its destination;
- Geographic: Data is routed based on GPS coordinates.

Data-centric routing protocols such as COUGAR [48] or ACQUIRE [49] allow users to specify the type of information they would like to receive. They both are centralised approaches where one node is responsible for aggregating and processing data. This node also asks the right sensors to stream their data to their "leader node". When an event of importance happened, the leader node sends the event to the gateway.

Another data-centric routing algorithm has been proposed under the name "Efficient Overlay for Publist/Subscribe in Wireless Sensor Networks" [50]. This algorithm is also centralised and allows to attach a semantic to data. It also lets any node in the network subscribe to some events with some conditions attached, contrarily to the two other data-centric protocols we previously introduced.

Hierarchical routing protocols such as LEACH (Low-Energy Adaptive Clustering Hierarchy) [51] or RPL (IPv6 Routing Protocol for Low-Power and Lossy Networks) [52] are meant to save power and minimise the length of the routing tables. This is often needed in Wireless Sensor Networks because of the very-limited RAM available.

The LEACH routing protocol uses a clustering approach. The election process for the clusters is done purely locally based on the remaining power and probabilities. When a node decides to become a clusterhead it sends a message telling so and waits for surrounding nodes to attach to it. Nodes attached to this clusterhead can then send their messages periodically before going back to sleep. Clusterheads however need to keep their radio in RX mode at all time to be able to receive the messages of the nodes of the cluster. To limit the number of messages sent to the gateway, the data of all the sensors can be aggregated in the clusterhead. All the clusterheads are re-elected periodically which indicates that a time synchronisation is necessary. This approach works well for densely populated networks with nodes positioned randomly but may result in a loss of connectivity if a clusterhead has no route available towards the sink.

The RPL routing protocol is reactive and organises the network in a tree. The depth of a node in the tree is based on a so-called "rank". The rank is an attribute of each node in the tree and is a number strictly increasing when going down the tree toward the leaves. When a node is willing to add itself to the network, it sends a DIS message to probe the surrounding nodes which then answer in a DIO message that primarily contain their rank. The node will then need to attach to a preferred parent and should choose the one with the lowest rank as it is the closest to the root of the tree. When a node wants to send a message, it sends the message to one of its parent which will check if it knows a route to the destination or not. If it does, it sends the message to the right son. If it does not, then it sends the message to its parents which applies the same algorithm. If even the root of the tree does not know the route to this node, a DAO message will be broadcasted to look for the right node. Upon receiving a DAO message of which it is the destination, a node should send a DAO-ACK message to the root of the tree. All the parents along the way can store the route to this node if they have the RAM space to do so. When the DAO-ACK reaches the root of the tree, the message can be forwarded to the right son which will in turn forward it to the right son until it reaches the destination node. Nodes with a limited battery can refuse to become a parent by not answering DIS requests or by advertising a very large rank number to provide connectivity to only the nodes that have no alternative but using this node as a parent. It is also possible for a node to increase its rank to become less desirable to the sons. Nodes may also randomly select the parent among all its parents with the same ranks to route upward messages. Nodes can also periodically select another preferred parent. All these techniques allow

spreading the load on more nodes in order to increase the lifespan of the network.

Geographic routing algorithms such as GRP (Geographic Routing Protocol) [53] use GPS information to optimise the routing path. In the case of GRP, it is a reactive protocol that uses the GPS information to reduce the un-necessary control traffic.

CA-Path [54] is a routing algorithm that takes advantage of the broadcasting nature of wireless transmission to minimise the number of re-transmissions if a relay did not manage to receive the message but a neighbour did. This provides an interesting cognitive ability to routing that decrease the average power consumption and spectrum usage.

An evaluation of the GRP, AODV and OLSR routing algorithms for mobile ad hoc networks with a video conferencing application has been carried out in [55]. Depending on the conditions, the GRP, AODV or OLSR algorithms become best-suited. It is thus very important for an application to choose the routing algorithm that is best suited to the type of traffic and QoS needed.

2.2.3 Efficient protocols

Another way of saving power in green wireless networks is to reduce the size of the messages exchanged among the nodes.

The most effective way of reducing the message size is to reduce the size of the headers found in all messages. The IPv6 header is 40 bytes long while the UDP header is 8 bytes long. By using 6LoWPAN [56] instead of the standard IPv6, it is possible to compress the network header to 7 bytes and 4 bytes for the UDP header. This results in a 37 bytes saving as the new size of the headers is 23% of the original size. It however comes at a cost as only 16 UDP ports are available per node and a gateway needs to be installed to allow 6LoWPAN and IPv6 computers to communicate.

HTTP is a protocol that is widely used in the world-wide web to access resources. For instance, it has been used by SOAP to create an RPC protocol using XML. HTTP could be very useful in wireless sensor networks to query the state of a sensor or actuator. However, due to its high parsing complexity and big data size, using HTTP is not possible. CoAP [57] has been proposed by the Internet Engineering Task Force (IETF) to provide a RESTful protocol that can be mapped to HTTP by using a gateway and yet has a low parsing complexity and header size. Like HTTP, it also supports caching. However, unlike HTTP, a service can subscribe to a resource to get notifications every time the resource changed which avoids continuously polling on a resource to check for updates.

2.3 Computer architectures

Optimising every layer of the OSI model with the solutions proposed in the previous section is not enough to achieve the lowest energy usage possible in a network. In the end, the energy usage is always consumed by the hardware that runs the applications,

may it be for data transmission or processing. We thus introduce ways of making the hardware more efficient, may it be radios or processors.

A radio's efficiency is limited by the heavy attenuation of the communication channel. Overcoming this attenuation requires a lot of radiated power which has to come from the energy source. Improvements in power amplifiers however allow saving a lot of power while still meeting the output linearity requirements. Up to very recently, power amplifiers had to be powered at a voltage slightly higher than the maximum voltage it was supposed to output. All the voltage difference between the output and the input power would be dissipated as heat. Envelope-tracking technologies such as [58] allow supplying the power amplifier with just the right voltage at any time for it to run constantly at its peek efficiency. With such technology, Qualcomm managed to save 20% of power and reduce the thermal generation by 30% [59] on a 3G/4G LTE mobile interface compared to a previous version of the radio. Another proposed solution is to have multiple fixed voltages, select the most appropriate voltage and compensate for the non-linearity in software. This technique allowed to increase the total efficiency of a HSUPA transmitter from 17.7% to 40.7% and from 11.3% to 35.5% for WLAN 802.11g [60]. Using an improved version of this technology, Eta Devices managed to create an LTE transmitter with a 70% efficiency, compared to the 44 to 55% currently available [59].

Application developers used to mostly ignore the processor they were running on as its performance and power efficiency would increase exponentially due to improvements in the transistor etching process. Developers just had to wait for the required characteristics needed were met before shipping a project.

In 1975, Moore successfully predicted that the number of transistors found in microprocessors would double every two years [61]. In 2011, Koomey was able to observe that the efficiency of processor doubled every 18 months for the past 60 years. This observation is now called Koomey's law [62].

However, as transistors get smaller and smaller, their ability to function as a perfect switch diminish because of tunnelling quantum physics effect. Indeed, in quantum physics, the position of the electron is described as a probability density function. When the insulating part of the gate becomes too thin, some high-energy electrons can move to the other side of the gate. This leaky-faucet effect is responsible for what is known as the static power consumption. This effect could be mostly ignored until the year 2000 but is now becoming a prevalent part of the total power consumption [6].

The etching process of transistors is constantly improving to reduce the static power consumption but it takes several years of effort to propose a new generation. In parallel of this effort, hardware designers introduced different techniques to save power on their processors. A relatively simple technique consists in dynamically adjusting the clock frequency along with the supply voltage of processors. This technique is called Dynamic Voltage/Frequency Scaling (DVFS) [63] and does not require a complete redesign of the internal architecture of a processor. To further improve the power efficiency, two other

techniques are usually found in modern processors: clock gating and power gating. The first is about not supplying a clock to a block of transistor that does not need it [64] while the latter is about cutting the static power consumption by cutting the power of un-used blocks. In both cases, it is necessary to modify the hardware design to get the needed activity signal for those techniques to work.

The 3 techniques presented (DVFS, clock-gating and power-gating) require some intervention by the operating system to be useful. It is thus not possible anymore to wait for better hardware in order to solve the power consumption problem of a processor, the software also needs to get involved.

2.4 Autonomic Computing

As we saw in the previous sections, both the hardware and the networks are becoming more and more complex and, as such, they require more and more knowledge from the technicians trying to maintain them. Also, as the number of computer systems grows, less and less human resources can be used to administrate them. This likely results in a sub-optimal configuration which conflicts with the goals of green networking.

The vision of autonomic computing is the creation of systems which are able to manage themselves without the need for a constant human input [23]. In this vision, such systems would only receive high-level goals from humans and they would continuously configure themselves in order to meet these goals. This vision is inspired by biology where some actions are performed by our subconscious without needing any conscious thinking. This effectively hides the complexity of the system by only taking high-level inputs which, in turn, simplifies the administration of such a system. Automatic management also enables optimisation opportunities which can result in a higher performance and lower power usage.

This vision was first proposed by IBM in 2001 in the Autonomic Computing Initiative (ACI) [65]. To be considered autonomic, a system must implement the following self-management functions:

- **Self-configuration**: Create a functional configuration that may be sub-optimal but already fulfil the user's need;
- **Self-optimisation**: Improve the performance of the metrics of interest while lowering the power consumption when possible;
- **Self-protection**: Protect itself from adversaries while also protecting other users' information (confidentiality and integrity);
- **Self-healing**: Detect dangerous states for the system (overheating for example) and react to it to come back to a safer state.

Designing and deploying autonomic and collaborative systems would thus not only lower maintenance costs but also improve the quality of service, performance and increase both the power- and the spectrum-efficiency throughout a network.

2.5 Conclusion

In this state of the art, we introduced the concept of green networking in wireless networks. After explaining how RF communications are possible, we introduced optimisations at every layer of the OSI model to reduce the footprint of communications on both the power and the RF spectrum usage. We then introduced power saving techniques for the hardware used by the network nodes.

Because the complexity of modern hardware and green network architectures, it is unlikely that an operator could continuously optimise the network to adapt to changes in the activity. To ease the administration of the network and always use the most efficient power-efficient configuration, the nodes and the network should self-manage as proposed in the autonomic computing vision.

This chapter proposed an overview of techniques used in green networking to lower the resource usage of wireless networks. A more detailed state of the art will be introduced in the related chapters, starting with the highest layers of the OSI model, the application to the network layer.

Chapter 3

Application & Network: Decentralising data processing in surveillance networks

Contents

3.1 Introduction

Surveillance networks require multiple wireless sensors spread over a large area. It would be impractical for nodes to have the necessary power to be able to reach the computer that centralises the sensor's data directly. Surveillance networks thus usually use a multi-hop ad-hoc network topology.

Data transmission in a multi-hop wireless ad-hoc network plays a large role in nodes' power consumption [39] and it will comparatively grow larger and larger as computing costs go down thanks to the constant improvement in the processors' efficiency (Koomsley's law [62]). This is especially the case in Wireless Sensor Networks (WSNs) where computing power is hardly necessary because sensor nodes are usually only expected to keep the network connected, acquire data from their connected sensors, generate network frames containing the sensors' data and send them to a gateway.

The energy cost of transmitting 1 KB over a distance of 100 m is approximately the same as the cost of executing 3 million instructions by a 100 million instructions per second (MIPS)/W processor [66]. However, a transmission does not only drain the emitter's battery, it also increases the power consumption of surrounding nodes as they have to demodulate the incoming signal and process the decoded frame. It also increases the contention over the network which prevents radios from being asleep when they do not need to communicate because they need to check if the medium is free or not. Improving the lifespan of WSNs thus means diminishing both the number and the average size of the messages sent in the network, even if it means using more processing power to do so. This trade-of depends on the distance at which the message should be sent and how efficient the processor is.

As we will see, localizing data processing among the nodes of a network is the best way of reducing the number of communications and their size even if it creates other challenges.

In Section 3.2, we proposed the state of the art concerning data-processing schemes and network architectures in Wireless Sensor Networks along with their shortcomings. In Section 3.3, We introduce a modality-agnostic collaborative detection of spatio-temporally correlated events in a WSN which I evaluate in Section 3.4. Section 3.5 details how the proposed algorithms were implemented and deployed in a realistic scenario. We conclude on decentralizing data processing in Section 3.6.

3.2 State of the art

We identified 4 kinds of Wireless Sensor Network architectures with varying degrees of autonomic behaviour when it comes to data processing. We also try to categorise them by average communication length and how often messages are sent.

3.2.1 Direct communication

Wireless Sensor Networks started as simple ad-hoc networks with the information flow going from all the sensors to the gateway. Each sensor node collects data from the sensors, average/aggregate it and then send this data to the gateway. The gateway then either processes the information itself, stores it or sends it through the Internet to another server. This gateway node is often referred to as "the sink" as all the traffic ultimately is routed through it. This behaviour can be seen in Figure 3.1.

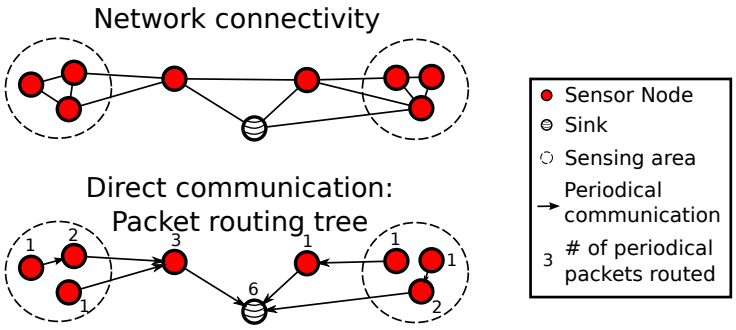


Figure 3.1: Example of a direct communication architecture deployment

In this kind of WSN, sensors' readings can be stored before being sent in batch to limit the number of packets at the expense of latency.

Examples of applications implementing this architecture include environment monitoring [67] and smart metering as the information they collect is meant to be stored by a centralised entity for processing.

The average communication length on this architecture is the average route length that links every sensor node to the gateway. As all sensor nodes send data periodically, this kind of network's lifespan depends on how often the sensors' data is sent to the gateway.

In this network type, data flows from the sensor nodes to the sink in a tree-like fashion. Network packets should be sent to the neighbour closest to the sink in order to reach it in as few hops as possible. A routing algorithm such as RPL [52] does exactly that as nodes advertise their rank (number of hops to reach the sink). A node should thus send data to the node with the lowest rank.

3.2.2 Cluster-based Data Aggregation

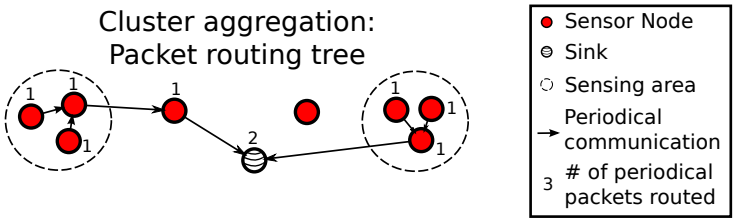


Figure 3.2: Example of a cluster-based aggregation deployment

A way to lower the average length of communication is to let a data-aggregation cluster node gather data from its surrounding nodes, aggregate it into a single message and send it to the gateway. This architecture can be seen in Figure 3.2.

This architecture introduces two kinds of communications:

- Short-distance: A direct communication (one hop);
- Long-distance: A multi-hop communication to the sink.

Long-distance communications should be avoided as much as possible to lower power consumption in the network. Indeed, they not only consume energy on both the emitter and gateway, but they also consume the energy of every routing node in-between. Moreover, nodes around the route are also consuming more energy because they need to receive and parse at least the MAC header in order to determine if they are the recipient or not. Finally, since the gateway is always the ultimate destination for every packet, the closer to the gateway a node is, the less likely the communication medium is to be available because of the ever-increasing traffic found when getting closer to the gateway. This further increases power consumption.

On the other hand, short-distance communications consume energy on a much more limited zone and more uniformly.

With the cluster-based data aggregation architecture, most communications are local which spreads the power usage more evenly in the network.

The LEACH [51] routing algorithm matches almost-exactly the needs for this type of network. It should thus be used.

3.2.3 Local Event Detection

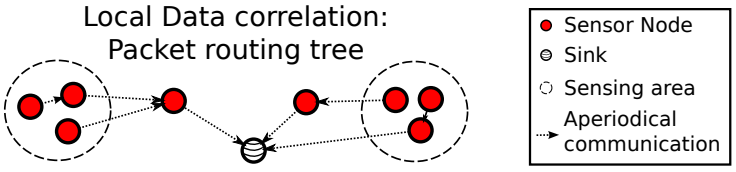


Figure 3.3: Example of a local event detection deployment

Instead of addressing the average communication length problem, local event detection addresses the problem of the number of messages generated.

Both the sink and the cluster-based data aggregation architectures rely on the outside of the network to process data. These architectures are data-oriented as they only acquire and forward sensors' data to the gateway.

If the WSN is used for real-time detection of events, then it is possible to reduce the number of messages by decentralising data processing and only sending sensors' data conditionally [68]. This aperiodical traffic can be seen in Figure 3.3.

For instance, in an intrusion detection application, it is not necessary to constantly send the raw values read by the sensors. Instead, local processing of sensors' data avoids unnecessary transmissions by letting the sensor node find out whether the sensor detects something important or not.

Local event detection may be challenging for sensor nodes because of their very tight CPU, ROM and RAM constraints, but it is possible to use digital signal processing units (DSP) [69] as these processors are getting more and more efficient [62].

Awareness of the WSN's application allows sensor nodes to limit the number of messages they emit by filtering the unneeded information [70][71].

As the data consumer is out of the network, packets still need to be routed to the sink. A tree-like RPL routing protocol is still the most efficient way to do so because it routes towards the sink in the fewest number of hops possible.

3.2.4 Collaborative Detection

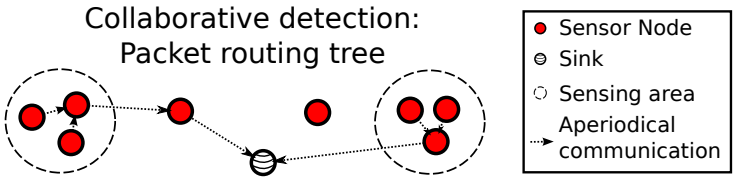


Figure 3.4: Example of a collaborative detection deployment

If sensors are spatially- and temporally-correlated, collaborative detection can build on the concepts of both local data aggregation and cluster-based data aggregation.

Sensors are first locally correlated to filter unneeded information. Useful data points, are then collected in a cluster in order to correlate them with all the sensors of a given area [72][73]. This behaviour can be seen in Figure 3.4.

This method further limits the message count by avoiding false positives induced by defective or ill-calibrated sensors. Less false positives also means reduced long-distance communications.

This method also works on heterogeneous sensor networks [74][75] and is also known to produce better results than value-fusion as found in cluster-based data aggregation when fault-tolerance is needed [76][77].

This architecture provides both a lower average communication distance and fewer messages in the network because of its aggressive filtering at both the node level and the cluster level.

However, to our knowledge, already-existing collaborative detection networks do not abstract the sensor modality. This severely limits the capability of adding new sensors at run time since the correlating node needs to be aware of all the sensor types and know how to correlate them.

Moreover, collaborative detection makes it more difficult for the network operator to know if the network is operating correctly because of the low volume of messages. In case of a detection failure, it also makes it harder for her/him to debug what went wrong because no logs are available on her/his work computer.

Finally, because of the correlation found in collaborative detection, it is difficult for the network operator to detect faulty sensors. It should thus be the cluster's task to monitor the false positive and false negative rate of each sensor and recalibrate its parameters to deal with those. Likewise, if a sensor node becomes unavailable, the cluster node should reconfigure itself to operate without the sensors connected to this faulty node.

In a collaborative detection network, data is processed in the network and events are sent when some conditions arise. Those events have a semantic attached to them and can be produced or consumed by any node in the network. Events should thus be routed from the producer nodes to the consumer nodes.

The RPL routing protocol is not suitable for this kind of network as it is meant for routing information to the nearest sink and it is not data-centric. Instead, we want that when an event is published, it should be dispatched to all the nodes that subscribed to it using as little packets as possible. A centralised version of this Publish/Subscribe routing algorithm has been proposed in [50] but we hope that a decentralised version of it will be proposed soon.

An evaluation of the performance of each WSN type is available in Table 3.1 which will be explained in Section 3.4.

WSN type	readings	short-distance	long-distance
Sink	5400	0 (0%)	5400 (100%)
Cluster aggregation	5400	3600 (67%)	1800 (33%)
Local detection	5400	0 (0%)	540 (10%)
Collaborative detection	5400	$\leq$ 540 (10%)	$<$ 180 (3.33%)

Table 3.1: Comparing WSN data management on a 3-nodes area with a sensor false positive probability (p=0.1, f=1Hz)

3.3 Contributions

Logical reasoning allow a system to make inferences using only logical deductions [78]. An automated logical reasoning can use prior knowledge and/or experience to improve

its reasoning abilities [79].

To address the shortcomings of the state-of-the-art collaborative detection, we propose three different reasoning components:

- modality-agnostic collaborative detection of spatio-temporally correlated events;
- offline logging capabilities;
- sensors reputation management.

3.3.1 Modality-agnostic Collaborative Detection of Spatio temporally Correlated Events

Design goals

This proposal is based on the following assumptions:

1. an area defines a spatial zone where all sensors are correlated and can communicate with each other;
2. area's sensors should all detect an intrusion within a definite maximum time called minimum intrusion time;
3. sensors are noisy and randomly emit false positives;;
4. sensors' may not be reachable at all time;;
5. sensors are ill-calibrated..

Assumptions 1 and 2 have to be enforced when sensors are deployed. It also means network's lifespan can be improved by using collaborative detection to lower both the number of messages and the average distance of communications.

Assumptions 3 and 4 mean that the network should be fault-tolerant and thus, be using decision-fusion [76]. The correlation node should also look for new sensor nodes in the spatial zone while also monitoring for failing nodes. Upon changes in the available sensor nodes, the correlation node should reconfigure itself to limit the number of false positives or false negatives.

Assumption 5 raises the problem of sensors correlation. Since non-calibrated sensors' values cannot be averaged, it means that the values read by the sensor can only be interpreted by the sensor node that produced it. Indeed, the value can only be interpreted when put into its context, that is to say, the previous values/trends. Moreover, decision-fusion mandates the sensor to produce a decision whether it detects an event or not.

Thus, instead of reporting a given value, a sensor needs to locally correlate its values then decides if an event is going on or not. If an event is detected, a confidence rating ranging from 1 to 3 is attributed to the detection. This proposal abstracts the sensor type and has many additional benefits such as:

- hiding away calibration errors from the correlation node;
- correlating sensors of arbitrary type.

Messages between the sensors and the reasoning node are sent using a Publish/Subscriber communication paradigm. This is done to ease the creation of topic of interests sensors can subscribe to. Moreover, a publisher does not have to know what its subscribers are interested in. This allows the creation of a highly-dynamic collaborative behaviour.

Sensor nodes

In this proposal, individual sensors are required to detect events by themselves and give a confidence rating ranging from 1 to 3. The algorithm that should be followed highly depends on the sensor modality and the event that should be captured. As an example, let's imagine someone wants to use a binary infrared optical barrier to detect a car driving down a street. The signal produced by the sensor would be a simple square signal whose length will depend on the length and the speed of the car.

To detect such a square signal, the sensor node can poll the sensor's value periodically. The polling period depends on how wide the optical barrier is and how fast the car is. The maximum speed of the event to detect is one parameter of the area. As soon as the polled sensor detects something, a correlation timer is set. If the value sensor keeps on detecting something for, for instance, more than a tenth of the minimum detection time, then a message (hereinafter referred to as "alert") should be sent to the correlation node with the minimal confidence level. The confidence level should then be increased gradually to 3 as long as the sensor keeps on detecting the intruder.

A minimum delay between alerts should be implemented in order to further reduce the number of messages. Doing so while keeping the correlation node up to date concerning the state of each sensor may require to break the re-emission rule if the confidence level increased. For instance, in Figure 3.5, we see the evolution of the value returned by an analog sensor. An alert is first sent when the value increased over a threshold and then re-emitted after a period. If the value changes in a fast way, an alert can be sent earlier.

The correlation node

When the correlation node receives an event from a sensor (alert), we propose that it stores both the current timestamp and the confidence level into memory for future reference. Then, it should compute the area's criticality level by summing the contribution of all the area's sensors. Being able to automatically trigger responses to an event allows the creation of a fully-autonomic collaborative WSN.

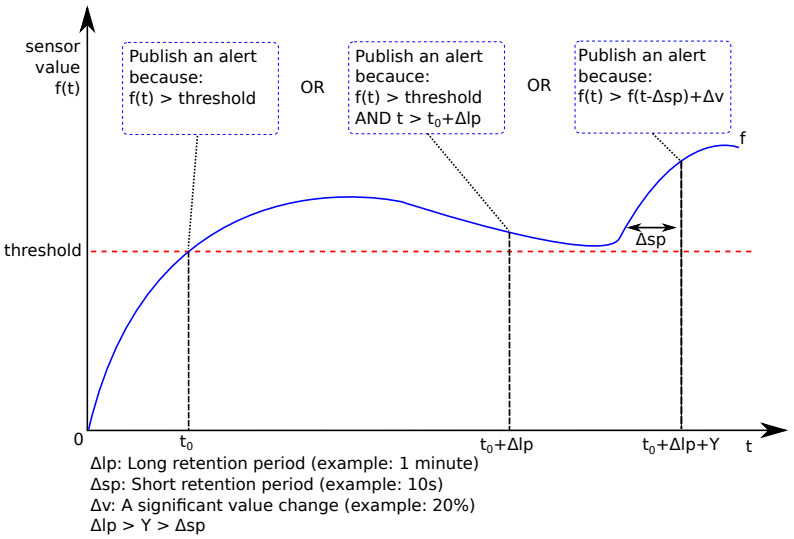


Figure 3.5: An example of alert re-emission policy depending on the evolution of an analog sensor's value

Being able to automatically trigger responses to an event allows the creation of a fully-autonomic collaborative WSN.

A sensor's contribution to the criticality level is the confidence level of the alert times the age factor. The age factor linearly decreases from 1 to 0 in *correlation_time* seconds. The correlation time should be roughly the same as the maximum time it takes for an event to happen. In an intrusion detection scenario, it is the maximum time it takes for a pedestrian to cross the area. If a sensor becomes unavailable, its criticality level should be raised to the maximum for a few minutes to prevent attacks on the system that involve disabling sensors remotely by shooting them.

The age factor is important because alerts should expire after some time. Moreover, older alerts should not influence the correlation as much as fresh alerts do. The choice of a linearly decreasing function is motivated by the simplicity of implementation. It may however need to be adjusted depending on the kind of application used in this system. Computing the area's criticality level is done following (3.1).

$$age\_factor(a) = \begin{cases} 0 & alert\_age(a) > correlation\_time \\ 1 - \dfrac{alert\_age(a)}{correlation\_time} & otherwise \end{cases}$$

$$\begin{aligned} alert\_age(a) &= current\_time() - alert\_timestamp(a) \\ contrib\_alert(a) &= confidence(a) * age\_factor(a) \\ criticality &= \sum_{a=0}^{alert\_count} contrib\_alert(a) \end{aligned} \tag{3.1}$$

When most sensors from an area simultaneously detect an event, the criticality level of the area should be higher than a threshold that depends on the number of sensors in the area, the minimal correlation factor we want to achieve and a latency mode indicating how early an alarm should be sent to notify the zone operator or to trigger an automatic response. The area's threshold is computed using (3.2) when a new sensor node is added to the area or after a sensor node has been unreachable for some time, 10 minutes for instance.

$$criticality\_threshold = sensors\_count * 1.5 * latency\_mode \tag{3.2}$$

$$latency\_mode = \begin{cases} 0.25 & Red\ mode \\ 0.50 & Orange\ mode \\ 0.75 & Yellow\ mode \\ 1.00 & Green\ mode \\ 1.50 & White\ mode \end{cases}$$

When the criticality level goes over the threshold, the correlating node of the area publishes a message telling that an event has been detected by the area. This message, hereinafter referred to as alarm, can then be used by other nodes to trigger automatic actions such as lighting up a spotlight or an alarm.

The 1.5 constant from Equ. 3.2 is the maximum confidence level (3) divided by 2. This means that when using the green latency mode, all the sensors should report an average of 1.5 before an alarm is sent.

An example of the evolution of an area's criticality level can be seen in Figure 3.6. At t_0, a node sent an alert which raised the criticality level. This criticality level then linearly decreased towards 0. At t_4, the contributions of alerts received at t_1, t_2, t_3 and t_4 raised the criticality level enough to reach the criticality threshold, which led to the emission of an alarm. At t_5, the correlation node received another alert which contributed more than the alert at t_0 because the alert had a higher confidence level.

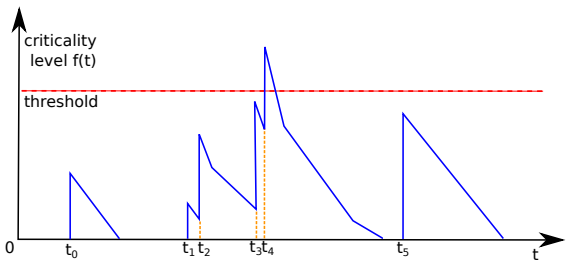


Figure 3.6: Example of the evolution of an area's criticality level

We also propose that the first node to be added to an area should be "elected" as the correlating node. However, this node may not be available during the whole lifespan of the network for the following reasons:

- energy source depletion;
- hardware malfunction;
- selective jamming on this node.

To overcome these challenges, the correlation node role should be duplicated in two. Both correlation nodes would subscribe to the alerts sent in their area. The only difference between the two nodes would be that the master node (the first one elected) would emit an alarm as fast as possible. On the contrary, the slave correlation node (the second node to be added to the area) would wait for a few seconds before emitting an alarm unless the master node emits the alarm before the expiration of the delay.

This proposal increases reliability by avoiding the single point of failure that was the correlation node. However, a re-election should be made whenever the master correlation node's battery is running low or when the slave correlation node detects that the master is not behaving correctly. In this case, the slave should elect himself as the master and query a list of potential candidates in order to select a suitable slave node to succeed him. If the slave node becomes unavailable, the master node should re-elect another slave node.

Many clustering algorithms exist for wireless sensor networks [80]. We however propose that the election mechanism for the correlation node should be made according to these criteria:

- energy available (in Joules, not percentage);
- number of hops to the gateway;
- number of routes to the gateway.

Redundancy could then be further improved by electing n correlation nodes with an increasing delay between detection and the emission of an alarm. This works because every node of the area can communicate with each others (assumption 1).

However, redunding the correlation node comes at the expense of power consumption and thus, the network lifespan. Indeed, every correlating node is required to subscribe and receive every alert and alarm sent by the area and process them. As a result, the average power consumption of the correlating nodes is higher than other nodes in the area. This means there is trade-off between battery life and redundancy.

3.3.2 Offline Logging Capabilities

Due to the drastic message reduction found at the gateway when using in-network reasoning, auditing the system becomes difficult. Auditing is needed by the network operator to understand what is going wrong with the system in case of false positives or false negative detections.

We consider a false positive happened when the network emitted an alarm when no real event happened. Likewise, a false negative happened when the network did not emit an alarm when a real event happened. At a sensor's level, a false positive happens when the node sent an alert when no meaningful event happened and a false negative when it did not send one when it should have.

A false negative can happen when the sensor did not detect the event (wrong polling frequency on the sensor or wrong calibration). A false positive can happen when the sensor's sensitivity is too high or if detected an event that was not meaningful to the network (a dog instead of a pedestrian for instance).

To address this problem, we propose that the correlating node should be required to store the history of what happened in an area in the past hours or days. Given the stringent constraints found on sensor nodes, it is impossible to store all the data. It is thus important to find an efficient way to only keep meaningful data.

The proposed solution stores events. An event is characterised by a local time stamp, a confidence level and the list of sensors contributing to this event and their relative contribution to it.

The system can only store a selected few events. Events older than the history storage will first be deleted. When a new event needs to be stored, a usefulness score is attributed to each event in the history. The event with the lowest score gets replaced by the new event.

An important event is an event that both drove the confidence level close to the alarm threshold and was also a local maximum for a long time. However, when an event gets older, its importance tends to lower. This is why the scoring system (3.3) depends on the confidence level of the event, how long it was a local maximum and the age of the event.

$$score(e) = \frac{confidence(e) * local\_max\_time(e)}{1 - \frac{age(e)}{max\_storage\_time}} \tag{3.3}$$

An event is not stored in the history unless it reaches the history threshold. An example of the evolution of an area's criticality level can be seen in Figure 3.7.

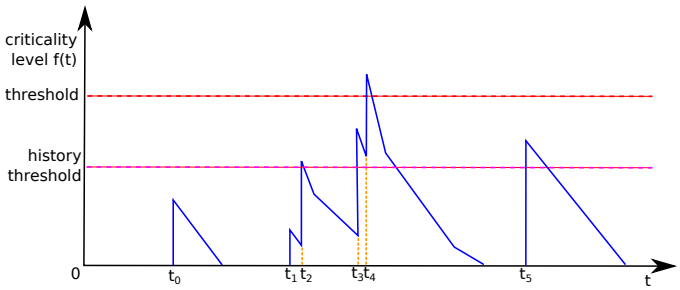


Figure 3.7: Example of the evolution of an area's criticality level with regards to the the history threshold

This Figure features the same criticality level evolution as in Figure 3.6 with the addition of the history threshold. In this example, only the events at t_2, t_3, t_4 and t_5 will be stored to the history because the others did not increase the criticality level enough to reach the history threshold. The most significant entry would be the one found at t_4 because it is the one with the highest criticality level and it has been the local maximum for a long time. The least significant event in this example is the one from t_3, despite its relatively-high criticality level. This is because it has been the local maximum for a very short period. This event will thus be the first one to get deleted if space was becoming a problem.

The history along with all the other parameters and context can then be queried on-demand by a network operator using a REST-like protocol such as CoAP [57]. This improves the network's debugging abilities without needing an external radio to spy on the exchanged messages.

3.3.3 Sensors Reputation Management

The history is not only useful to the network operator, it is also useful to the WSN itself. An obvious usage of the history is to juge how useful a sensor usually is at detecting a certain type of event. We refer to this usefulness score as reputation.

Reputation is separated in two scores:

- False positive: How often are the events emitted by a sensor not correlated with his surrounding nodes;
- False negative: How often is a sensor not participating in the correlation of real events.

Both reputations are represented by a value ranging from 0 to 1, 0 being the lowest possible reputation and 1 being the best one.

$$reputation\_fp(a, s) = \frac{area\_detection\_count\_involving(a, s)}{sensor\_events\_count(s)} \tag{3.4}$$

False positive reputation is calculated when alerts are deleted from the history. When an alert is deleted, the detection count (sensor_events_count(s)) of the associated sensor is incremented. If the alert has been used to emit an alarm, then the alarm counter of the associated sensor (area_detection_count_involving(a,s)) is incremented. The false positive reputation is just the ratio of alerts that have been correlated over the total alert count for a given sensor. The equation is detailed in (3.4).

$$reputation\_fn(a, s) = \frac{sensor\_correlated\_count(s)}{area\_detection\_count(a)} \tag{3.5}$$

False negative reputation is calculated when an event is added to the history. If the event is an alarm, then the alarm count (area_detection_count(a)) is incremented. Then, all the sensors that participated to this alarm have their correlation count incremented (sensor_correlated_count(s)). False negative reputation is the ratio of how many times a sensor was involved in the emission of alarms over the total alarm count, the equation is detailed in (3.5).

Additionally, the reasoning node could alert the operator when one sensor gets one of his reputations lower than a certain threshold. It can also be used during the correlation process to weight a sensor contribution according to its false positive reputation. A lower false positive reputation would result in a lower contribution during the event correlation.

Finally, we propose that sensors producing too many false positives should be evicted from the correlation process to avoid polluting the history of the correlation node and reduce the number of false alarms. Likewise, we propose that if a node has too many false negatives, they should be evicted or simply not taken into account when calculating the threshold. We propose those binary decisions because the area's threshold computation should be stable which prevents using the false positives/negatives values for the criticality and/or the threshold computation.

3.3.4 Summary

The decision process we proposed in our modality-agnostic collaborative detection of spatio-temporally correlated events proposition has two levels, as can be seen in Figure 3.8.

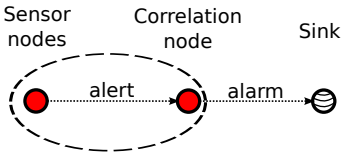


Figure 3.8: The two levels of reasoning

The first level of reasoning aims at sending the minimum amount of information as possible to keep the current view of the correlation node up to date without streaming a

continuous stream of data. The first level thus runs on all the nodes hosting one or more sensors and aim at detecting events using only values returned by the sensor. When an event happens, the node hosting the sensor sends an alert to the correlation node with a confidence level ranging from 1 to 3.

The second level of decision-making aims at correlating the alerts sent by the sensors of the area before sending alarms to further reduce the number of messages sent in the network. This also attaches semantic to the messages which allow automatic an automatic response by the network.

Since the correlation node is filtering most of the messages of the area, it would be impossible for the operator of such a network to audit the cause for a false positive or a false negative. To enhance the auditing capabilities of the network, we proposed storing a history of the most important events that happened in the network for a defined period of time.

Finally, we proposed a reputation mechanism to detect faulty sensors automatically to alert the network operator. This mechanism can also be used inside the network to evict faulty sensors from the correlation process.

3.4 Evaluation

This research was conducted during the DIstributed Applications and Functions Over Redundant Unattended Sensors (DIAFORUS) ANR project [81]. The project's goal was to develop an energy-efficient framework for distributed applications and functions over redundant unattended sensors. As a demonstration of the framework, an intrusion detection application was written using redundant and heterogeneous sensors.

For evaluating the contribution, we used a physical intrusion detection scenario. The latency of detection should be under 10 seconds and usually around 5 seconds. It was decided that sensor nodes should not correlate values for more than 1 second before sending it to the network. This leaves up to 4 seconds for the network to carry the detection message and its acknowledgement (ACK). The event is re-emitted if no ACK is received by the sensor node that emitted the detection message.

It is however unnecessary to flood the network with alarms when an intrusion is detected. Two alarms can be separated by at least the area's minimum crossing time unless a new sensor has been correlated with an existing alarm. This amendum to the rule is made so as the operator gets new meaningful information in the timeliest fashion. This proposal enables the operator to be aware of the current situation while also limiting the number of messages.

The evaluation has been carried out in a custom-made emulated environment based on FreeRTOS. Each emulated sensor node is executed as a FreeRTOS Linux process. When a sensor sends a message, instead of sending it through the radio like it would be done

on real nodes, the message is sent using TCP sockets to a program called dispatcher. The dispatcher emulates the communication medium. It forwards the message to the destination node if nodes are within reach of each other.

The dispatcher gets the sensor nodes' location from an XML deployment file we designed. This file not only lists the sensor nodes, their positions and their radio range, it also contains all the sensor-specific configuration such as what sensors (modality) are connected to each sensor node and how (I2C address, gpio line, ...). This deployment file is used to generate the node configuration as C header files that are then compiled and linked with the node's firmware. This compilation process is controlled by a custom-built python script called "build_network.py" which can also be used to flash the firmware of all the nodes. An example of such a file is available in Appendix C.

For the network layer, we decided to use IPv6 Low Power Wireless Personnal Area Networks (6LoWPAN) instead of IPv6 because of its reduced header size.

Routing is handled using RPL [52], an IPv6 Routing Protocol for Low-Power and Lossy Networks, because it is becoming a standard in wireless sensor networks. We use to query nodes via the REST protocol CoAP [57] and route PubSub messages from/to the broker. RPL's DIO mechanism is used as an heartbeat indicating to surrounding node that the node is still available. This heartbeat is used to detect faulty nodes.

In simulation, the values read by the sensors are generated using a simulation environment called Diase. This environment, developed for DIAFORUS, enables:

- the deployment of a simulated network;
- the creation of intrusion scenarios;
- the monitoring of both sensor nodes and the network.

The general interface of Diase while running an intrusion detection scenario can be seen in Figure 3.9. It showcases 2 areas containing 8 nodes, 3 seismic sensors, 2 infrared directional barriers sensors, a multi-directional barrier sensors and 1 actuator (an alarm).

The monitoring view gathers data from the network using a REST protocol for Constrained Applications called CoAP [57]. It then displays the gathered data into a textual or a graph form. Examples of such graphs can be seen in Figure 3.10. More information about Diase and all its capabilities is available in Appendix D.

3.4.1 Modality-agnostic Collaborative Detection of Spatio temporally Correlated Events

Distributed and collaborative detection is meant to lower both the number and the average length of communications. We evaluate those metrics in this subsection.

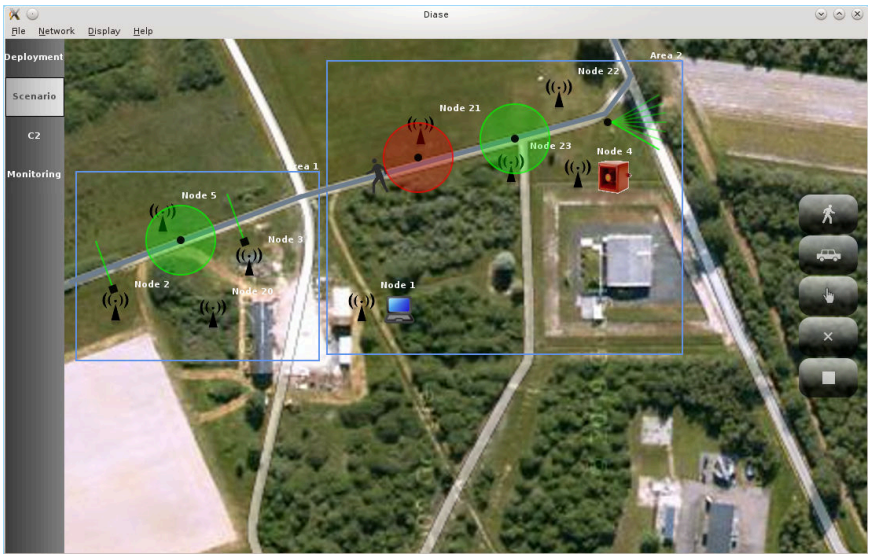


Figure 3.9: Screenshot of Diase running an intrusion scenario in real time and injecting simulated events to the emulated wireless nodes.

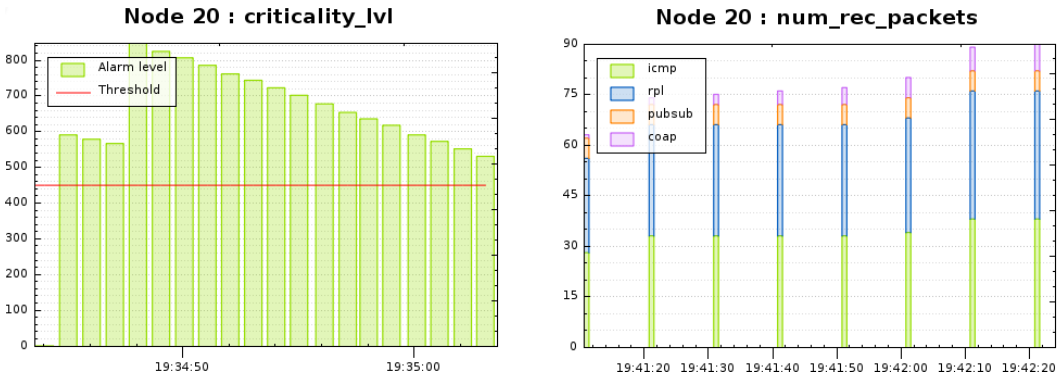
Sensors are never perfect, if calibrated to detect the smallest event they will be prone to false positives. We choose to model the detection of an event by a sensor using a Bernoulli distribution. When no real intrusion is going on, sensors will have a probability p of wrongly detecting an intrusion and a probability of $1 - p$ of not detecting one. The experience is repeated every second.

This means there is a probability p of getting a false positive. Knowing this, We evaluate how many messages are exchanged in different kinds of Wireless Sensor Networks. A distinction between short-distance (one hops) and long-distance (towards the gateway) communications is also introduced. The number of messages is then compared to the number of values read by the sensor. This comparative study is done for an area of 3 nodes, reading values every second for 30 minutes. The results can be found in Table 3.1.

In the *tree topology* WSN, all sensors readings are forwarded to the gateway every second to fulfil the 1 second correlation time rule. This only generates long-distance traffic and is the worst possible case.

In the *cluster data aggregation* WSN, 2 of the 3 sensors send their values to the aggregation node. This node aggregates these 2 values with the value of his sensor in one message before sending it to the gateway. This means 67% of the traffic is local and 33% of the traffic is long-distance.

In the *local event detection*, sensors detect intrusions themselves. When they detect a suspicious event, they forward the acquired value to the gateway. As the sensor's probability to detect an event is 10%, then 10% of the values read are forwarded to the



(a) Criticality level of an area **(b)** Types of packets received

Figure 3.10: Example of the real-time visualisation capabilities of Diase

gateway and thus, be considered as long-distance communications. No short distance traffic is generated.

In the *collaborative detection* WSN, when sensors detect an event, it is sent to the correlation node. This node then correlates these events that expire after a certain amount of time that depends on the maximum time it takes to cross the area. This means up to 10% of the communications will be local. This value depends on the number of sensors connected to the correlation node. Telling how often a long-distance communication will happen is non-trivial because this highly depends on the algorithm used to correlate the local events.

We now move on to testing the influence of the sensor noise and the correlation time on the number of short-distance and long-distance communications in our proposal. All these communications are false positives, so lower is better. We simulate the worst case scenario, when no sensors are connected to the correlation node.

In Table 3.2, we evaluate the influence of the correlation time over the number and length of communications. As expected, the number of local detections is around 10%, which is the sensor noise. Concerning the long-distance communications, the longer the correlation time, the higher the number of messages. This can be explained because the longer the correlation time, the higher the probability of correlation between sensors. Areas should then be as small as possible to limit the number of false positives.

In Table 3.3, we evaluate the influence of the sensor noise over the number and length of communications. As expected, the number of local detections is linear with the sensor noise. Long distance communications count is increasing with the sensor noise.

In tables 3.2 and 3.3, the number of long distance communications do not scale linearly with the correlation time and noise probability because of the no-alarm-re-emission policy that is meant to limit the number of messages. In the case of Table 3.3, when p=1, we got an average of an alarm every 8.5 seconds even though the minimum intrusion time was

Correlation	readings	short-distance	long-distance	$\frac{long}{short}$ ratio
c=10s	5400	545 (10%)	8 (0.15%)	1.5%
c=20s	5400	558 (10.3%)	23 (0.43%)	4.1%
c=40s	5400	541 (10%)	32 (0.59%)	6%
c=60s	5400	516 (9.5%)	33 (0.61%)	6.3%
c=90s	5400	525 (9.7%)	40 (0.74%)	7.7%
c=120s	5400	518 (9.5%)	41 (0.76%)	7.8%
c=150s	5400	552 (10.2%)	50 (0.93%)	9%
c=180s	5400	520 (9.6%)	56 (1.03%)	10.7%

Table 3.2: Message count in DIAFORUS with noisy sensors (f=1Hz, p=0.1) and a correlation time c. Experiment time of 30 minutes.

Sensor noise	readings	short-distance	long-distance
p=0	5400	0 (0%)	0 (0%)
p=0.002	5400	11 (0.2%)	0 (0%)
p=0.02	5400	94 (1.7%)	0 (0%)
p=0.1	5400	545 (10%)	56 (1.03%)
p=1	5400	5400 (100%)	210 (3.89%)

Table 3.3: Message count in DIAFORUS with noisy sensors (f=1Hz, p) and a correlation time of 180s. Experiment time of 30 minutes.

set to 10 seconds. The 15% difference is due to the amendum to the no-alarm-re-emission policy that states than an alarm could be re-emitted if a new sensor was correlated with the on-going alarm.

With a relatively small correlation time (60s) and a relative low probability of false positive (2%), no alarms have been emitted during these 30 minutes. In normal situations, both the number and the average length of communications were lowered compared to non-collaborative approaches. Sensors' average noise determining the minimum number of messages that will be sent. In the worst case scenario studied, the number of messages is increased by 3.89% compared to the "Tree-topology WSN" architecture while the average communication distance tended towards one hop which is a considerable improvement on large scale networks for both latency and power consumption.

3.4.2 Sensors Reputation Management

In the previous subsection, we investigated the influence of the average sensor's noise and the maximum intrusion duration on the average communication length and the number of messages. We saw that most of the time, the reasoning node was able to filter out false positives as long as sensors are not too noisy. In the case of a single faulty sensor, it is likely that the operator will never receive any information concerning this sensor. We propose using our reputation contribution to make sure the operator gets

this information by sending a "faulty-sensor" pubsub message containing the ID of the faulty sensor when the false positive or the false negative reputation drops below 0.5 and there at least 3 detections.

We created one scenario with sensors that are not correlated in each area to validate both the false positive and the false negative computation. It features 2 areas which can be seen in Fig. 3.11.

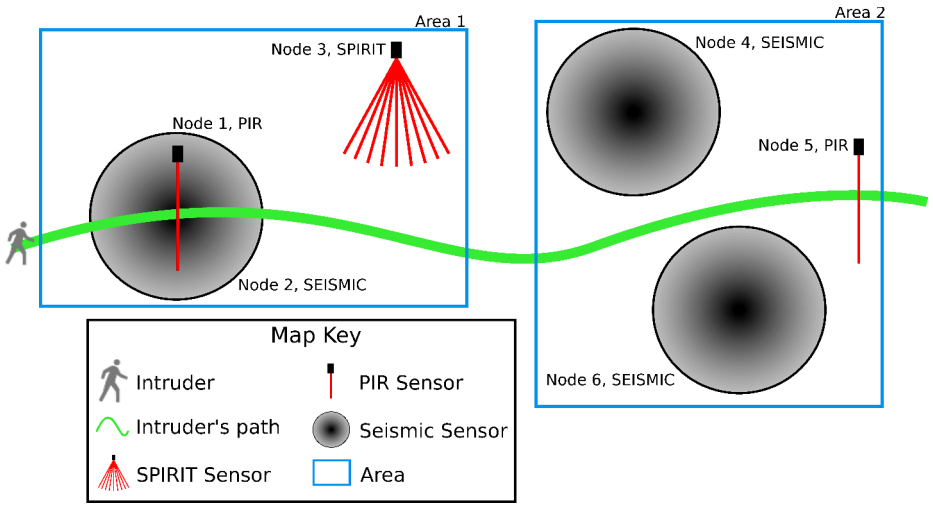


Figure 3.11: Scenario validating the reputation. 2 areas, 6 sensors.

Area 1 is composed of sensors 1, 2 and 3 and is meant to test false negatives. An intruder is repeatedly detected by sensors 1 and 2 but is never detected by sensor 3. No sensor noise was added to ease validation. After some time, nodes 1 and 2 are expected to have both a perfect false negative and a perfect false positive reputation. However, node 3 is expected to have a perfect false positive reputation but the lowest false negative reputation because it never contributed to any alarms.

Area 2 is composed of sensors 4, 5 and 6 and is meant to test false positives. An intruder is repeatedly detected by sensors 5 but is never detected by sensors 4 and 6. Again, no sensor noise was added to ease validation. After some time, nodes 4 and 6 are expected to have both a perfect false negative and a perfect false positive reputation. However, node 5 is expected to have a perfect false negative reputation along with the lowest false positive reputation because none of the alerts it sent led to emission of an alarm.

Results found in Table 3.4 perfectly match the expected reputation values. These results demonstrate the ability of our proposal to detect both the false positive and the false negative cases.

Node ID	False positive reputation	False negative reputation
1 & 2	1 (18/18)	1 (283/283)
3	NaN (0/0)	**0 (0/283)**
4 & 6	NaN (0/0)	NaN (0/0)
5	**0 (0/9)**	NaN (0/0)

Table 3.4: Results of the reputation experiment found on Fig. 3.11

We then simulated the impact of noise (p=0.1, f=1Hz) on the sensors' reputation. No simulated intrusion is running during this experiment. The results are shown in Table 3.5.

Node ID	False positive reputation	False negative reputation
1	0.34 (57/168)	1 (119/119)
2	0.34 (57/167)	1 (119/119)
3	0.34 (57/169)	1 (119/119)

Table 3.5: Reputation of noisy sensors (p=0.1, f=1Hz) after 30 minutes and correlation time of 20 seconds

With a noise probability (p=0.1, f=1Hz), sensors apparently sent an alert every 10.7s in average. With a correlation time of 20 seconds, it was pretty likely for all sensors to be participating in all alarms that were sent during these 30 minutes. This explains the false negative reputation of 1 for all three nodes.

As expected, the false positive reputation of sensors 1, 2 and 3 is relatively low. It would be even lower if not all sensors were faulty or if not all sensors were so noisy because less alarms would have been sent.

3.5 Real-life deployment

The results presented earlier have been obtained using a simulated network. However, in the DIAFORUS ANR project, we ported this network on real nodes and validated the simulation results. The algorithms used for the simulation are the ones used on the real nodes. This means our proposal is feasible on standard sensor nodes. The hardware used and the tested scenarios are presented in Appendix E.

3.6 Conclusion

In this chapter, we demonstrated a modality-agnostic collaborative detection of spatio-temporally correlated events. Correlating sensors' values inside the network instead of forwarding data to the gateway achieves a drastically lower and more evenly-spread power consumption by both reducing the amount and localising communication.

Our contribution enhances the state-of-the-art collaborative networks by abstracting sensors which allows the reasoning node to correlate ill-calibrated heterogeneous sensors. It also allows short-loop automatic responses to events detected in the network. This abstraction finally eases the creation of better auditing capabilities that overcome the debugging problems found in decentralized data processing by providing introspection inside the network and nodes. These capabilities can be used by both the operator and the network itself to, for instance, automatically react to faulty sensors, in a true autonomic fashion.

Future work will focus on improving the reputation management by allowing the WSN operator to report false positives and false negatives. This would increase the accuracy of the sensors' reputation by letting the operator specify when an intrusion happened and the criticality level did not rise enough to reach the threshold. The possibility of improving the criticality threshold computation by taking into account the reputation of each node to make sure the threshold can be reached will also be investigated. A study should also be carried out to evaluate the power consumption cost of adding redundancy for the correlation node. Finally, the influence of the sensor density over the number of communications will be studied.

Chapter 4

PHY/MAC: Efficient spectrum sharing

Contents

4.1 Introduction

After providing in the previous chapter a coherent contribution to green networking involving the application layer down to the network layer, we continue to go down the OSI stack to study how to improve the communication capabilities in wireless networks.

Nodes of a wireless network generally use the radio frequency (RF) spectrum to communicate using techniques we described in the state of the art. However, the modulated RF signal propagates itself from the source to the destination(s) through a shared communication medium.

Sharing this medium is difficult because of its heavy attenuation compared to the one usually found in wired communication mediums. This means that every node of the network has a different view of the spectrum. It is thus not possible for a node to make the assumption that if the medium is free on its side, that it is free for all the nodes that are supposed to receive the transmission ("hidden-node" problem). This is problematic because when two incoming RF signals of similar power interfere with each others at a receiving node, the node's radio cannot demodulate and decode any of the transmissions. The two transmissions are then said to have collided.

Collisions are thus the enemy number one of wireless networks because they use up bandwidth without transmitting any meaningful information. They also increase the communication delay as messages need to be sent again after a random back-off delay growing exponentially to avoid congesting the medium even more.

To increase the reliability and the immunity to outside interference, it is possible to use error detection and correction codes such as the Forward Error Correction (FEC) codes. Such codes can correct messages with up to n bits inversions and be able to detect m, with $n <= m$ at the cost of added redundancy in the message. However, these correcting codes are not sufficient to allow a receiver to correctly decode a message if multiple transmissions happened at the same time in its vicinity.

Making communications more resistant to outside interference requires spreading the communication on a wider frequency spectrum. The most resistant technique is called Frequency-Hopping Spread Spectrum (FHSS) [82]. It requires the emitter to split the message in sub-messages that will each be sent on a different frequency. The receiver needs to listen on the right frequency at all time which means the receiver has to know the hopping sequence in advance and need to be synchronised with the emitter. Coupled with error correction codes, a message sent in FHSS could be resistant to multiple channels being non-free during the transmission, depending on the code used. This leads to an increased reliability of the communication link.

Another way of increasing the reliability of transmissions is to use Direct-Sequence Spread Spectrum (DSSS) [82]. This technique increases the bandwidth of the transmission but increases the resistance to intended and unintended jamming thus allowing

multiple users to share the same channel. This technique can only be used in multi-MHz-wide bands. If the transceiver is operating in a narrow frequency band, FHSS could be used. DSSS and FHSS are the most common ways of increading the resistance to jamming but other solutions exist.

All the presented solutions increase the likeliness of receiving a message but it is still possible to completely block their communications. Evading crowded or perturbed frequency bands is difficult on a traditional radio because it is not possible to be both available to receive messages and to look for non-crowded frequencies (sensing) without adding a preamble longer than the time it would take for radios to check the occupancy of another band. Long preambles are not a good solution as they needlessly use the channel longer than actually needed, increasing the pressure on an already-limited resource. Software radios are a solution because they can perform sensing in a wide frequency band while receiving one or multiple communications in the sensed frequency band.

Contrarily to hardware/traditional radios, software (SW) radios do not demodulate the signals themselves, they merely sample the voltage found at the antenna or the output of a mixer. This allows computers connected to the software radio to listen to any frequency band available in a much larger tunable frequency range. The width of the frequency band can also be controlled by how often the voltage is sampled. This allows them to analyse their surroundings and become true cognitive radio nodes (CR).

In this chapter, we propose a PHY- and a MAC-layer signalling protocol to take advantage of the capabilities of SW radios in order to create a network resilient to unintentional jamming. Section 4.2 introduces the state of the art relative to software radios and cognitive nodes. Detecting transmissions using SW radios is detailed in Section 4.3. Our propositions for the PHY and MAC signalling protocols are respectively found in Sections 4.4 and 4.5. We conclude in Section 4.6 and present the future work.

4.2 State of the art

4.2.1 Software-defined radios

In essence, a software-defined radio is simply composed of a device that converts electromagnetic waves into voltage (an antenna) and then connects it to an Analog-to-Digital Converter (ADC) to sampling the voltage very rapidly. An ideal radio is shown in Figure 4.1.

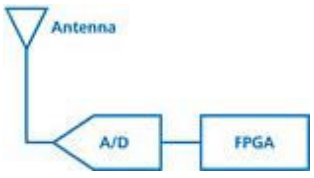


Figure 4.1: A simplified vision of a software-defined radio

Such a radio would work as long as no frequency received by antenna would be higher than half the sampling frequency (fs) of the ADC to avoid aliasing effects, as described by the Nyquist frequency. For instance, in baseband processing, a sine at x Hz above $fs/2$ would "generate" two sines of respectively x Hz and $(fs/2) - x$ Hz after sampling. To avoid this case, a low-pass filter should be added before the ADC to attenuate every frequency above half $fs/2$.

ADCs have a finite dynamic range as they output samples with a fixed number of bits to represent them. If an ADC outputs n-bit samples and the maximum voltage it can receive is V_{max}, then the minimum voltage that can be registered is $\frac{V_{max}}{2^n}$.

As the received power at the antenna is usually extremely low and ADCs are quite sensitive to noise, it is better to first boost the voltage using an analog low-noise amplifier before sampling it.

Another problem with the presented design is that ADCs cannot reach multi-GHz sampling rates at a reasonable cost. It is also almost impossible to handle such a quantity of data in real time without using a super computer. It is however possible to use a mixer to modulate the received signal with a local oscillator (LO) to shift the frequency of the signal found at the antenna, allowing us to move the RF window of interest around the 0 Hz frequency. The LO's frequency determines by how much the spectrum is shifted. This operation however creates a complex output which forces the radio to add another ADC sampling the voltage at a 90° phase difference to get both the real and imaginary samples (which we will later refer to as I and Q). Because a sample is now a complex number, the Nyquist frequency states that the maximum frequency allowed is the same as the sampling frequency.

Instead of using a fixed LO frequency, it is possible to use crystal to generate a fixed frequency which can be multiplied before being divided using a Phase-Locked Loop (PLL). We will introduce PLLs further in Chapter 5.

After all the proposed modifications, we obtain the radio found in Figure 4.2. This radio allows us to listen to a specific window in the RF spectrum. The central frequency is specified by changing the LO's frequency and its width can be changed by varying the sampling frequency.

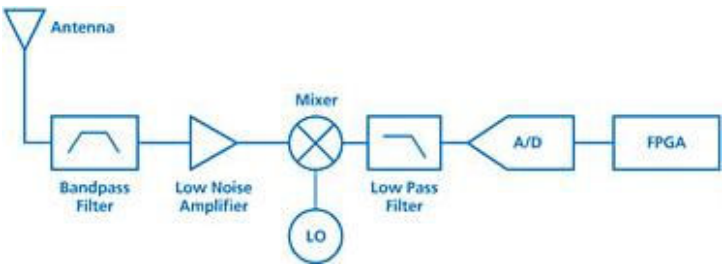


Figure 4.2: A simplified vision of a software-defined radio

For instance, the bladeRF [83], from the company Nuand, can receive or emit signals in a frequency band of up to 28MHz with a central frequency ranging from 300MHz to 3.8GHz. The computer can then perform the sensing operation to detect transmissions happening inside this 28MHz window and also decode them at the same time by copying the received signal and filtering it with a band-pass filter around each transmission. With a software radio, it is thus possible to evaluate the interference found in the spectrum window while receiving transmissions.

In Table 4.1, we compare the characteristics of different software-defined radios. The sample rate, given in mega samples per second, varies by two orders of magnitude between the cheapest and the most expensive radio. The interface that links the radio to the computer is also important since it has to have a throughput sufficient to stream all the RX and TX samples to/from the main computer.

Name	Tunable band	Sample Rate	RX/TX	Interface	Price
RTL2832U	[24, 1766] MHz	2.4 MSps	RX-only	USB2	$10
HackRF One	[10, 6000] MHz	20 MSps	Half duplex	USB2	$299
BladeRF	[300, 3800] MHz	40 MSps	Full duplex	USB3	$420
USRP B200	[70, 6000] MHz	56 MSps	Full duplex	USB3	$675
USRP X300	[DC, 6000]* MHz	200 MSps	Full duplex	PCIe	$3900

Table 4.1: Comparing different software radios. The USRP X300 cannot listen from DC to 6GHz continuously, it requires the appropriate daughter-boards which all have a smaller tunable band.

We did not however compare the size and the speed of their FPGAs nor the speed of their embedded microprocessor which would allow us to offload some operations.

4.2.2 Cognitive Radio Networks

The simplest form of cognitive radio network uses a Common Control Channel (CCC) that is used to initiate communications and exchanging sensing information [84][85]. The drawback of using a CCC is that it presents a single point of failure and does not scale well with multi-hop ad-hoc networks. It is thus better not to use a CCC although it presents other difficulties [86]. Indeed, in the absence of a CCC, transmissions could happen anywhere in the RF spectrum. In order to communicate, two transceivers first need to find each others by using a "blind rendez-vous" technique [3] which can guarantee two receivers will find each others if they have one available channel in common. Nodes willing to rendez-vous should all agree on a channel list and follow the jump sequence given by the algorithm. This requirement is hard to enforce in software radios because each CR may have a different list of available bands which are unlikely to translate to the same channel. However, the major problem is that when a CR is willing to find other CRs, it should start following the hopping pattern given by the algorithm which is incompatible with communication with other CRs.

In [4], the proposed MAC protocol provides a decentralised and CCC-free rendez-vous mechanism which also uses the Signal-to-Noise Ratio (SNR) in order to use the fastest modulation achievable. This approach is the most resilient one we found although it assumes that CRs already know their surrounding nodes and that a channel list is shared between them. The first assumption can be satisfied by adding a beaconing mechanism where CRs broadcast their presence. However, the second assumption is problematic because SW radios usually require knowing the central frequency of a transmission to be able to decode it. Not having a channel list available means a different way of detecting transmission is required. This approach however uses spectrum for every frame as the synchronisation is limited to transmitting one frame. Finally, this proposition does not directly allow a CR to receive or emit multiple frames at the same time.

Implementing MAC protocols in a SW radio has additional challenges coming from the fact that samples are not processed close to the ADC like in usual radios [17]. Samples are usually sent to a computer using a non-real-time communication medium, such as USB or Ethernet, and processed on a non-real-time operating system. A lot of buffering is thus necessary to keep the radio fed with the samples. This buffering increases the latency and the variability of the emission and reception time. Signalling protocols in CRNs need to be able to cope with this variability.

4.3 Detecting transmissions with software radios

The way transmissions can be detected and decoded using SW radios is quite different from traditional radios. Since our propositions heavily make use of these differences, we briefly introduce spectrum sensing and our software infrastructure.

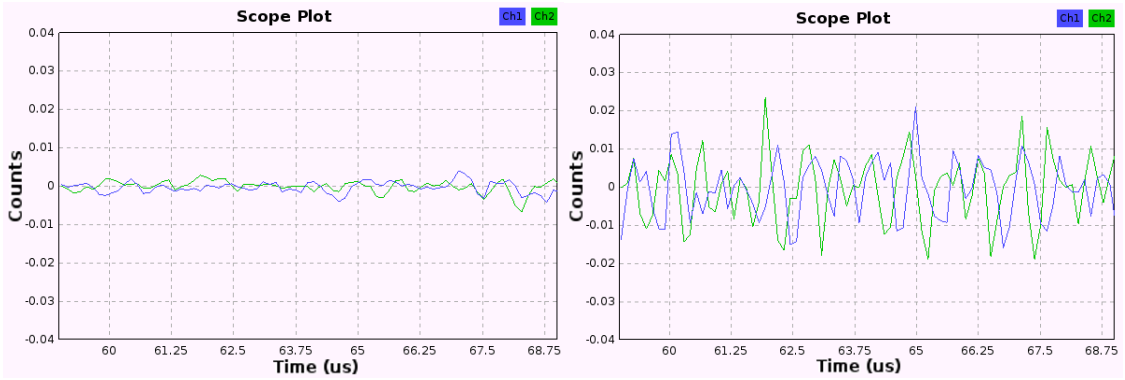
The software radio we used when designing this interface is the USRP1 [87], from Ettus Research. We then used a USB2 connection and the libUHD [88] to control the radio and access the RX and TX sample streams.

4.3.1 Time Domain vs Frequency Domain

Traditional radios can only decode one transmission at a time, they tune to a central frequency, set filters and the sampling rate according to the fastest modulation that will be used and wait for the power received by the antenna to be higher than a threshold that depends on the thermal noise of the radio. Such detection algorithm is said to process in the time domain because it operates directly on the samples. As an example, Figure 4.3a shows the power received by the radio when no transmission is happening in the sensed spectrum while Figure 4.3b shows the received power when one or multiple transmissions are happening.

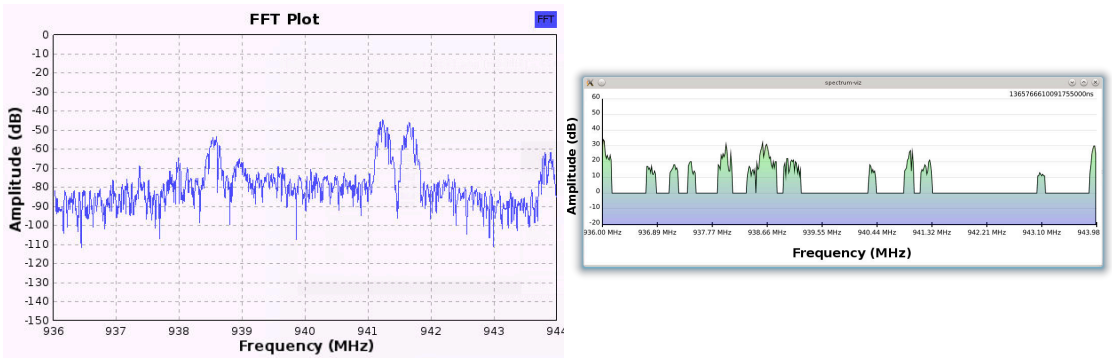
With software radios, it is possible to decode multiple transmissions happening at the same time provided we frequentially isolate them. Detecting transmission in the frequency domain requires to transform the samples we receive from the radio to the frequency domain. This is done using a Fast Fourier Transform (FFTs). The output of an FFTs is the received power and phase at various frequencies. The frequency resolution

is linear with the number of samples used to compute the FFTs. In our case, we used FFTss on 1024 samples. An example of the output of an FFTs is shown in Figure 4.3c where at least 5 transmissions are clearly visible. Other transmissions could be visible with the appropriate filtering to get rid of the thermal noise.



(a) Time domain, no transmission

(b) Time domain, transmission



(c) Freq. domain, no filtering

(d) Freq. domain, with filtering

Figure 4.3: Detecting transmissions in the time domain (4.3a, 4.3b) and the frequency domain (4.3c). Sub-figure 4.3d illustrates the need for filtering the data from the frequency domain before being usable.

This algorithm is in the energy-detection class of transmission detection. It has the advantage of being agnostic to the modulation scheme used by the transmitters but it works poorly in low-SNR ratio and DSSS scenarios. Other detection techniques could be applied on top of our system, such as the cyclostationary detection technique, to help improve the detection of low-SNR signals at the expense of CPU usage. Since our proposition already consumes a full core of an i7 860 to perform time domain to frequency domain conversion of 10 million samples per second, we did not investigate these techniques.

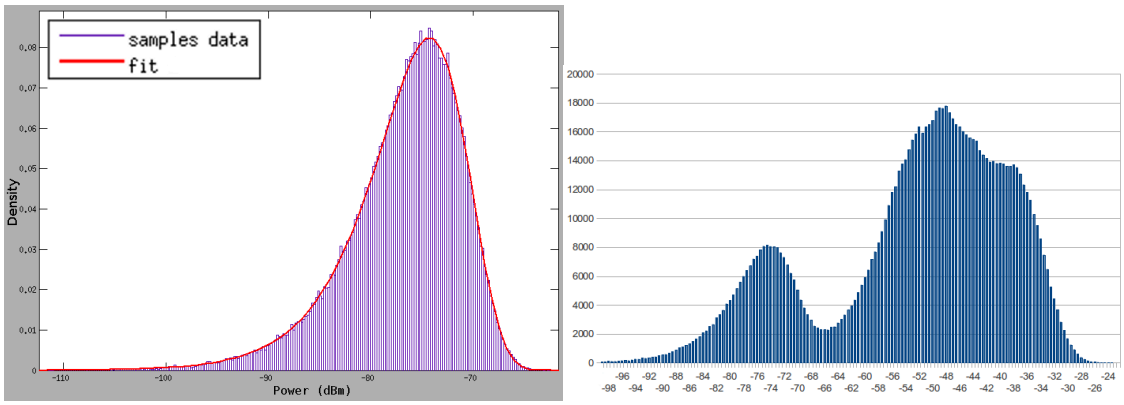
4.3.2 Filtering in the frequency domain

Filtering the signal in the frequency domain can simply be done by averaging it because the noise will cancel itself and the SNR ratio will go up. However, averaging works best

as long as the transmissions received are longer than the number of samples used in the filter. This is problematic in frequency bands where some transceivers use FHSS. We thus looked for another solution.

Although thermal noise is often considered additive, white and Gaussian (AWGN), we experimentally found out that when using Ettus Research's USRPs Software Radios, the noise distribution was not Gaussian but rather mapped perfectly to an extreme value distribution, given in Eq. 4.1, with $\mu = 1.67$ and $\sigma = 4.5578$. The only variable element was the average of the distribution which varies depending on the central frequency and the offset to it in the spectrum window observed by the software radio. In Figure 4.4a, we can see how well the model fits the experimental data.

$$f(x|\mu,\sigma) = \frac{1}{\sigma} e^{\frac{x-\mu}{\sigma}} e^{-e^{\frac{x-\mu}{\sigma}}} \tag{4.1}$$



(a) Noise-only **(b)** Noise + frequent transmissions

Figure 4.4: Histogram of the received power at one frequency

Based on Eq. 4.1, we propose to set a threshold right at the maximum power taken by the noise (14 dB after the tip of the curve). If the received power exceeds this threshold, this means that additional power has been received by the antenna and thus that a transmission is happening. In Figure 4.4b we can find the histogram of the received power at a given frequency in the presence of an intermittent transmission.

The end of a transmission is harder to detect because the noise may "pull the power down" under the threshold unless the received power is above the noise's "variance". We thus propose to use the number of samples under the threshold as a way to detect the end of a transmission. Eq. 4.2 represents the probability for the power to be under x dB above the noise's average. Using this equation, we can calculate n, the number of samples under the threshold we need to wait in order to have a confidence probability of $p_{confidence}$ that the transmission has ended (Eq. 4.3).

$$p(x|\mu,\sigma) = 1 - e^{-e^{\frac{x-\mu}{\sigma}}} \tag{4.2}$$

$$p(x|\mu,\sigma)^n > p_{confidence} \Leftrightarrow n = \frac{ln(1-p_{confidence})}{ln(p(x|\mu,\sigma))} \tag{4.3}$$

4.3.3 Decoding and collecting statistics

We are now able to detect the beginning and the end of transmissions in the frequency domain. We propose to use this information to fill a table called the Radio Events Table that contains PHY-layer metadata about transmissions such as the start/end time, frequency band and the received power at the antenna. Using this information, a decoder reads the samples that contain the transmission, apply a pass-band filter to only let this transmission pass, roughly correct for the massive frequency offset and feed this to the usual demodulators used in software-defined radios. The demodulated frame can then be used to add additional information to the Radio Event Table such as the source node's ID and its nature (primary or secondary user). The ID could be the MAC address of the source node when applicable or an ID generated by the first CR that detected the transmissions for broadcasting emitters such as FM, TV or radar transmitters. Figure 4.5 presents an overview of the sensing/receiving process.

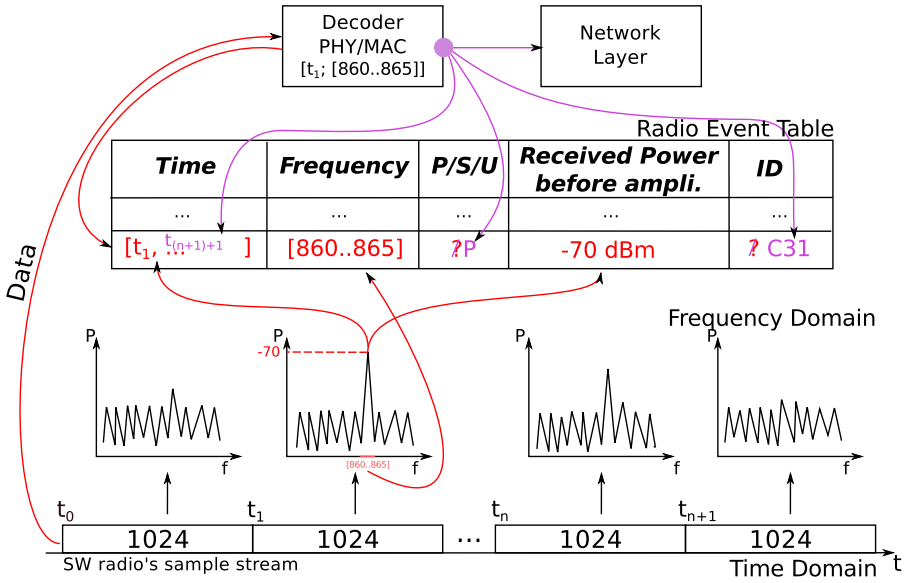


Figure 4.5: Overview of the sensing process - Filling the Radio Event Table from the software radio's sample stream

These experimentations have been used as part of the LICoRNe project, funded by the French National Research Agency (ANR), in order to create a video stream between two CRs that hops onto another channel when a primary user (PU) appears in the current channel. Additional experimentation on transmission detection, demodulation and transmissions in the time domain have also been used by 8 students for building a smart box capable of detecting its surrounding wireless sensors and automatically

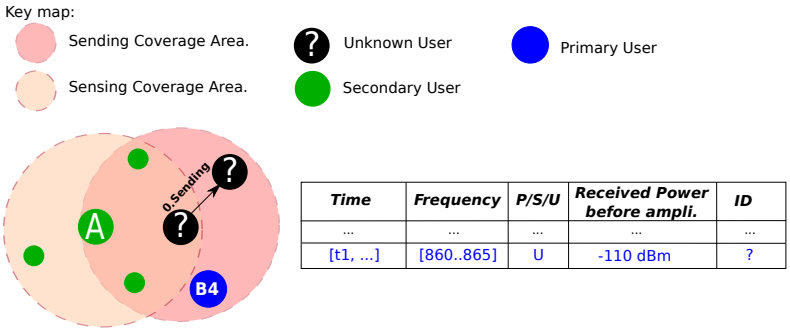
finding their modulations. Our experimentations are available publicly [89] and some are partially explained in Appendix F.

4.3.4 Collaborative sensing

It is critical for a CR to be able to identify transmissions happening in its vicinity to avoid accidentally interfering with a primary user.

If a CR is unable to identify the origin of a transmission, either because it was already happening when the CR arrived in the band or because it the SNR is insufficient, the CR should request its neighbours to provide information about this transmission.

Sensing

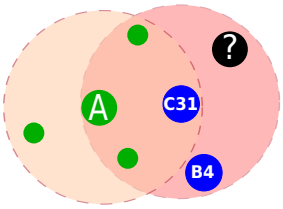


Time	Frequency	P/S/U	Received Power before ampli.	ID
...	...	...	...	...
[t1, ...]	[860..865]	U	-110 dBm	?

Figure 4.6: Collaborative Sensing: CR A performs sensing and detects a new transmission

In Figure 4.6, CR A is performing sensing in the RF spectrum when it detects a new transmission. The CR creates an entry in its Radio Event Table containing the time at which the transmission began, the frequency band it uses and the power received at the antenna. Since it does not know yet if the transmission originates from a primary or a secondary user (SU), it tags the transmission as coming from an unknown source.

Demodulating and decoding the frame



Time	Frequency	P/S/U	Received Power before ampli.	ID
...	...	...	...	...
[t1,+14ms]	[860..865]	P	-110 dBm	C31
...	...	...	...	...

Figure 4.7: Collaborative Sensing: CR A decodes the unknown frames and fills the RET

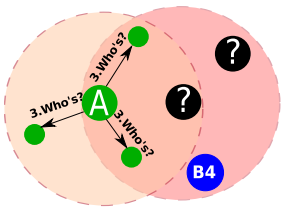
In Figure 4.7, CR A attempts to find the PHY parameters of the transmission and then try to demodulate it. If it succeeds, it forwards the frame to the MAC Layer that in turns tries to decode the frame. If the CR A has successfully decoded the frame, it tags the transmission as originating from a primary ("P") or a secondary ("S") user and adds the source id of the emitting node to the corresponding entry of the Radio Events Table. When the transmission ends, CR A completes the Radio Event Table entry by adding the time at which the transmission ended.

Contrarily to what we found in the state of the art [90][91], we do not believe that a primary user should be identified by PHY-layer parameters because it would force secondary users to use other modulations for not good reason. Also, this technique has been found to be unsecure as secondary users could impersonate primary users. Multiple techniques have been proposed [92] but none is effective in a mobility scenario or if no prior knowledge of where primary users are located.

Instead, we propose that a CR should be able to recognise if the frame bas been sent by a primary or a secondary user based on its MAC address. We propose allocating new entries in the 24-bit-long Organisationally Unique Identifier (OUI) for cognitive radios. A secondary user could thus be easily detected by checking the first 3 bytes of its MAC address. If a CR is a primary user in one band and a secondary user in another one, it should use two different MAC addresses.

Our proposition is not secure as secondary users can use any MAC address they want. However, we think that the only reliable solution to be able to distinguish a primary from a secondary user would be for the state to grant certificates that the primary user would broadcast from time to time or when it detects the activity of a secondary user in its band. The certificate would contain the owner, the frequency band and the type of usage allowed to this user. It would then be signed by a state-mandated authority as the state is currently in charge of the spectrum allocation. A cognitive radio would only need to be pre-loaded with the public keys of all the state telecommunication agencies to be able to reliably distinguish PUs from SUs.

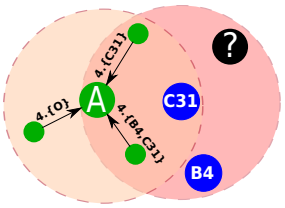
Collaboration



Time	Frequency	P/S/U	Received Power before ampli.	ID
...	...	...	...	...
[t1,+14ms]	[860..865]	U	-110 dBm	?
...	...	...	...	...

Figure 4.8: Collaborative Sensing: CR A did not manage to decode the frame and queries its neighbouring CRs.

In Figure 4.8, CR A did not manage to demodulate or decode the frame. It thus broadcasts to its neighbours the information about one or more transmissions it did not manage to decode. The transmissions are identified by a time range and the frequency band it was seen in. However, since the clocks of the radios are unlikely to be synchronised, we proposed sending the start and finish time relatively to the time of emission of the request. For instance, CR A would ask its neighbours what was the communication that started 500 ms and finished 486 ms ago in the [860, 865] MHz band. If a neighbour CR also wondered about the origin of this transmission and received CR A's request, it should simply wait for the surrounding CRs answer.



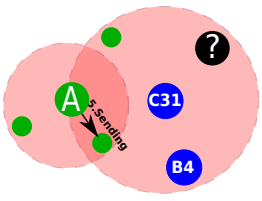
Time	Frequency	P/S/U	Received Power before ampli.	ID
...	...	...	...	...
[t1,+14ms]	[860..865]	P	-110 dBm	C31
...	...	...	...	...

Figure 4.9: Collaborative Sensing: CR A receives 3 answers to its query and updates the Radio Event Table.

In Figure 4.9, the surrounding CRs answer back to CR A's request. One CR tells it was a primary user with the ID C31. Another one says it could either be from primary user B4 or C31 as both have the same attenuation. The last surrounding CR does not know. Since there is no ambiguity, CR A thus tags the transmission as coming from the primary user C31 in its Radio Event Table.

If a neighbouring CR is still waiting for the answer because it was not able to hear it, it should send the request again. This mechanism reduces the number of requests sent.

Filling the maximum emitting power and the neighbours tables



Neighbouring PUs table

ID	Frequency Range	Max Power Rx	Max Power Tx
...	...	...	...
C31	[850..860]	-70 dBm	10 dBm
...	...	...	...

Maximum emitting power over frequency table

F_{ID}	Frequency Range	Maximal Power	ID
F_0	[850..860]	Unlimited	N/A
F_1	[860..865]	10 dBm	C31
F_2	[865..915]	Unlimited	N/A

Figure 4.10: Collaborative Sensing: CR A now updates its maximum emitting power and neighbours table to remember what the maximum emitting power that should be used in every band.

In Figure 4.9, CR A discovered the presence of a primary user in the frequency band [860, 865] MHz. In Figure 4.10, it stores in its neighbouring primary users (PU) table

that PU C31 operates in [860, 865] MHz, that the maximum received power from this node was -70 dBm and that the maximum power CR A can use without interfering with this primary node. It also adds in its maximum emitting power table that the emission power in the frequency band [860, 865] MHz should be limited to 10 dBm and that the reason for this limitation is PU C31. If the PU was not to be heard from for some time, this limitation can easily be deleted as it is linked to C31.

Conclusion

As a summary, upon sensing a transmission, a CR should follow the following flowgraph, found in Figure 4.11.

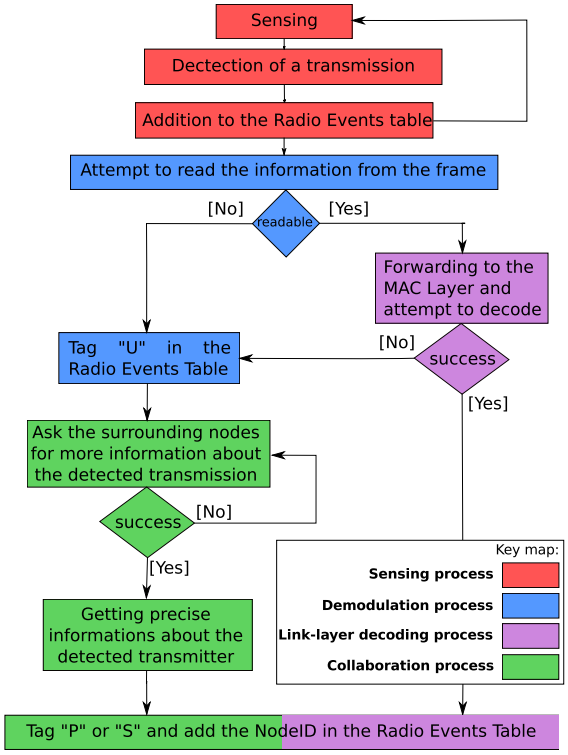


Figure 4.11: Flowgraph of the collaborative sensing process among cognitive radios.

4.3.5 Software & Hardware architectures for software radios

The experience and experiments used to study both the hardware and software architectures necessary for software radios has been carried out on multiple USRP from Ettus Research and Nuand's BladeRF [83].

Receiving data from the Software Radio in the userspace

As seen previously in the state of the art about software radios, if a software radio wants to listen to a spectrum window of 10MHz, it has to receive 10 million samples

per second. In software radios, a sample is actually a complex sample which means one sample is composed of two 16-bits values.

Software radios usually send the sample stream through a USB or Ethernet interface. When an application wants to access the stream, it asks the operating system (kernel) to copy a chunk of the stream into the provided buffer. Since the USB and Ethernet interfaces are multiplexing multiple streams for different applications, the kernel has to de-multiplex data first before being able to synchronously write the samples to a user-defined buffer. This copy uses CPU power while the constant system calls made by the application to get the samples generate many context switches which is hurting performance and latency. Since all these operations depend on caches and the general load of the machine, this creates jitter that is very harmful when emitting samples because latency needs to be increased in order to avoid buffer underruns which would result in a corrupted signal. Characterisation of this jitter has been carried out by [17] and can be found in Table 4.2. This test was done on a USRP1 [87] listening to a 8MHz spectrum window, using a USB2 connection.

Measurement	Avg	SDev	Min	Max
User →Kernel (μs)	24	10	22	213
Kernel →User (μs)	27	89	13	7000
4096 Kernel ↔ FPGA (μs)	291	62	204	360
512 Kernel ↔ FPGA (μs)	148	35	90	193
GNU Radio ↔ FPGA (μs)	612	789	289	9000

Table 4.2: Kernel-level delay measurements on a USRP1 [17]

The fastest software radio found today is the USRP X300 [93], which can output up to 200 MS/s in Full Duplex. This represents 12.8GBit/s worth of data when transmitted as 16-bit integers. In order to cope with all this data, the kernel should be taken out of the equation by allowing the software radio to write directly in the application's memory, using Direct Memory Access (DMA). Since the USB and Ethernet interfaces are not meant to achieve this, Ettus Research introduced for this model a PCI-express interface. With this interface, they claim to achieve a 10μs latency between the ADC and the application responsible for processing the data.

Temporarily storing samples for immediate processing

In Figure 4.5, we saw that the samples coming from the software radio are first transformed to the frequency domain before performing transmission detection. When a transmission has been detected and its frequency band identified, the demodulator (which operates in the time domain) needs to access the samples ranging from right before the beginning of the transmission to a few samples after the end of the transmission.

First of all, this means that samples must be stored in RAM for a few hundreds of milliseconds to give enough time for a demodulator to be instantiated. Secondly, it is

necessary to be able to efficiently locate the samples that were used to compute an FFTs. Thirdly, the data structure may contain samples taken with different parameters for the software radio (frequency band, bandwidth, gain, ...). Lastly, since we will store tens of millions of samples per second, the data structure should be highly efficient.

We designed a new template-based data structure in C++ that never blocks and also allows to annotate elements with meta-data. It is based on a ring buffer.

The "never block" property is important for this application as we want to cache the latest N elements of a stream regardless of which thread is going to read them. The point is that the producer (the software radio) should never be blocked and should keep on adding samples. The problem with that property is that consumers cannot lock the content they are reading. This means the content can be overridden by the producer at any time while a consumer is reading it. A consumer should thus attempt to copy the samples it wants to access before checking again that the first sample copied is still available. If this is not the case, then the samples may have been overwritten during the copy which makes it invalid.

The "annotate" property is important as some meta-data (the software radio's PHY parameters) need to be attached at some positions in the sample stream. This increases the complexity of the ring buffer as both the meta data and the data should be added at the same time. For this reason, we introduced a staging area to be able to expose both the data and meta-data at the same time.

Samples are indexed using a 64 bits unsigned integer. We consider that an application will not run long-enough to overflow this integer as it would take 97 years to do so with the fastest software radio existing today (200 MSps [93]). This integer can be incremented and/or read atomically using C++'11's <atomic> library. Atomic operations cannot be interrupted which makes them thread-safe. They are also way more efficient than traditional locks.

We proposed a ring buffer that is thus thread-safe and lock-less. It works when there are no more than one producer thread and as many consumer threads as needed. This ring buffer also allows attaching meta data to samples to be able to know the parameters of the software radio for each sample in the ring buffer. The code of this data structure along with usage examples is available in Appendix G.

Power saving opportunities

To save power, the software radio could use its embedded processor to analyse the sample stream to detect transmissions through energy detection. When the radio would detect a transmission, it would send some samples from before the transmission and would stop sending them a few samples after the end of the detected transmission. This would save a lot of processing power on the main CPU when no transmission is happening which would save a lot of power because the CPU is not as efficient as the software radio's processor. We experimented with this architecture in the *hachoir_UHD* project that allowed

us to demodulate different types of modulation and PHY parameters automatically, as shown in Appendix F.

To further increase the power savings, the software radio itself could detect transmissions in the frequency domain, bring them in the baseband and decimate them before sending them to the CPU. To do so, the software radio should implement in its FPGA the Fast Fourier Transform in order to convert the signal from the time to the frequency domain. The embedded processor could then detect transmissions in the frequency domain by following our filtering contribution. Finally, it would convert the signal to baseband before sending the transmission to the main computer. In this case, the radio would only output the minimum of information necessary for the CPU to decode all the transmissions.

In our implementation of the proposition found in 4.5, we found out that a vast majority of the computational time was spent on transforming the time domain to the frequency domain. Having the radio offload this operation would thus offload the most expensive operation.

We however did not have time to implement either power savings techniques to evaluate how power would be saved as the best solution would require about a year worth of implementation work to be able to have a working implementation.

4.4 PHY-layer signalling protocol

CRNs require information about the network topology, available channels, channel fading and on which frequency bands any surrounding CR is available in order to provide efficient unicast or broadcast schemes. Unfortunately, the schemes proposed in the state of the art try to reduce the requirements to provide connectivity at the expense of spectrum utilisation, delay and throughput. Moreover, blind rendez-vous, unicast MAC protocols and broadcast protocols are not compatible as they all require to be responsible for selecting the frequency band the radio will listen and send on.

The goal of our PHY-layer signalling protocol is to allow CRs to find each others (rendez-vous) and to know when and how to contact each others in the future, even if CRs are hopping from frequency bands to frequency bands. To do so, CRs should have at least part of their frequency hopping pattern predicable. Since CRs can be available on a limited amount of bands without drastically increasing the maximum latency to contact them, advertising the actual frequency hopping pattern explicitly is possible without creating a very large beacon frame. Once the hopping pattern and the current position in the pattern of a CR is sent in a frame, CRs that received it can predict when and on which band they should send transmissions to this CR. To make this time synchronisation more accurate, the emitting and receiving CRs should compensate for the delay introduced by the kernel and the radio in the propagation of the emitted or received samples. In a situation where no under-runs happen and the radio is emitting/receiving a constant stream of samples, the TX delay is entirely predicable due to the fixed sampling

rate. However, the average RX delay needs to be specified. A beacon example is given below in its ASCII form.

```
<beacon_frame>{ node_id=23, tx_pwr=10dBm,
[
  { {band1}, len=0.4, period_offset=0.0 },
  { {band2}, len=0.4, period_offset=0.0 },
  { {band3}, len=0.3, period_offset=0.5 },
],
period=1000ms, cur_period_offset=0.126 }
```

In this example, the beacon is sent by node_id 23 which is available on 3 bands. At the beginning of the hopping cycle, node 23 is available on both *band*1 and *band*2 for 400ms ($0.4 * 1000ms$). At 500ms ($0.5 * 1000ms$), node 23 will be available on *band*3 for 300ms. Node 23 is currently 126ms ($0.126 * 1000ms$) in its hopping pattern cycle which means the radio will stop being available on bands 1 and 2 in 274ms (400ms - 126ms). One slot of 100ms (400ms to 500ms) and one of 200ms (800ms to 1000ms) are not currently allocated in the beacon. This allows the radio to either put itself to sleep or perform sensing anywhere in the tunable frequency spectrum. This beacon has been sent at 10 dBm, this allows the receiver to compare it with the received power to evaluate the channel fading so as CRs can lower their transmission power if they assume the fading is roughly symmetric. This enables power savings for the emitting node while reducing spectrum usage.

Using this beaconing mechanism, it is possible for radios to define sleep periods which would allow them to save power. Because this sleep period would be advertised in the node's beacon, surrounding radios will not have to constantly re-send packets to the sleeping radio as they would know about its sleep pattern. The periodical emission of beacon allows the radios to synchronise each others to compensate for their internal clock drift.

4.4.1 Bootstrapping

We propose that when booting up, CRs should first sense the tunable spectrum in order to find frequency bands that are usable without disrupting primary users. During this sequence, the transceiver is passive and should linearly scan the tunable spectrum. If beacons are received from cognitive users, they should be stored in a neighbours CR list.

4.4.2 Advertising

Once some frequency bands have been found to be available, we propose a CR should send its beacon when:

1. It becomes available on a frequency band;
2. No beacon has been sent on this band for some time;
3. Another CR requests the CR to resend it;
4. Another CR requests every CR to resend it.

Rules 1) and 2) generate a background traffic that allows CRs to keep their hopping patterns synchronised without needing an external time-synchronisation mechanism. Rule 3) allows CRs to re-synchronise with one another and to check if a CR has left from the band. Rule 4) enables a CR to request all his surrounding CRs to get the list of currently available CRs on the current band to ease the discovery process.

4.4.3 Scanning for other cognitive radios

Our PHY-layer proposition is meant to make scanning as simple and un-intrusive as possible for CRs. Information about surrounding CRs is gathered simply by receiving their beacons. This operation can be done while being available on the advertised frequency bands or while sensing the spectrum. The information about surrounding CRs should be stored in a database containing the latest-received beacons of every CR.

Since the characteristics of the hopping pattern of surrounding CRs is not known, the best approach to discover other CRs is to randomly hop in the frequency spectrum. With our proposition, CRs can thus rendez-vous with unknown CRs at no cost and can perform sensing if and when wanted.

4.4.4 Updating the hopping pattern

If two CRs need to communicate often or have a low-latency requirement, they need to adjust their hopping pattern so as to maximise the time they spend being reachable from one another. Changing the hopping pattern of a CR can be challenging to do locally without breaking the connectivity of the CRN.

It is safe to update the frequency hopping period or drop a frequency band as long as every neighbouring CR will still be available. This can be checked by finding at least one periodical overlap between the CR's new beacon and every other beacon from CRs it can already communicate with. It is however unsafe to update the beacon and to rely on a neighbour CR to maintain the connectivity because they may change their hopping pattern in the future.

Since our PHY-layer signalling protocol provided the discovery and the loose time synchronisation necessary for communicating with surrounding CRs and since the connectivity problem cannot be addressed entirely locally, we believe that optimising the hopping pattern is outside the scope of our proposed PHY-layer signalling protocol and that it should be addressed at the network layer, with a cross-layer approach. An example of hopping pattern optimisation is however available in Chapter 7.

Once a decision has been made to change the hopping pattern, the CR needs to advertise its new beacon. A CR can change its hopping pattern abruptly but it may result in CRs failing to contact it depending on the change that was made. To avoid this situation, it may be necessary for the CR to advertise the change before it happens. If this is the case, we propose that a CR should send its beacon with the *UDPDATE* flag set and also containing the future cycle (bands + period). This beacon should be sent for an entire

cycle before using the new hopping sequence. An example beacon is given below in its ASCII form.

```
<beacon_frame>{ node_id=23, tx_pwr=10dBm, flags=UPDATE
[
  { {band1}, len=0.4, period_offset=0.0 },
  { {band2}, len=0.4, period_offset=0.0 },
  { {band3}, len=0.3, period_offset=0.5 },
],
period=1000ms, cur_period_offset=0.6,
[
  { {band5}, len=0.2, period_offset=0.0 },
  { {band3}, len=0.5, period_offset=0.5 },
],
period=2000ms }
```

In this beacon, node 23 is advertising a change in its hopping cycle. After the end of the current cycle, which will happen in 400 ms, the period will be increased from 1 to 2 seconds. band5 will then be available for 400 ms before the radio is put to sleep for 600 ms. band3 will then be available for one second before returning on band5 for 400 ms again. A switch has happened. The beacon sent on this new hopping pattern is the following:

```
<beacon_frame>{ node_id=23, tx_pwr=10dBm
[
  { {band5}, len=0.2, period_offset=0.0 },
  { {band3}, len=0.5, period_offset=0.5 },
],
period=2000ms, cur_period_offset=0.012 }
```

4.4.5 Keeping nodes' hopping pattern synchronised

With time, the CRs' clock drifts and their hopping pattern become less and less synchronised. The clock difference between two CRs can be evaluated by one CR when receiving a beacon from the other CR by comparing the expected *cur_period_offset* with the value found in the beacon.

If the difference is small and relatively constant, it means that both CRs are counting at roughly the same average frequency. It is thus possible to simply adjust our current phase to the average phase of surrounding CRs in order to maximise its availability. To do so, a CR computes the difference between the start of the cycle of each surrounding CR with its own. An average phase error can thus be computed and the CR should then adjust its own period offset to match this average phase in order to maximise its availability. If every CR does that periodically, a tight time synchronisation between all the surrounding CRs of a network is possible throughout its life.

If the difference is increasing steadily, it indicates that both radios do not count time at the same speed. This would indicate that the crystal of one or both CRs is not oscillating at the right frequency. It would be possible to correct for this frequency difference locally

if a reliable clock source was available. This is however not the case. Another solution is to evaluate the clock difference between all the surrounding CRs, average it and consider this average difference to be the reliable clock source necessary to compute the frequency offset calibration. This will minimise the average distance with other CRs and will result in a tighter synchronisation.

Not only does the proposed solution works locally but it also works globally in a ad-hoc wireless network if all the cognitive CRs are following the same program. Indeed, if each CR lowers the average phase and frequency difference between itself and its surrounding CRs, then all the CRs will tend towards the same average frequency and will keep the same phase.

The convergence delay of this proposition depends on the network size and how well it is connected. Indeed, the less a network is connected, the longer it will take for all the radios to converge as information takes time to propagate. On the contrary, if all the CR of the network are connected to the every CR, each CR will only need to adjust its phase and frequency once. A proper evaluation of convergence delay in common scenarios would however be interesting, especially when taking into account the instability of clocks.

4.4.6 Evaluation

We evaluate the proposed mechanism on both software and hardware radios. To speed up the experiments and evaluate the influence of every parameter, we wrote a non-real-time simulator that allows repeating our experiments millions of time to get an accurate average and maximum discovery time.

The source code of our C++ simulator is available publicly [89].

Software radios

To evaluate our proposition on software radios, we use two simulated CRs with a tunable band ranging from 300MHz to 3GHz and a spectrum window of 25MHz. These characteristics are similar to Nuand's bladeRF [83] software radio.

The first node has a hopping pattern period of 1s and advertises two bands that are available respectively in [0.0, 0.4] and [0.5, 0.9]. The two bands are selected randomly at the beginning of the experiment. The node will send a beacon 5ms after switching to a band and at a user-defined period after that. Beacons are sent at 1 MBit/s, using a simulated bandwidth of 500kHz and 296μs to send it, according to the binary structure of the beacon found in Figure 4.12.

1	6	2	1	1	1	1	4	4	1	1	4
Frame Type	Src Node	Period	Current Offset	Flags	TX PWR	Bands Count	Freq Start	Freq Stop	Duration	Period Offset	Checksum
Beacon		ms	0 - 0.0 255 - 1.0		signed, dBm		kHz	kHz	0 - 0.0 255 - 1.0	0 - 0.0 255 - 1.0	

Figure 4.12: Format of the beacon frame

The second node performs only sensing. It randomly hops from one band to another with a user-defined hopping period. The experiment finishes when the sensing node is able to hear the entirety of a beacon sent by the first node.

Figure 4.13 shows the average time (over 10000 instances) that the sensing node takes to receive a beacon from the other node depending on both the beaconing period and the sensing hopping period.

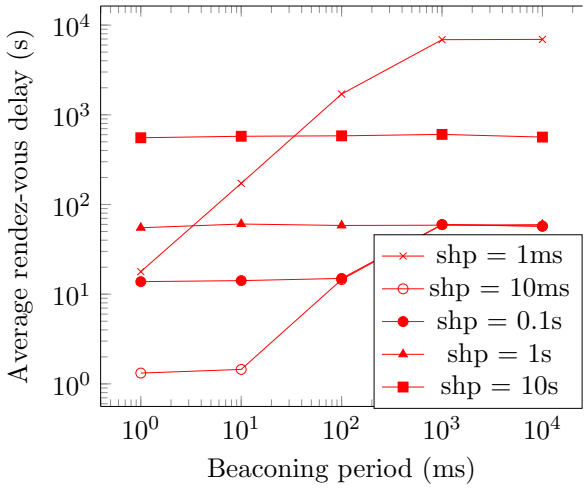


Figure 4.13: Influence of the beaconing and sensing-hopping period on the average rendez-vous delay.

In "shp = 1ms", the sensing CR has a high probability of not hearing the complete beacon because the sensing period is short compared to the emission time (280μs) of the beacon. Multiplying by 10 the beaconing period increases the average rendez-vous 10 folds until reaching one second.

After this point, the only beacon sent by the CR is the one sent when hopping to a new band because the hopping cycle takes 1 second and no increase in the delay can be observed. The same behaviour can be observed with the other sensing hopping periods with an added initial plateau due to the sensing hopping period being higher or equal to the beaconing period, ensuring reception of the beacon.

The sensing period should thus be set around 10ms and the beaconing period set as low as possible (based on the utilisation of the band) to achieve the lowest discovery delay possible.

To be able to decode a beacon, the sensing radio must get the entire transmission both in the time and the frequency domain. If a beacon is sent at a central frequency f (in MHz) and uses b Mhz of bandwidth to be sent, then the sensing radio with a bandwidth sbw and a central frequency $f2$ can receive it if the inequality 4.4 is true.

Additionally, the sensing radio also has to be available on this frequency band during the whole transmission.

$$f - sbw + \frac{b}{2} \leq f2 \leq f + sbw - \frac{b}{2} \tag{4.4}$$

This means that when the sensing radio uses a 25 MHz bandwidth and only one beacon is sent at a time, the central frequency of both radios cannot be separated by more than 12.25 MHz in order to receive the transmission.

A CR may send more than one beacon at a time to increase the likeliness of being found by surrounding CRs. When the beacon count is greater than 1, the beacons are sent at both the lowest and highest frequency possible to maximise the probability of being found. Beacons are then spread evenly in the band, as can be seen in the spectral representation found in Figure 4.14. The blue colour represents the tunable band of a CR while the yellow indicates its RF window. The red indicates the beacons it sends while the numbers under them give an idea how our placement algorithm works when increasing the number of beacons.

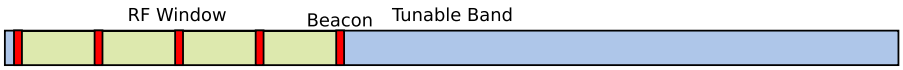


Figure 4.14: Spectral representation of where a cognitive radio sends its beacons when having *beacon_count* = 5.

In Figure 4.15, we evaluated the impact of the number of beacons sent by the first node along with the sensing radio's bandwidth on the average rendez-vous time. This experiment was carried out with a beaconing hopping period (bp) of 10ms and with a sensing hopping period (shp) of 10ms.

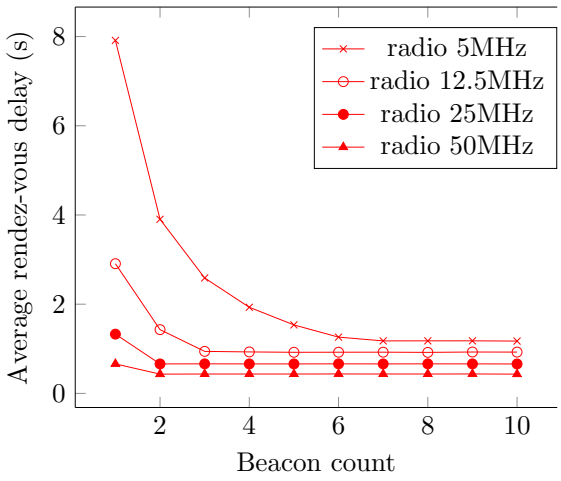


Figure 4.15: Influence of the number of beacon sent and the sensing radio's bandwidth on the average rendez-vous delay. shp = 10ms, bp = 10ms

When both radios have the same bandwidth (25 MHz), increasing the beacon count from one to two halves the average rendez-vous time. This is because the sensing radio cannot receive both beacons at the same time which means the probability of hearing one beacon was multiplied by two.

When the beacon count is increased to 3, the third beacon's central frequency is halfway between the first and last beacon (12.25 MHz). This means that if the sensing radio is able to hear this third beacon, it is able to hear either one of the other beacon because they are located at less than half the bandwidth of the sensing radio.

It is thus not necessary to increase the beacon count when the distance in MHz between them is lower than the sensing radio's bandwidth. In Figure 4.16, we evaluated the impact of the sensing radio's bandwidth on the rendez-vous delay. The experiment has been repeated one million times per data point and we plotted the minimum, average and maximum rendez-vous time.

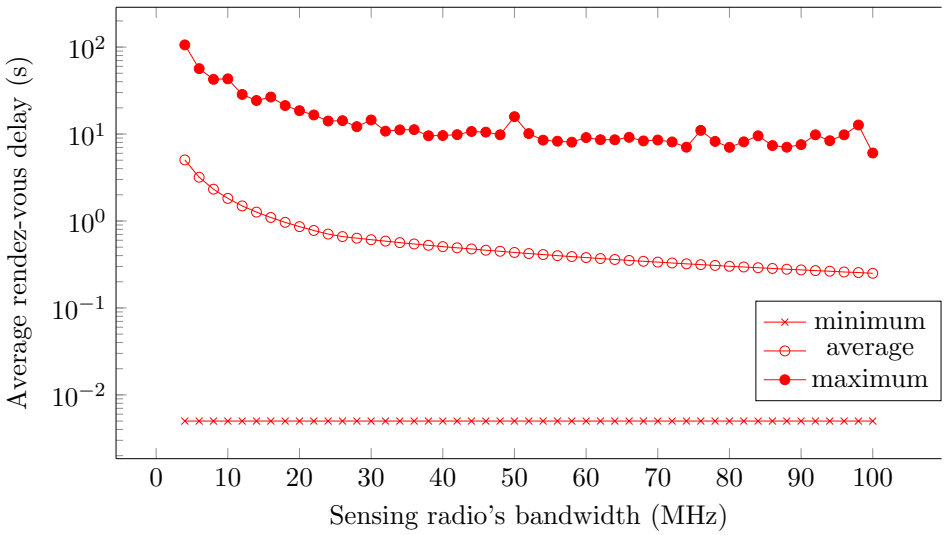


Figure 4.16: Variance of the rendez-vous time with shp = 10ms, bp = 10ms, bc = 2, 1 million iterations

We found out that the minimum rendez-vous time was always 5 ms, which is the delay a CR should wait after switching to a band before emitting anything. The average delay decreased from 5 seconds to 250 ms when increasing the sensing radio's bandwidth from 4 to 100 MHz. Likewise, the maximum delay decreased from 106 seconds to 6 seconds but had much more variance due to the limited amount of times the experience was run (1 million times). It however follows the same trend as the average delay.

Hardware radios

Our proposed beaconing mechanism can also be used on hardware radios even if it requires the nodes to agree on the modulation used for all the communications.

We decided to evaluate it on a simulated nRF24L01 [94], a highly configurable cheap and power-efficient ubiquitous radio operating in the 2.4GHz band. When configured for using 1 MHz channels, it can operate on 115 channels ranging from 2.4 to 2.515 GHz.

Our simulator used for the software radios has been extended to better suit the characteristics of the nRF24L01 as the central frequency had to be an integer in MHz. We then used the same scenario as we did for evaluating software radios. This scenario only involves two cognitive radio nodes as it is the worst case scenario. Indeed, adding more nodes would result in more beacons sent per second which would increase the likeliness of being discovered no matter where the CRs are emitting. The first node has a hopping pattern period of 1s and advertises two bands that are available respectively in [0.0, 0.4] and [0.5, 0.9]. The two bands are selected randomly at the beginning of the experiment. The node will send a beacon 5ms after switching to a band and at a user-defined period after that. The second node performs only sensing. It randomly hops from one channel to another with a user-defined hopping period. The experiment finishes when the sensing node is able to hear the entirety of a beacon sent by the first node. The result of this experiment is available in Figure 4.17.

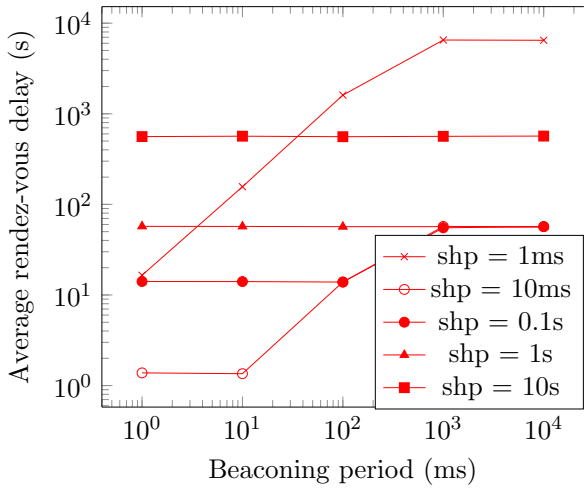


Figure 4.17: Influence of the beaconing and sensing-hopping period on the average rendez-vous delay on a hardware radio.

The results of this experiment are almost exactly similar to the ones we got simulating a software radio. The only difference is that the hardware radio got an average rendez-vous time sightly lower than the one found with software radios. This is because the nRF24L01 only has 115 central frequency possible while the software radio had less "channels" (108) but the central frequency could be set anywhere in its band.

To evaluate the rendez-vous delay's variance, we plotted in Figure 4.18 the minimum, average and maximum rendez-vous delay depending on the beaconing period. We can

see that the maximum delay is about ten times higher than the average delay whatever the beaconing period.

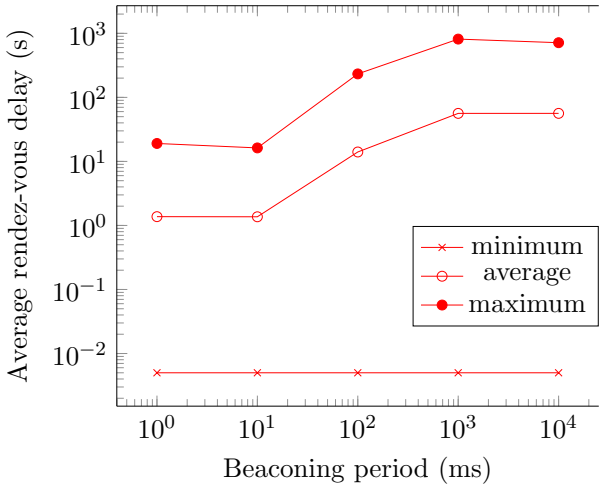


Figure 4.18: Variance of the rendez-vous time with shp = 10ms, 100000 iterations

As hardware radios can only send one packet at a time, we cannot send more than one beacon at a time. Likewise, hardware radios cannot demodulate transmissions whose central frequency is different from the radio's. The evaluation of the impact of the sensing radio's bandwidth on the average rendez-vous time is thus not applicable to hardware radios.

4.4.7 Conclusion

We evaluated the performance of our beaconing mechanism on both hardware and software radios. In both cases, a radio was able to find the other node in less than 2 seconds in the hardware radio case and less than a second for the software radio case. However, software radio's abilities to emit and receive more than one communication at a time can dramatically reduce the rendez-vous time while also allowing easier updates in the PHY-layer specifications for the beacon.

A hybrid network composed of both hardware and software cognitive radios is thus possible even if hardware radios are often limited to a narrower RF band.

We compared our solution to the enhanced jump-stay algorithm [3] which has a one-time rendez-vous cost like our proposition. We set the timeslot size for this algorithm to 20 ms, as recommended in an earlier version of the paper. We found out that for the same use case presented earlier, the average rendez-vous time would be 3.3 seconds while the maximum rendez-vous time would be 8.72 seconds.

We also compared our solution to [4] that provides a per-frame rendez-vous mechanism. We set the sensing time per channel to 1 ms as it takes some time for the radio to change

frequency and a few hundred of microseconds to perform energy sensing. We found out that it would take an average of 54 ms to rendez-vous while the maximum rendez-vous time would be 108 ms.

In Figure 4.19, we show the differences in rendez-vous of our proposition when using hardware and software radios compared to [4] and [3]. Our proposition is respectively 2.4 and 5 times faster in average than the other one-time synchronisation cost algorithm [3] for hardware software and software radios while also being able to communicate. Our maximum time to rendez-vous is however up to 2 times as long as [3]. Both our proposition and [3] are slower than [4] which however does not allow multiple transmission to be emitted or received by a CR at the same time and has a frame latency of at least 54 ms which may not be suitable for some application. It is however well suited for low-traffic networks where our solution would use the spectrum mostly for the beaconing.

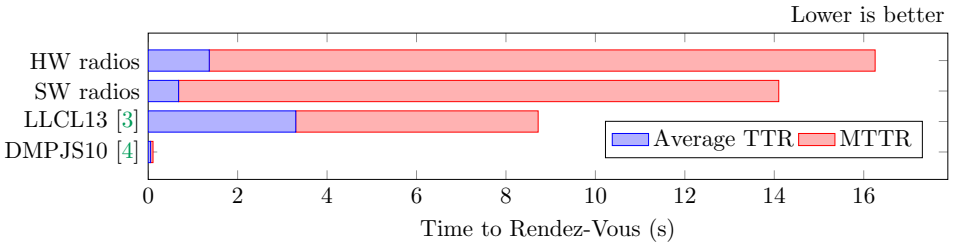


Figure 4.19: Comparing our proposition to [3] and [4] when using a sensing hopping period and a beaconing period of 10 ms, and software (25 MHz) and hardware radios.

Future work involve introducing a shortened version of the beacon that would be sent when the beacon has not changed for some time. This short beacon would only contain the current position in the cycle and the relative time since the beacon last changed (in seconds). This would be sufficient for nodes to keep the time synchronisation. If a new node gets added, it can request the full version of the beacon using the mechanism we already introduced.

4.5 MAC-layer signalling protocol

The role of our MAC-layer signalling protocol is to provide a handshake mechanism that allows unicast communication by selecting the PHY-layer parameters that should be used to transmit a frame.

Contrarily to the PHY-layer signalling protocol which advertises the generally-available frequency bands, the MAC-layer signalling protocol selects a frequency band that is available at the time of transmission and picks a modulation that is compatible with the channel fading and the time available before one of the two nodes jumps to another frequency band. Our proposition is based on the “Willing To Send” (WTS), “Ready To Receive” (RTR) and “Ready To Send” (RTS) control messages, shown in Figure 4.20.

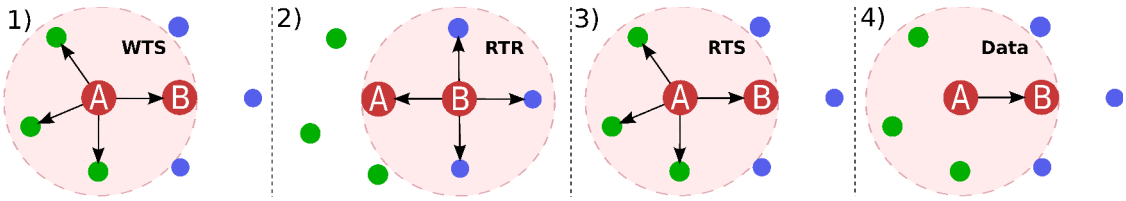


Figure 4.20: Overview of the MAC signalling protocol

In this figure, we can see what happens when node A wants to send a frame to node B. Before sending the data frame, it first sends a WTS frame to tell node B it wants to send a relatively large message. The WTS frame also contains the list of bands available and the possible modulations. Node B selects a band and a modulation suitable for the transmission before sending a RTR frame containing this information. The RTR frame is re-emitted by node A but renamed RTS. Thanks to this signalling protocol, nodes A and B selected the most appropriate bands and the fastest modulation possible for the transmission. Upon receiving a WTS, RTR or RTS frame, surrounding nodes should refrain from communicating in the referenced bands until their expiration time. This effectively allows nodes A and B to allocate spectrum and reduce the chances of collisions due to the hidden-node problem.

4.5.1 The WTS, RTR and RTS frames

The "Willing To Send" (WTS) frame allows a node to initiate the spectrum allocation process along with selecting the most appropriate modulation to use as little spectrum as possible. The following example is a textual representation of a WTS frame:

```
<WTS_frame>{ src=23, dst=12, data_len = 589,
  [ {band1}, {band2}, {band3} ]
  [ {modulation1}, {modulation2} ]
  deadline=152ms, expires=20ms, tx_pwr=20dBm
}
```

In this example, the WTS frame indicates that node *23* wants to send *589* bytes to node *12*. The transmission can happen in any sub-band of *band 1, 2 or 3* and should use *modulation1* or *modulation2*. A band is composed of a central frequency, a bandwidth and the maximum transmission power that can be used without interfering with any neighbouring primary user. A modulation is composed of its type (BPSK, QAM16, ...) and the maximal symbol rate that can be sent. The receiver has *152 ms* to receive the message, starting from the moment the WTS frame got emitted. After this point, the emitter will jump to another frequency band. Every other node receiving this WTS frame should refrain from emitting or accepting new transmissions in these frequency bands, during at least *20ms* or until a follow-up RTR or RTS frame is received. This parameter should be set according to the maximum time node 12 is supposed to take before answering back to node 23. This way, if node 23 is not able to decode this transmission, the lock on the frequency band can be lifted earlier than 152ms.

The WTS frame is sent simultaneously on bands 1, 2 and 3 with a transmission power of *20dBm* which will allow node 12 to assess the fading found at every frequency band and request the wanted TX power from node 23 in order to have a sufficient SNR to reach the fastest modulation possible.

Upon reception of the WTS frame, node 12 uses the content of the Radio Event Table defined in 4.3.3 and shown in Figure 4.5 in order to select the best candidate bands that intersect with the available bands found at node 23 and that have not been reserved by another node's WTS, CTS or RTS frame. Based on the fading found on the selected band (5dBm - received power before amplification) and the maximum emission power of node 23 for each selected band, node 12 selects the bands with the highest SNRs. Node 12 then selects the modulation (type and symbol rate) which: (i) is compatible with its capabilities and the transmitter's, (ii) fits in the frequency band selected, (iii) works with the expected SNR and (iv) allows transmitting the 589 bytes fast-enough to meet the deadline. If a solution that meets all the criteria is found, then node 12 can emit on the selected band the following RTR frame that contains the PHY parameters (selected band, modulation type, symbol rate and transmission power) node 23 should use to transmit its frame. Nodes receiving this frame can now cancel all the reservation on all the frequency bands selected by the previous WTS frame and only lock the band found in the RTR frame for the next 24ms. This delay is the expected maximum time it will take for the emitting node to handle the RTR frame, emit the RTS frame and send the data frame at the expected data rate.

```
<RTR_frame>{ src=12, dst=23, data_len = 589,
  {band}, {modulation}, tx_pwr=15dBm,
  expires=24ms
}
```

When receiving this RTR frame, node 23 should immediately emit the following RTS frame on the selected band containing the frequency band that will be used for the transmission to node 12 along with the time during which the surrounding nodes should refrain from using it. The RTS frame takes precedence onto the previous RTR and WTS frames. The expiration time is lower than the RTR frame because it takes time for the samples to reach the software that will demodulate the signal and decode the frame.

```
<RTS_frame>{ src=23, dst=12, {band},
  tx_pwr=15dBm, expires=19ms
}
```

In this frame, it is said that node 23 now has 19ms to send the data using the PHY-layer parameters that were defined in the RTR frame. The 4ms difference with the RTR's expiration time is an example of the processing time needed for the samples to reach the signal processing software, being demodulated, decoded and sent to the network stack. Vice versa for the emission of the RTS frame.

The WTS/RTR/RTS frames' binary representation is shown in Figure 4.21. The minimum size of a WTS frame is 38 bytes, with an additional 8 bytes per added bands

and 4 bytes per added modulation. The size of the RTR and RTS frames is respectively 34 and 30 bytes.

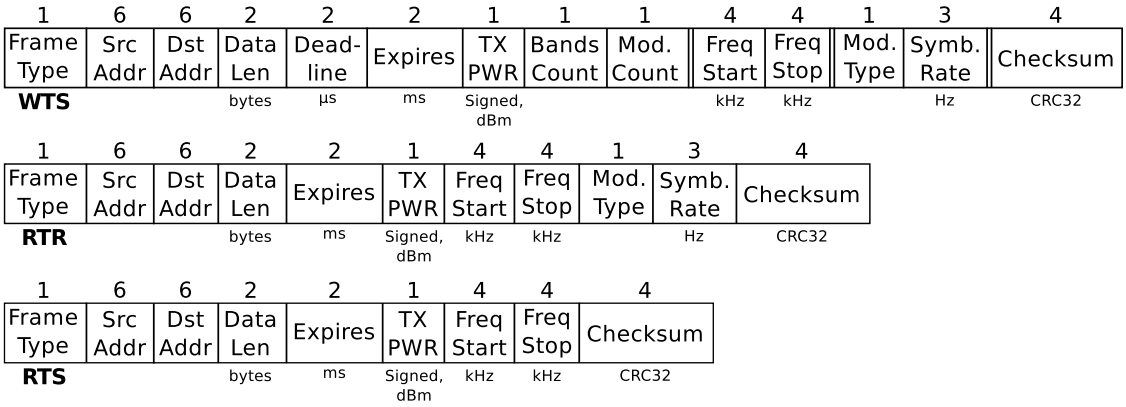


Figure 4.21: Format of the MAC frames (WTS, RTR then RTS)

Our control frame are thus about twice as long as IEEE 802.11's RTS/CTS frames which are respectively 20 and 14 bytes long. They can however make up for their increased size by allowing multiple transmissions to happen on adjacent bands.

4.5.2 Selecting the modulation of the control frames

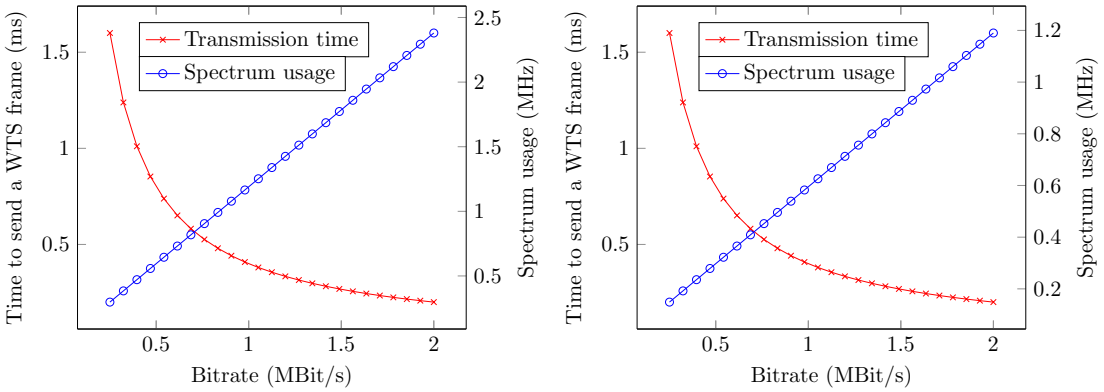
The WTS, RTR and RTS frames should use a modulation that supports poor SNRs to increase the chance of reception by the destination and the surrounding CRs which would in turn lower the chances of a collision during the transmission.

Modulations supporting a poor SNR are usually slow and, at a constant bitrate, use more frequency spectrum because they need more symbols per second than a faster modulation. Figure 4.22 shows the impact of the modulation and the bitrate on the transmission time of a WTS frame and on the spectrum usage. The transmission time does not take into account the preamble or any error-correcting code such as FEC which would both increase the emission time.

From Figure 4.22, we can see that the emission time of the control frames is non-trivial. Frames should thus be relatively large to make it worth it to perform this bandwidth allocation. For streaming applications, the allocation may be renewed continuously by periodically re-sending the RTS and RTR frames on both side of the band used for communicating. This would also result in a higher throughput as the communication would never be interrupted. A CR may reclaim the band by sending a WTS including the band that it is willing to use.

4.5.3 Discussion

Our MAC-layer signalling protocol is, in its nature, an enhancement of the IEEE 802.11 RTS/CTS handshake. It thus inherits its characteristics among which is to partially



(a) BPSK modulation **(b)** QPSK modulation

Figure 4.22: Impact of the bitrate on the transmission time of a WTS frame of 50 bytes using a BPSK modulation (4.22a) and a QPSK modulation (4.22b).

prevent the hidden node problem. However, since we are using this mechanism in a CRN environment, nodes may hop in and out of the frequency band selected during the transmission of a frame. If a node hops in the frequency band after the RTR frame is sent, is far-enough from the emitter not to be able to sense the transmission and yet is close enough to interfere with the receiver in case it decides to send data on this band, it may create a collision. One way to mitigate this problem is to have a relatively low hopping frequency compared to the time it takes to send a frame and to mandate that nodes should be mute for a few ms when hopping on a new band.

Aside from the noted limitation and the increased size of the frames, our proposition should behave in the exact same way as the IEEE 802.11 RTS/CTS handshake and the same limitations and mitigations apply. This means that when the risks of collisions are low or when the message to be sent is small, this mechanism can be bypassed as it would only increase latency and decreases throughput needlessly.

4.6 Conclusion

In this chapter, we proposed a sensing technique for detecting transmissions from primary and secondary users in order to identify available frequency bands. An efficient software and hardware architecture was then proposed to implement this sensing technique.

We then proposed a discovery mechanism that allows nodes to temporally and spatially synchronise to enable communication among CRs. This mechanism is a clear improvement over the state of the art because it allows guaranteeing availability on a number of bands while allowing nodes to perform sensing on other bands or saving power by powering-off their radios. In our simulations, the synchronisation could happen in less than 15 seconds and as little as 660 ms in average for realistic scenarios that do not require any prior knowledge of the PHY parameters, as demonstrated in Appendix F.

Our MAC-layer signalling protocol proposition brings advanced cognitive behaviours to frame transmissions by allowing to pick the least-crowded frequency band, the weakest transmission power and the fastest modulation possible to reach the destination while creating as little interference as possible on surrounding primary or secondary users.

With such a signalling protocol, it becomes possible to allocate spectrum according to the actual need of applications. For instance, it is possible to allocate a narrow band communication streaming data constantly at a low latency which would be very suitable for VoIP applications while also using intermittent wide-band communications for web-browsing or file transfer applications. It thus becomes possible for an application to use both the spectrum and the modulation that is the most suitable for the type of the current communication without needing several radios.

These contributions allow for a more efficient RF spectrum sharing among cognitive nodes to create a cognitive network resilient to interference that can operate alongside non-collaborative nodes but also allocate the most suitable modulation possible for applications. An example of usage of these contributions can be found in Chapter 7.

Future work will focus on designing a MAC layer that can use our propositions to adapt to any kind of traffic and QoS needs and use the least amount of spectrum possible while still being able to share it with surrounding nodes. This is especially necessary for continuous communications and multicast communications. The possibility of using a cross-layer approach to improve the performance will also be studied by using our MAC signalling protocol less often or for bigger frames in bands where the probability of collisions is very low.

Chapter 5

Hardware: Defining an autonomic low-power node

Contents

5.1 Introduction

Up until now, we provided contributions to many layers of the OSI stack. We we have however mostly ignored the power cost of computing. This was a reasonable assumption for sensor nodes since their barely use any of it. However, cognitive radio nodes are entirely reliant on processing power and they thus require potentially-power-hungry hardware.

Historically, processors manufacturers were aiming at increasing performance as it was the usual metric used by consumers to select what to buy. However with the rise of laptops and smartphones, consumers have started to prefer higher battery-life, slimmer, quieter, and cooler devices. This led processor manufacturers to not only take performance into account, but also performance-per-Watt. Higher performance-per-Watt means lower heat dissipation for the same amount of computing power, which in turn allows shrinking the heatsink/fan that keep the processor cool. This results in slimmer and/or quieter devices. Decreasing power usage is also a major concern for datacenters and supercomputers, as they already consumed 1.5% of the USA's electricity production in 2007 [21].

As power management (PM) is a non-functional/auxiliary feature, it is usually non-standard, poorly documented and/or kept secret. As some PM features require the intervention of the operating system (OS), a driver is often needed to obtain the best performance-per-Watt.

Designing power-aware computers has been a hot topic for many years in the research community but the absence of in-depth documentation and closed-source drivers sets back research as it severely limits experimentation and requires reverse engineering the undocumented features which may not even be advertised, then implementing a driver for the hardware. This lack of documentation is most present in the GPU world, especially with the company called NVIDIA which has been publicly shamed for that reason by Linus Torvalds [95], creator of Linux.

We have worked for more than 4 years to build up technical knowledge on power management with students, engineers and researchers working on the Nouveau project [96] which aims at creating an Open Source driver for (almost) all NVIDIA GPUs. This work involved reverse engineering NVIDIA GPUs, reading patents and source codes, and discussing with engineers from AMD, Intel and NVIDIA. This driver, primarily developed through reverse engineering, has been the default NVIDIA driver in the Linux kernel since December 2009. Nouveau is developed collaboratively and most of its developers are hobbists, students, researchers or engineers. As of today, 3 engineers are paid to work on Nouveau, 2 of which are paid by Linux distributions. The other one joined in early 2014 and is paid by NVIDIA to add support for their System-on-Chip (SoC) GPU Tegra K1. Currently, the driver is able to accelerate 2D, 3D and video decoding but its inability to change the frequency of the clocks limits its performance. Along with other members of the community, we have been documenting power management and improving it in the Nouveau driver on modern NVIDIA GPUs and hope to turn it into a viable driver for Linux users and as well as a complete open source testbed for power

management researchers. Indeed, Nouveau and/or its reverse-engineered documentation, found in envytools [97], has already been used by researchers to create a GPU task scheduler [98], allow the Operating System to use the GPU as an accelerator [99], to study buffer migration techniques and their impact in [100], [101] and [102] or to implement GPU virtualisation at the hypervisor level [103]. We also published power-related papers on Nouveau [104][105][106][107] that will introduce in this chapter.

Because of all these power management capabilities, we think it is possible that modern processors could already be in line with IBM's vision of autonomic computing, as proposed in 2003 [65].

In Section 5.2, we first give a general introduction to transistors, data storage and the electronic components behind every modern processor. We then introduce in Section 5.3 the power management features found in modern processors. Most of these features have been studied through reverse engineering NVIDIA GPUs because no hardware manufacturer documents them. An analysis of performance and power consumption in a desktop PC environment is then presented in Section 5.4. We then evaluate in Section 5.6 how modern processors fit in IBM's autonomic computing vision. Section 5.5 studies the power and performance policy used by NVIDIA on its Kepler GPU family. We conclude in Section 5.7.

5.2 General introduction to power management

This section is meant to provide the reader with information that is important to understand the following sections.

5.2.1 Voltage regulation

In wireless networks, the energy source is likely to be a non-rechargeable battery for sensor nodes because it is the cheapest source of electricity while providing the highest energy density and the lowest self-discharge current. These batteries reduce the size of the sensor node and are thus particularly fit for sensors that do not have an energy-harvesting device such as a photovoltaic panel and should be operational for 10+ years.

Rechargeable batteries for nodes such as a smartphone, laptop or weather monitoring stations are characterised by a higher self-discharge current and a limited discharge current if the battery is to last. Regulating the power consumption of a wireless node can thus be needed in order to avoid stressing the battery.

A modern desktop GPU draws its power from the PCIe port or PCIe connectors (6 or 8 pins). The PCIe port and 6-pin PCIe connectors can each source up to 75W while the 8-pin PCIe connector can source up to 150W.

These power sources all provide voltages that are higher than the operating voltage of the unit they power. A DC-DC voltage conversion is done using a voltage regulator

that transforms the input power voltage to the operating voltage of the unit. The output voltage can often be adjusted dynamically using binary output pins or an I2C bus. Figure 5.1 illustrates a simple view of the power subsystem of a GPU.

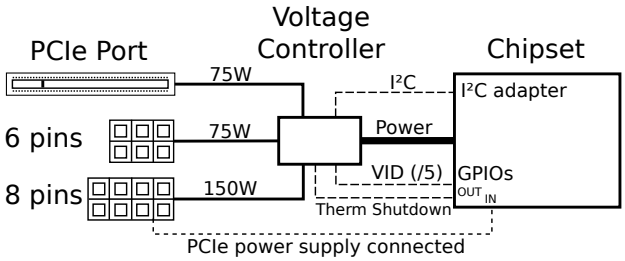


Figure 5.1: Overview of the energy sources of a discrete NVIDIA GPU

The voltage regulator should balance the load across multiple power inputs but must also limit the power draw of each power input to their maximum power rating. If the total power draw gets close to the sum of the maximum power rating of each power input, the voltage regulator should send an interrupt to the processor. If no drastic action is taken by the processor, it should cut down the power in order to protect itself and the energy source.

Voltage regulators are expected to keep their output voltage constant, even when the load varies. They are however not perfect and can only react to a limited dI/dt without the voltage dropping or increasing too much. This is of great concern for power amplification in radios because the power output is highly variable and needs to be highly accurate for amplitude-based modulations such as QAM. For this reason, many power amplifiers are usually connected straight to the battery in order to increase the Signal-to-Noise Ratio (SNR) at the expense of increased power consumption and dissipation. Envelope-tracking power amplifiers are specifically designed for RF radios in order to keep a high linearity in the amplification while providing just enough voltage for the amplifier to operate [58]. This greatly improves the efficiency at low amplification. The power savings achievable using Envelope-Tracking can be seen in Figure 5.2.

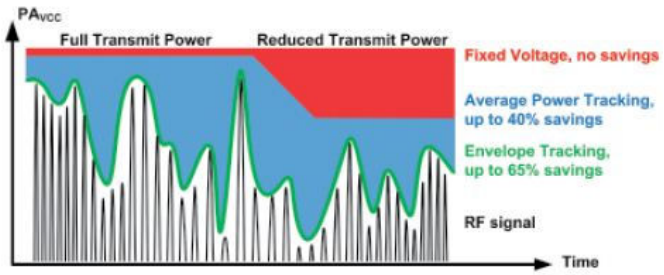


Figure 5.2: Power amplifier supply voltage for fixed voltage, average power tracking and Envelope Tracking. Source [5].

In [60], the authors claim to have achieved efficiency improvements ranging from 11.3% to 40.7% depending on the technology and the output power.

A bigger capacitor could be used to keep the voltage constant under highly-variable loads. It would however slow-down the voltage changes and would also cost money because of its unit price and because it would increase the PCB's size. There is thus a trade-off between voltage stability, how variable a load can be and how fast voltage can be changed to the desired level.

5.2.2 Clocks

Processors are synchronous machines that follow one or multiple clocks. Modulating the clock's frequency on the fly allows to dynamically change the performance of a processor. The source clock is usually an on-board crystal of a few MHz. The frequency of this clock can then be increased by using a Phase-Locked Loop (PLL). A PLL takes the input frequency F_{in} and outputs the frequency F_{out}. The relation between F_{in} and F_{out} in the simplest PLL is detailed in equ. 5.1. The parameters N and M are integers and have a limited range. This means that not all output frequencies are achievable.

$$F_{out} = F_{in} * \frac{N}{M} \tag{5.1}$$

Another common component involved in the clock tree is the multiplexer. A multiplexer takes as an input several signals and only outputs one of them. The selection of the output signal is controllable electronically and is exposed by a register to the driver. Figure 5.3 is an example of a 4-inputs multiplexer with 2 select-lines to be able to address every input.

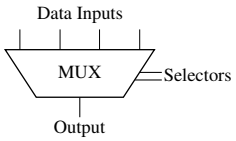


Figure 5.3: Example of a 4-data-input multiplexer

It would be simplistic to think that modern processors use a single clock. For instance, a modern GPU has multiple engines running asynchronously which are clocked by different clock domains. We found out through reverse engineering that there are up to 13 clock domains on NVIDIA's Fermi [108] chipset family. An example of a clock tree for a clock domain can be seen in Figure 5.4.

5.2.3 Transistors

Modern processors are composed of CMOS (Complementary Metal Oxide Semiconductor) transistors. They are characterised with a very-fast switching time, a low power consumption compared to other technologies and a wide range of operating voltage.

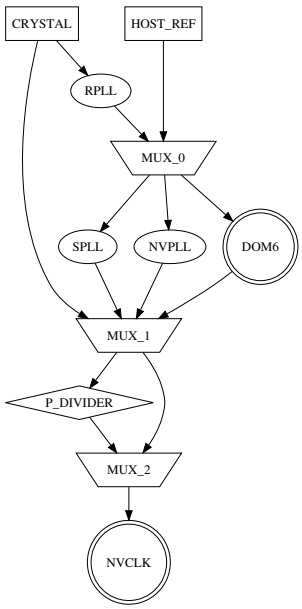


Figure 5.4: Overview of the clock tree for nvclk (core clock) on an NVIDIA nv84 GPU

A P-channel MOS transistor is a simple electronic switch with three poles that lets the current flow from the Source pole to the Drain pole when the voltage applied to the Gate is higher than the threshold voltage V_{th}. A CMOS inverting gate is shown in Figure 5.5.

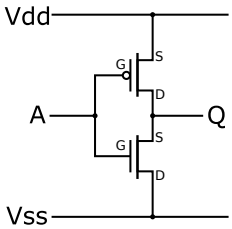


Figure 5.5: Schematic of a CMOS inverter gate

There is a known relation between the voltage, the frequency of the clock and the final power consumption for CMOS transistors [109] as shown by equ. 5.2, 5.3 and 5.4.

$$P = P_{dynamic} + P_{static} \tag{5.2}$$

$$P_{static} \propto V * I_{static} \tag{5.3}$$

$$P_{dynamic} = CfV^2 \tag{5.4}$$

P is the final power consumption in Watts of the transistor and results from both its dynamic ($P_{dynamic}$) and its static (P_{static}) power consumption.

The dynamic power consumption comes from the energy cost of charging and discharging the capacitance C switched by the transistor (which decreases with the transistor size). It is also linear with the frequency f at which the transistor is switched and the voltage V at which the transistor is powered at [6].

The static power consumption is a constant leakage, even when the transistor is in the off state. This is analog to a leaky faucet. I_{static} is dependent on the voltage and the temperature.

The most efficient way to drive a CMOS transistor from a dynamic power consumption perspective is to supply it with the lowest possible voltage. However, if set too low, the transistor will not be fast enough at switching the voltage to the right side of the decision threshold (V_{th}) of the gate of the next transistor. The relation between voltage and frequency at which the transistor is switched depends on equ. 5.5.

$$f_{max} \propto (V - V_{th})^2/V \tag{5.5}$$

Reducing the voltage requires lowering V_{th} if f_{max} should be affected. As the static power consumption is inversely proportional to the exponential of the V_{th} [110], this static power consumption could become non-negligible or completely negate the dynamic power consumption savings of reducing the voltage. Future processors are expected to have different voltage thresholds depending on the performance needed from the transistor.

The distribution of power consumption between the dynamic and the static power consumption is very dependent on the etching process. Static power consumption used to be negligible [6] but it became an increasing problem when shrinking the transistors. Figure 5.6 illustrates the trends in static and dynamic power dissipation as foreseen by the International Technology Roadmap for Semiconductors in 2003. However, this figure does not take into account the recent progress in etching techniques such as Intel's 3D tri-gate or high-k metal oxides improvements [110] which drastically reduced static power consumption.

As a summary, for any given etching process, it is not possible to have both the best power efficiency and the highest performance. It can also be more power efficient to lower the clock speed and have more transistors than having a higher frequency processor. This explains why the frequency of our processors is not increasing as fast as it used to but we instead see an increase in the number of cores.

5.2.4 Storing data : Registers, DRAM, SRAM and Flash

The simplest and easiest way of storing data is to use two NOR gates as shown in Figure 5.7. This circuit is called a RS flip flop and is able to hold one bit of data, exposed by the Q output when both the R and S inputs are not set. Setting R will set Q while setting S will clear Q. The Q output will otherwise remain stable until power is lost.

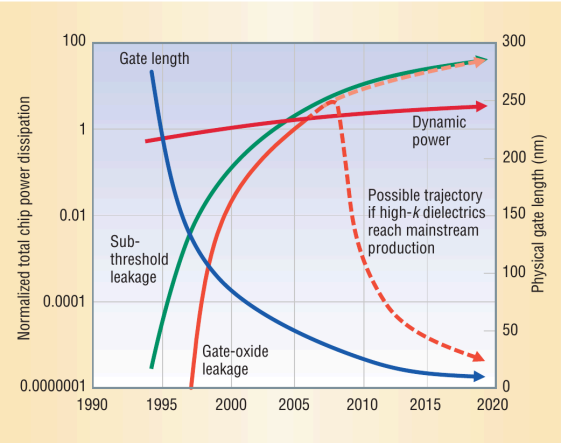


Figure 5.6: Total chip dynamic and static power dissipation trends based on the International Technology Roadmap for Semiconductors [6]

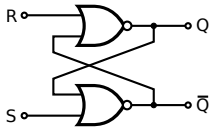


Figure 5.7: Implementation of a RS flip flop using two NOR gates

RS flip flops are difficult to use because they require two different inputs instead of a single data input. It is possible to convert an RS flip flop to a D-type flip flop by adding 3 additional NOR gates. As a CMOS NOR gate requires 4 transistors, this means a D-type flip-flop can be made using 20 transistors. As NOR gates are made using the CMOS technology, we can use equ. 5.2, 5.3 and 5.4 to model the power consumption of the D-type flip flop. Depending on the transistors' etching process, its static power consumption can be negligible or quite high and its maximum speed is proportional to equ. 5.5.

Processor's registers require speed and there are relatively few of them. They thus are usually made of D-type flip flops. However, as using 20 transistors to store one bit takes a lot of silicon space on the processor's die, this is not suitable for storing a large amount of data.

Data can be stored at a very high density using Dynamic Random-Access Memory (DRAM). DRAM only requires one transistor and a capacitor to store one bit as a voltage in the capacitor. Setting the bit requires charging the capacitor while clearing the bit requires discharging it. As the capacitor leaks its charges over time, it has to be refreshed periodically. It also has to be refreshed after being read because reading the voltage drains its charges through the associated transistor. The static power consumption of a DRAM bit comes from the leakage of the capacitor while the dynamic power consumption comes from charging and discharging the capacitor when the bit is read or written. The power consumption can be modelled accurately, as seen in [111]. To save even more space,

eDRAM [112] uses a single transistor to store one bit by exploiting the transistor gate's capacity instead of using an external capacitor. They must however be refreshed more often.

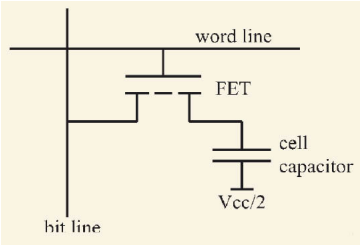


Figure 5.8: Storing one bit using DRAM

DRAM is good for storage density but its performance is limited by the time it takes to charge the capacitor. A middle-ground for storage density and performance exists, it is called Static Random Memory Access (SRAM). A power efficient SRAM uses 6 transistors to store one bit. Like the D-type flip flop, it has non-destructive reads but it requires a dedicated piece of circuitry to be readable which increases the output latency compared to a D-flip flop. SRAM is often used for implementing caches as they are faster than DRAM and use up less space than D-type flip flops. The SRAM's power consumption cannot directly be derived from equ. 5.2, 5.3 and 5.4 because not all the transistors are the same. Different power consumption models have been proposed and tested by [113]. A SRAM cell, shown in Figure 5.9, is however generally more efficient than a DRAM cell.

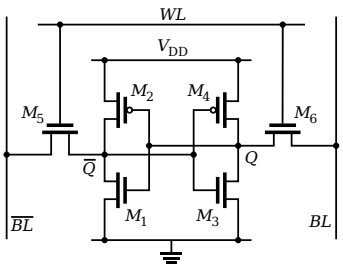


Figure 5.9: Storing one bit using SRAM

D-type flip flops, SRAM and DRAM cells are all volatile data storage mediums. This means that cutting their power source results in a loss of the stored data. Non-volatile data storage techniques such as Flash generally have a higher dynamic power consumption than volatile data storage mediums but can be very efficient for low-performance computers because they can be powered-off when not actively in use. They are however slower for reading and writing operations. As the cells have a maximum number of writes before losing their property, they also require a complex controller to spread the wear across the whole memory cells [114].

5.2.5 Temperature management

Modern processors are prone to overheating. In order to regulate their temperature, a temperature probe is usually inserted inside their package. The processor can then vary the average power received by a fan dedicated to cooling the processor, using Pulse-Width Modulation (PWM). However, not all devices have fans. In this case, the processor should take actions to lower its power consumption. Possible actions will be detailled in the following section.

Not all parts of a computer or wireless nodes are covered with temperature sensors. Maximum average power consumption should then be defined for heating-prone components such as the voltage regulator in order to preserve its properties. The processor should then make sure it does not draw more power than the limit found where the VR heats more than it is able to dissipate.

5.2.6 Power reading

Estimating the power consumption can be done by measuring the voltage drop across a shunt resistor mounted in series with the chip's power line. This voltage drop is linear with the current flowing through the power line with a factor of R_{shunt}. The instantaneous power consumption of the chip is then equal to the voltage delivered by the voltage controller multiplied by the measured current, as per Ohm's law. This method is explained in Figure 5.10.

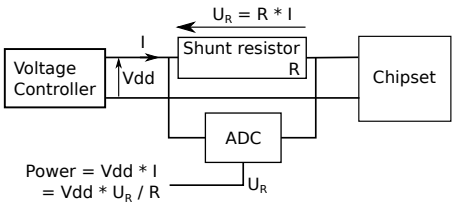


Figure 5.10: Measuring the power consumption of a chip

This solution can however be expensive as it requires dedicated hardware on the electronic board of the node and a communication channel between the Analog-to-Digital Converter (ADC) and the chip. The cost of the solution will depend on how often we want to get a power estimation as fast ADCs are expensive.

5.3 Power management features of modern processors

5.3.1 Accelerators

An accelerator is a piece of circuitry inside or outside processors designed to carry out a more-or-less specific task.

Accelerators are now commonplace from the ultra-low power world to desktop computers. For instance, Texas Instrument's MSP430 [115] is an ultra-low power processor

that has CRC32, AES-256, LCD controller and a true random number generation accelerators. These operations would use a lot of processing power if they were executed by the central processor itself. Similarly, modern Intel CPUs have cryptography-related, 2D, 3D and video decoding/encoding accelerators in order to free the main processor and increase the power efficiency.

In the System-On-Chip (SoC) world which powers most of the smartphones and tablets, the 2G/3G/4G LTE modems have traditionally been implemented using fixed-functions. However, as the complexity grew up, the lack of coordination between all the transistor blocks and redundant functions made these fixed functions expensive in terms of number of transistors (die size) and power consumption. NVIDIA introduced a software defined radio in their discrete NVIDIA i500 modem and their Tegra 4i SoC [116]. This allowed them to shrink their die size to 40% of their previous generation modem while lowering the modem's power consumption thanks to a processor made for baseband processing and DVFS techniques which are difficult to implement in fixed-function processors. Finally, software defined radios can be updated by upgrading a firmware which makes it possible to add new protocols on already-existing devices, lowering the adoption time of new standards.

In the future, we may also expect processors to contain reconfigurable accelerators based on a Field-Programmable Gate Array (FPGA). This would allow compute-heavy applications to ship with an accelerator suited to increase performance and potentially lower the power consumption of the processor. Such accelerator would have direct access to the central memory and be easily programmed and configured through Memory-Mapped Input/Outputs (MMIO), the standard for accessing peripherals and accelerators.

Although accelerators are not strictly power management features, using them can yield a much lower power consumption and an increased performance because they are designed specifically for the task.

5.3.2 Introspection

Sub-devices and low-level parameters

The first real autonomic feature for processors is to be aware of all its sub-devices and hardware constraints. For instance, a system needs to know about temperature and power probes in order to satisfy hardware constraints such as the maximum temperature the processor is allowed to reach or the maximum power it can draw.

Ideally, all of the sub-devices would be connected to a bus supporting enumeration such as USB [117] or PCI(e) [118][119] but this is not always the case and the operating system will need to get a list of devices and their constraints. In desktop/laptop computers, the BIOS or EFI firmware is responsible for abstracting away the platform and exposing the hardware, mostly through ACPI [120]. Discrete graphic cards come with a video BIOS (vbios) that contains different tables for temperature management, performance levels, the list of sub-devices or scripts to initialise the GPU. In the embedded world, a

new standard emerged called "Device Tree" (DT) [121] that enables describing both the hardware and the low-level parameters that should be set.

We found out through reverse engineering NVIDIA's vbios that it contains multiple power-management-related tables. The thermal table contains information such as how to compute the temperature, the temperature thresholds for down-clocking or shutting down the card, how often the fan speed should be adjusted and how (trip-point based or linear with temperature). The vbios also contains the list of subdevices (I2C), the power budget, what voltage should be set depending on the highest clock's frequency, the memory timings depending on the video RAM (VRAM) clock's frequency but most importantly, it contains the list of performance levels.

A performance level contains the list of clocks that need to be set for different performance levels. There are up to 4 performance levels on NVIDIA cards which can be switched dynamically using a Dynamic Voltage/Frequency Scaling algorithm, as described in Subsection 5.3.4. Some clocks may be tweaked at run time through the boost functionality, as explained in Subsection 5.3.7.

We developed and helped develop several tools to deal with NVIDIA vbioses:

- nvagetbios: Reading the vbios from the GPU;
- nvbios: Parse the vbios and display its tables in a textual form;
- nvafakebios: Upload a modified vbios not permanently to the GPU.

These tools can be found in envytools [97] and have been used to reverse engineer the vbios tables previously introduced by altering the vbios and checking what difference it made when running the NVIDIA's proprietary driver. These tools also allowed us to analyse and characterise the impact of different clock frequencies on power consumption and performance in [104] and [105].

Run-time usage information

The (hardware) performance counters are blocks in modern microprocessors that count low-level events such as the number of branches taken or the number of cache hits/misses that happened while running a 3D or a GPGPU application. On NVIDIA's Kepler family, there are 108 different GPGPU-related monitorable events documented by NVIDIA.

Performance counters provide some insight into how the hardware executes its workload. In the case of NVIDIA GPUs, they are a powerful tool to analyse the bottlenecks of a 3D or a GPGPU application. They can be accessed through NVIDIA PerfKit [122] for 3D applications or through Cupti [123] for GPGPU applications.

The performance counters can also be used by the driver in order to dynamically adjust the performance level based on the load usage of the processor. Some researchers also proposed to use performance counters on GPUs as an indication of the power consumption with an average accuracy of 4.7% [124].

We found out through reverse engineering that a counter receives hardware events through internal connections encoded as a 1-bit value which we call signal. This signal is sampled by a counter at the rate of clock of the engine that generated the event. The event counter is incremented every time its corresponding signal is sampled at 1 while a cycles counter is incremented at every clock cycle. This simplistic counter is represented by figure 5.11.

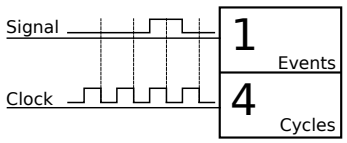


Figure 5.11: Example of a simple performance counter

However, it is expensive to have a counter for every possible signal. The signals can thus be multiplexed. In the case of NVIDIA GPUs, signals are grouped into domains which are each clocked by one clock domain. There are up to 8 domains which hold 4 separate counters and up to 256 signals. Counters do not sample one signal, they sample a macro signal. A macro signal is the aggregation of 4 signals which have been combined using a function. An overview of this logic is represented by figure 5.12.

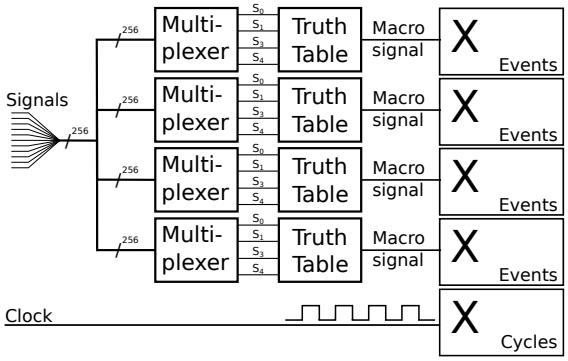


Figure 5.12: Schematic view of a clock domain in NVIDIA's PCOUNTER engine

The aggregation function allows to specify which combination of the 4 signals will generate a 1 in the macro signal. The function is stored as a 16 bit number with each bit representing a combination of the signals. With $s_x(t)$ being the state of selected signal x (out of 4) at time t, the macro signal will be set to 1 if the bit $s_3(t) * 2^3 + s_2(t) * 2^2 + s_1(t) * 2^1 + s_0(t)$ of the function number is set.

5.3.3 Clock and Power gating

In an effort to cut dynamic and static power consumption, processor manufacturers respectively introduced clock gating and power gating.

Clock gating shuts down the clock of unused transistor blocks in order to stop needlessly switching the state of useless transistors. This cuts the dynamic power consumption of the clock-gated block entirely. This technique is usually done in hardware because it is relatively easy for it to know when a block is needed or not. Software can usually disable hardware-based clock gating or force it on a transistor block.

Power gating shuts down the power of a transistor block that is not in use in order to prevent the transistors from leaking current. This shuts down the supply voltage of the transistor which prevents any potential electron leak. The major drawback of this technique is that power-gated registers and caches lose the data they store when power gating, as seen in Subsection 5.2.4. It is thus necessary to save the context before shutting down the power and reload it after putting power back up. Power-gating context-less transistor blocks is however not a problem.

Both clock and power gating can save a lot of power without affecting performance. However, they require adding some complexity to the clock tree and the processing part in order for the processor to know when each transistor block is going to be needed. They also make it difficult for the voltage regulator to provide a stable voltage because they increase the dynamicity of the load, as explained in Subsection 5.2.1.

We found out that NVIDIA implements clock gating by having activity signals telling a transistor block it needs to be active. The block will first receive power, then wait for this signal to be active for a certain amount of clock cycles before activating the clock. Likewise, the block will wait for a number of cycles before gating the clock then wait more cycles before power gating the block. A partial view of what NVIDIA does for clock gating can be seen in FIG. 5 of [125]. Such a mechanism reduces the ping-pong effect by making sure the block really is needed or not before gating it or not. This is very useful for the voltage regulator because it reduces the dynamicity of the load as clock-gating and power-gating have a dramatic impact on the power consumption if the block is large-enough.

We found out through reverse engineering that a minimum of 64 separate clock gating blocks and 52 power gating blocks for the 2D/3D/GPGPU main engine alone on a Geforce GTX 660. The result of our reverse engineering of power and clock gating on NVIDIA GPUs has been made available in envytools [97].

Intel defines different per-CPU idle states ranging from C1 to C6. The higher the C-state, the lower the power consumption and the longer it will take to exit it and return to an active state [126]. Going into a deeper C-state than needed hurts performance, responsiveness and may even negate any power saving because of the time it will take for the processor to exit this deep-sleep state. The C1 state merely clock gates the core's clock while the C6 state can power gate the entire core, interrupt controller and PCIe bus included. Of course, if a resource is shared, it will not be gated unless all the cores sharing it are done requiring it anymore.

We have not found a feature like the C-states in NVIDIA hardware but it can be reimplemented by the host driver as needed.

5.3.4 Dynamic Voltage/Frequency Scaling

Clock and power gating are a great way of lowering the power consumption of a processor when some of its blocks are inactive. This lowers the power consumption without negatively impacting performance, at the expense of an increased hardware complexity of the processor.

Another way of saving power is to dynamically change the frequency and the voltage of transistor blocks, depending on how heavily-used they are. The reduced frequency lowers the dynamic power consumption while the reduced voltage lowers both the static and dynamic power consumption. This technique is called Dynamic Voltage/Frequency Scaling (DVFS).

Since processors may have multiple cores or be composed of multiple accelerators, not all of them may be used at the same level at any given time. Having them driven by different clocks allows scaling the frequency of the engine dynamically in order to lower the dynamic power consumption of the processor. Having separate clock domains for each engine only requires one PLL per clock domain. It is however more difficult to have a power domain associated to every clock domain as it would require a voltage regulator per clock domain.

On Intel processors, all the cores share the same power domain and performance level [127]. Indeed, although the operating system can specify the wanted performance level for each core of the processor, they will all use the maximum performance requested by all the non-idling cores. A more in-depth explanation of power management states can be found on Intel's website [126] and in [8]. Intel's integrated GPU, however, has its own power and clock domain.

Through reverse engineering, we found out NVIDIA are more complex than Intel processors. Discrete GPUs only have up to 2 power domains, one for the chip and one for the video RAM. The number of clock domains went up to 15 in the Fermi [128] family with only 11 reclockable clock domains. This number went down a little in the Kepler and Maxwell families to reach 9. The different clocks are changed all at once by switching between different performance levels depending on the needed performance. These performance levels are found in the vbios, as explained in Subsection 5.3.2. The needed performance is evaluated dynamically using the performance counters that we introduced in 5.3.2. The reclocking policy of NVIDIA will be analysed in Section 5.4.

5.3.5 Thermal & power management

Processors have hardware limits on their maximum power consumption and highest operational temperature. The temperature is usually regulated by using a fan to cool-down the processor which is regulated using the internal an external temperature probe.

In the event the processor could not be cooled sufficiently by the fan or if it is exceeding its power budget, it needs to react by lowering its power consumption. Lowering the power consumption is usually done by reclocking but a full reclocking takes a long time and usually requires the intervention of the Operating System.

Letting the operating system reclock the processor when getting close to overheating is acceptable and the hardware can assist it by sending an interrupt when the temperature reaches a threshold. However, in the case where the operating system is not doing its job because it is locked up or because a driver is not loaded, the chip should be able to regulate its temperature by itself without being forced to cut the power to the voltage regulator's level.

Rapidly lowering the power consumption in hardware

A simple way for the hardware to cut its power consumption is to slow-down the clock of some engines. This will linearly affect the dynamic part of the power consumption, as seen in equ. 5.4. However, it will not be as power efficient as a full reclocking of the processor because the voltage is not changed, as explained in Subsection 5.2.3.

The frequency of a clock can be lowered easily without changing any PLL and can thus be done without the intervention of the Operating System. The easiest solution is the one taken by Intel, called T-state. A T-state, short for throttle state, gates the clock of a core from 12 to 90% of the time in less than 10 different steps [126].

We found out through reverse engineering NVIDIA GPUs that another solution is to divide the clock by a power of two which can be done very easily using counters. It is then possible to generate any average clock frequency between the original clock and the divided one by varying the time spent outputing one clock or the other. This solution has a wider spectrum of achievable slow downs (6.25% to 100% of the original speed) and is also much finer grained as it has 255 steps from the divided to the normal clock. This technique was introduced by NVIDIA in 2006 and, for the lack of a better name, we decided to call this frequency-modulation technique Frequency-Selection Ratio Modulation (FSRM). FSRM can be implemented by using the output of a Pulse-Width Modulator (PWM) to a one bit multiplexer. When the output of the PWM is high, the original clock is being used while the divided clock is used when the output is low. Any average clock frequency between the divided clock and the original clock is thus achievable by varying the duty cycle of the PWM.

Figure 5.13 presents the expected frequency response of the above system along with what has actually been measured through the performance counters when tested on NVIDIA's hardware implementation. The non-linearity of the output is likely explained by the fact that no clock cycle should be shorter than the original clock's which forces to create a long cycle during the transition.

Intel T-States or NVIDIA's FSRM can be used to rapidly lower the frequency of power-hungry clock domains in reaction to hardware events such as overheating. Such event

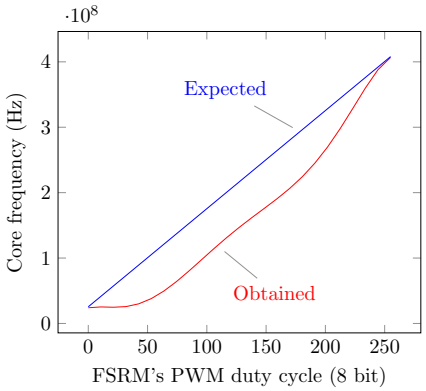


Figure 5.13: Frequency of the core clock (@408MHz, 16-divider) when varying the FSRM

could happen if the temperature mechanism explained in 5.2.5 fails because the fan is too old to spin properly or because the heat sink is clogged with dust.

We found out through reverse engineering that NVIDIA has dedicated hardware that can generate such events when the temperature reaches a certain temperature thresholds. Those temperature thresholds are in fact hysteresis windows to avoid the ping-pong effect of temperature oscillating around this threshold. There are 3 to 6 hysteresis windows depending on the hardware generation. Each window can specify the power-of-two divider and the wanted FSRM value. When two events happen simultaneously, the event with the highest priority will be used to configure the FSRM. This priority is fixed in silicon on the Tesla family (introduced in 2006) but may have become a parameter on newer generations. When no event is happening, the FSRM is deactivated to use the original clock all the time. These windows can thus be put one after the other to gradually limit the performance before shutting the power altogether at the voltage regulator's level.

Power capping - Limiting the power consumption of the processor

Power capping is needed to make sure a processor stays in its power budget, which can sometimes be advertised under the name Thermal Design Power (TDP).

The power consumption can be measured by reading the voltage drop across a small resistor put in series with the power supply, as explained in Subsection 5.2.6. Another solution is to compute the power consumption by having accurate hardware models for both the static and the dynamic power consumption inside a block. It is then necessary to keep track of which blocks are power gated and when blocks are active.

NVIDIA introduced in 2006 a power estimator that keeps track in hardware of which blocks are active. The activity level of the block is then multiplied by a constant dependent on the block's size to account for the fact that blocks do not all have the same amount of transistors. The multiplied activity levels are then summed and filtered to get a reading in mW. An overview of NVIDIA's power estimation technique can be seen in Figure 5.14, extracted from their patent "Power estimation based on block activity" [7].

This patent also describes a communication channel between the power estimator and the power supply, but we have not found any trace of it through reverse engineering. This solution is interesting because it is able to compute the power consumption at 540 kHz which allows for a very fast response to an over-current situation. It however does not take into account the static power consumption nor the voltage. It is not a problem for NVIDIA because they can set the block weights so as the reported wattage is accurate when the board is using the highest performance level.

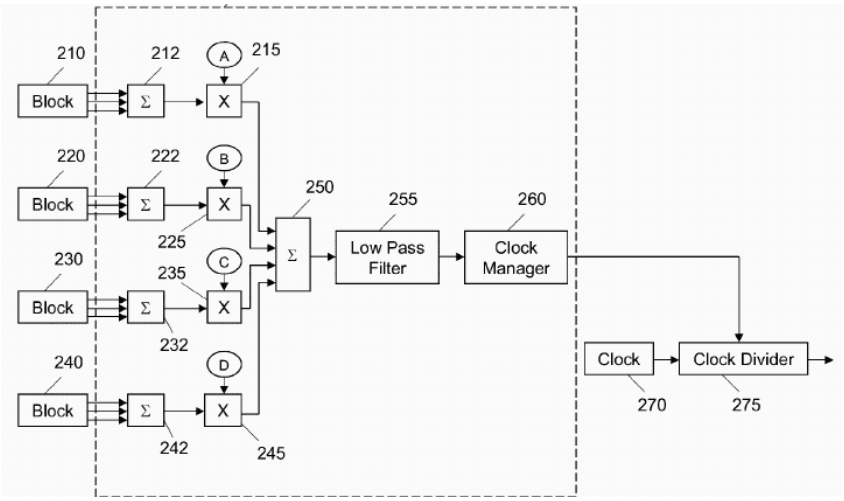


Figure 5.14: Overview of NVIDIA's power estimation technique [7].

Independent researchers managed to have an average prediction error of 4.7% [124] for the power consumption using performances counters to keep track of the active transistor blocks. However, like NVIDIA's solution, they do not track which blocks are power gated, and they do not have a static power consumption model which makes their model invalid in light-loads scenarios. Their model also does not work when no performance counter cover one block. Finally, it uses CPU computing power to compute the current power consumption, produces results at a much slower rate than NVIDIA's power estimator and does not cover all the transistor blocks at the same time because of the limited amount of counters observable at the same time. It however proves that NVIDIA could do a good job with its power estimator.

Intel released a very accurate hardware power consumption model in their Sandy Bridge processor family [8] in 2011, as can be seen in Figure 5.15. It finally takes into account the temperature, the voltage and the power-gated blocks in order to model power consumption accurately [8].

We also found out through reverse engineering that NVIDIA also introduced a hardware power-capping feature in 2006 using the FSRM and the power estimator we just introduced. It is based on two hysteresis windows used to alter the ratio of time spent using the fast or the divided clock. The inner window alters the ratio finely while the

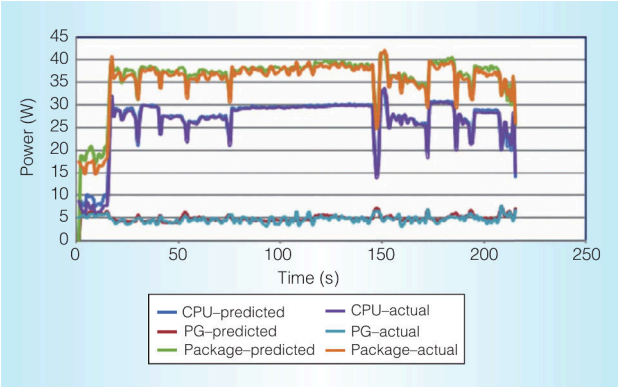


Figure 5.15: Power meter: predicted and actual power of the CPU, processor graphics and total package. The chart presents the actual measured power and the architectural power meter reporting for the IA core, processor graphics and total package. The actual and reported power correlate accurately. Source: [8]

outter one alters it coarsly to quickly react to big power consumption changes. An example of NVIDIA's hardware power capping can be seen in Figure 5.16 where the outer window is set to [130W, 100W] while the inner window is set to [120W, 110W]. The outer window will increase the FSRM value by 20 when the power is lower than 100W and will decrease it by 30 when the power is above 130W. The inner window will increase the FSRM value by 5 when the power is between 120 and 130W and will decrease it by 10 when the power is between 100 and 110W. The FSRM value is limited to the range [0, 255].

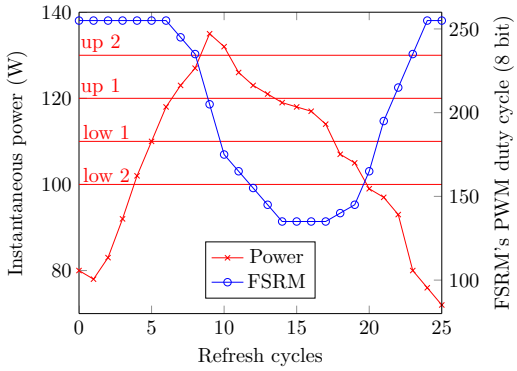


Figure 5.16: Example of the power limiter in the dual window mode

This hardware power capping feature is updated at 540 kHz which makes it very reactive to load changes. Unfortunately, NVIDIA never used this feature. This may soon change with the introduction of the Tegra K1 which is a powerful SoC GPU based on the Kepler family and which may be run fan-less and for which a hardware power model is less expensive than an external power sensor. A reactive power-capping infrastructure could also help NVIDIA squeeze a little more performance while still enforcing their power and

thermal budget. However, on discrete Kepler GPUs, NVIDIA uses an external power sensor which gets polled at 10Hz in order to allow downclocking the card to stay in the budget.

Intel introduced an equivalent technique known as Running Average Power Limit (RAPL) in the Sandy Bridge family in 2011. They documented the configuration registers in their "Software Developer's Manual" [129] but the implementation details are not known as all these computations are done in an internal microprocessor dedicated to power management instead of hard-wired logic like NVIDIA. This may explain why Intel's power estimation is able to deliver a reading roughly at 1kHz [130] instead of the fixed 540 kHz for NVIDIA [106].

5.3.6 Power Management Unit (PMU)

Modern processors now have an embedded processor running a Real-Time Operating System (RTOS) in charge of power management. It was introduced in order to reduce the power management processing load of the operating system. This allows saving more power because the RTOS can be more reactive than the main operating system of the computer and it also allows the main processor to be idle more often, where it is the most power-efficient. As the RTOS does not need to be operating at a fast speed, its transistors can be made to have a really low static power consumption which means they will only consume power when active.

The role of the power management unit can be to do:

- Fan management;
- Engine-level power gating;
- Hardware sequencing;
- Power budget enforcement;
- Reclocking the processor;
- Performance and system monitoring.

Fan management can be done easily by polling on the temperature sensor periodically and scaling linearly the power sent to the fan according to it.

Engine-level power gating requires saving the content of all the registers if we do not want their content to be lost, as we explained in Subsection 5.2.4. Power gating then requires following a precise script involving precise timing which is also known as hardware sequencing.

A PMU can also be required to poll on an external power sensor to enforce the power budget by reclocking the processor or asking the Operating System to do it. It may also poll on performance counters to implement a DVFS policy.

We found out through reverse engineering that NVIDIA introduced a non-documented PMU in 2009, in the GT215 chipset. This PMU is clocked at 200 MHz, has a memory management unit (MMU) and also has access to all the registers of the GPU. It also supports timers, interrupts and can redirect the interrupts from the GPU to itself instead of the host. Several independent communication channels with the host are also available. Starting from the Kepler family, the PMU is clocked at 324 MHz.

NVIDIA's PMU also has performance counters to monitor some engines' activity along with its own in order to send an interrupt to the host when the card needs to be reclocked. In other words, NVIDIA's PMU is a fully-programmable and real-time "computer" with a low-latency access to the GPU that can perform more efficiently whatever operation the host can do. However, it cannot perform heavy calculations in a timely fashion because of its limited clock frequency. This is however sufficient to perform complex operations such as reclocking the GPU.

Intel also has a PMU, called Power Control Unit (PCU) because they already had a Performance Monitoring Unit. Intel's PCU is responsible for computing the instantaneous power consumption, as seen in Subsection 5.3.5. The PCU is finally responsible for selecting the performance level, enforcing the power budget and sharing it between the CPU and the GPU, depending on both their loads [8]. On Sandy Bridge, the PCU uses more than 1 million transistors, roughly the same number of transistors found on an Intel 486 processor (1989). This however represents one thousandth of the number of transistors found in some Sandy Bridge processors released in 2011 [131].

5.3.7 The Boost feature

The hardware power capping feature allows enforcing a power budget. This budget is necessary to protect the :

- power source, especially if it is a battery;
- voltage regulator, which may overheat;
- the processor itself, when overheating (TDP).

Performance levels used to be statically designed so as no matter what application is being run, the processor would never exceed its power budget.

On modern processors, the limiting factor for a processor is often its capacity at dissipating heat. However, if the power usage has been low for some time, the heat-sink can handle dissipating more power than the TDP for some time. This transient power budget would however still be limited by the power supply's and the voltage regulator's maximum power output.

Intel calls this feature "Turbo boost" and it can increase performance by up to 40%, as can be seen in Figure 5.17.

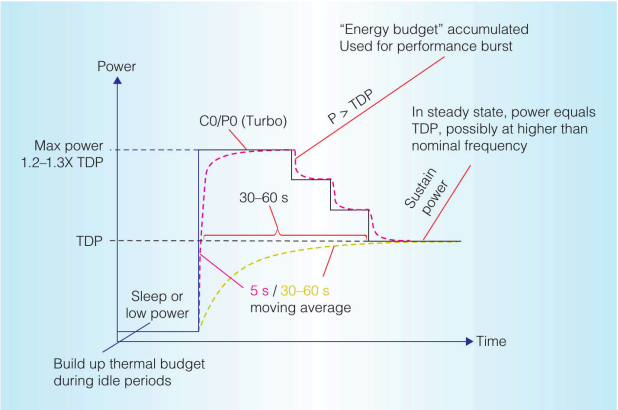


Figure 5.17: Dynamic behavior of the Intel Turbo Boost. After a period of low power consumption, the CPU and graphics can burst to very high power and performance for 30 to 60 seconds, delivering a responsive user experience. After this period, the power stabilizes back to the rated TDP. Source: [8]

The boost feature allows some dynamic flexibility in the way the power budget is allocated to each core of the Intel processors. For instance, if one core is busy and the others are idling, then the frequency of this core can be increased until the CPU's power consumption reaches the TDP. Likewise, when the Intel integrated GPU's performance is limited by its power cap, some of the CPU cores' power budget can be moved to the GPU dynamically.

NVIDIA introduced a similar technique, called "GPU Boost", in the Kepler family in 2012. It was introduced in [132] which states that the fastest performance level is chosen to be safe for the GPU in the worst possible thermal environment even while running the most power-hungry real-world 3D application possible. It also says that in usual scenarios, the GPU is likely to operate with a power consumption lower than the TDP. It is thus possible to safely increase the GPU clocks until the power consumption reaches the TDP.

We found out through reverse engineering that only the clocks of the 2D/3D/GPGPU engine can be adjusted dynamically. The candidate clock domains are referenced in the boost table in the vbios. The boost feature requires knowing the current power consumption. NVIDIA uses an external power sensor (referenced in the external devices table in the vbios) that monitors each voltage regulator's power input lane. For each lane, the power sensor acquires the supply voltage and the voltage across the shunt resistor placed in series in this lane. The resistance of these shunt resistors is either 2 or 5 mΩ, it can be found in the sense table in the vbios. The power budget is stored in a vbios table which states the peak power consumption allowed and the average power consumption which should be equal to the TDP. Finally, the voltage that should be set according to the fastest clock domain is defined in the cstep vbios table. The usual granularity of this table is around 26 MHz.

The boost reclocking policy will be explained in Section 5.5.

5.4 Power and performance analysis in modern NVIDIA GPUs

In this section, we will first study the impact of voltage, temperature and frequency of the clock on the power consumption of a modern GPU. These experiments have been carried out on a GeForce GTX 660, of the Kepler family, which has an embedded power sensor.

This power sensor is similar to the one used in [133], except we did not find any of the delay issues the authors experienced. This is probably due to an excessive averaging done by the proprietary driver, in hardware and/or software along with the inability for the GPGPU application to control when the power sensor should be polled. Because we reverse engineered the communication with the external sensor, we can poll the sensor whenever we want and program it to average as much as we want. The power sensor found in our Geforce GTX 660 is Texas Instrument's INA3221 [134].

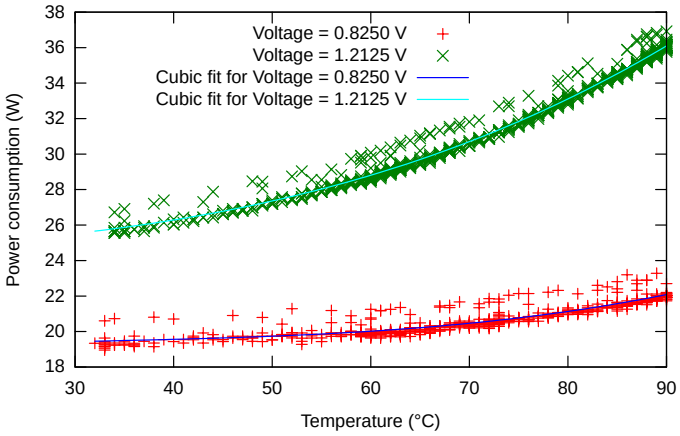


Figure 5.18: Power consumption of our Geforce GTX 660 as its temperature increases. Measurements done at the lowest and highest possible voltages.

In Figure 5.18, we used the power sensor to evaluate the impact of temperature on the power consumption of our Geforce GTX 660. We can see that the higher the power supply voltage of the GPU, the more sensitive to temperature the GPU's power consumption will be. Both curves have been fitted using a cubic function with different parameters. Data has been acquired by bumping the fan speed to the maximum to lower the temperature as much as possible. We then shut the fan down to allow the temperature to rise and to limit the variance due to the fan's own power consumption which will be detailed in Figure 5.19. As the GPU does not produce enough heat to reach 90°C by itself, we used a powerful hair-drier on the heatsink to make the temperature rise. Even though the GPU was as idle as possible, some of the power measurements are far from the fit,

this could be due to the self-refresh mechanism of the VRAM, needed to keep the data in memory (Subsection 5.2.4). At the maximum voltage, the power consumption, when temperature is at 90°C, is 38% higher than at 34°C. Active cooling can thus lower the power consumption by lowering the temperature despite an increased power consumption needed to drive the fan.

We evaluated the power consumption of the fan according to its commanded speed. According to our reverse engineering and to the thermal vbios table, this fan is supposed to be driven using a PWM controller, at a frequency of 2.5kHz. The commanded speed simply is the duty cycle during which the output of the PWM controller is high. This fan should be driven in the range [40, 100]%. In Figure 5.19, we show the relation between the PWM duty cycle, the fan speed in Revolution Per Minute (RPM) and the power consumption of the GPU. To perform our experiment, we set the fan to 100%, polled the power consumption and averaged the wattage before lowering the fan speed by 1% and polling the power sensor again. The fan stopped spinning at 23% which explains the sharp decrease in power consumption. Starting from 24%, the power consumption increased roughly linearly with the duty, peaking at 4.5W. The fan's RPM followed a more logarithmic curve with the power.

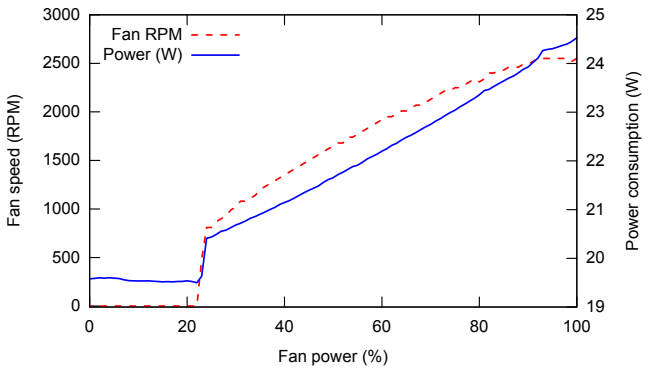
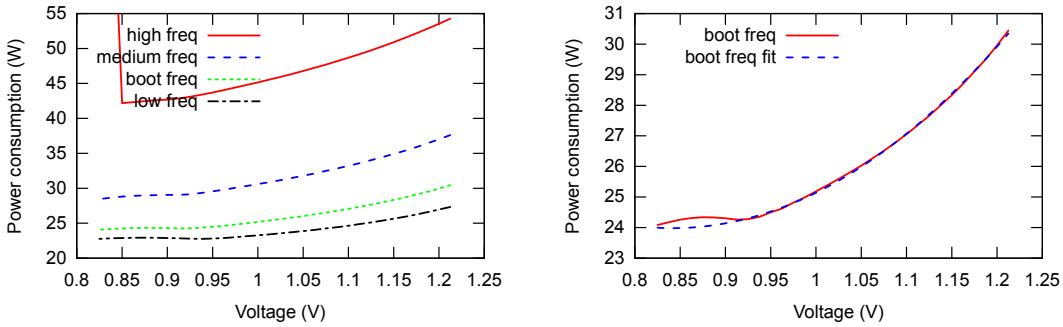


Figure 5.19: Power consumption of our Geforce GTX 660's fan depending on the PWM's duty cycle.

Finding the optimum between fan speed and lower power consumption is however harder because it depends on the thermo-dynamic laws and parameters such as the air temperature, the profile of the heatsink and the airflow generated by the fan. Driving the fan speed linearly with the temperature is however a good heuristic that is used by NVIDIA and other fan controllers such as the ADT7473.

In Figure 5.20, we tried evaluating the impact of voltage on static power consumption. As we do not currently have the knowledge to enable clock gating on most of the chip to cut down the dynamic power consumption to 0, we instead decided to evaluate the power consumption at different performance levels. For each performance level, we sam-

pled power consumption while going from the highest to the lowest voltage possible with the fan speed set to the maximum. Our voltage regulator supports 31 discrete voltages ranging from 0.8 to 1.2V. Throughout the test, the temperature varied from 31 to 34°C. This temperature change is low-enough for us to ignore its influence on power consumption. In general, the lower the performance level, the lower the impact of voltage on power consumption.



(a) Actual power consumption depending on the voltage and frequency

(b) Modeling the impact of power on power consumption using a quadratic fit

Figure 5.20: Power consumption of our Geforce GTX 660 as we vary voltage at a temperature of 32 °C

The core clock in our performance levels is the following: 27 MHz for low; 324 MHz for boot; 862 MHz for the medium and 1254 MHz for the high performance level. However, the exact frequency of each clock in the performance levels is meaningless because we have no idea how many transistors are clocked by each clock domain. Without having perfect knowledge about which portions of the GPU can be clock-gated and which of them are clock-gated by default, it is impossible to accurately use linear regression to get an estimate of the number of transistors in each clock domain. We are however working towards learning as much as possible about clock gating on NVIDIA GPUs which could allow us to get these estimates. For the lowest performance level, we configured as many clock domains as possible to run on a 27 MHz clock (crystal clock). The remaining clocks not running at 27 MHz were the host clock, running at 277 MHz and an unknown clock running at 250 MHz. This custom performance level allows us to lower the dynamic power consumption as much as possible without clock gating every part of the chip which may not even be possible. Using this performance level, the power consumption reached 25W with the highest voltage and the lowest possible temperature.

Since temperature does not affect the dynamic power consumption, we can see from Figure 5.18 that temperature can add up to 10W to the static power consumption at this voltage, pushing the maximum static power consumption of our GPU to 35W. As this GPU's power budget is 124W, the static power can thus represent up to 28% of the total power consumption. In a gaming scenario, with the voltage set to the maximum and a usual operating temperature of 50°C, the static power consumption is limited to

27W, which represents less than 22% of the TDP.

In an idle scenario, performance is not needed. Aside from power gating and clock gating every possible block, lowering the performance level and the voltage can yield power savings. At 32°C, the GPU consumed a bit less than 55W at the maximum voltage and frequency level while it consumed less than 25W at the lowest voltage and lowest performance level. It is thus possible to more than halve the power consumption in an idle scenario.

In 2012, we evaluated [104] the influence of the clock frequencies on a GTX 480. This GPU only had 3 performance levels and did not have the boost feature. It also did not have a power sensor which means we had to use an external power sensor which we polled at 20 Hz to get the current power consumption.

In Figure 5.21, we calculated the energy and time taken to compute 20,000 times a 512 x 512 matrix addition at different core (c-*) and memory (m-*) frequencies. Even with only 3 performance levels, it is possible to save 20% of energy while only increasing the processing time by 6%. The lowest performance level actually consumes more energy although it uses less instantenous power because it takes more time to compute the result.

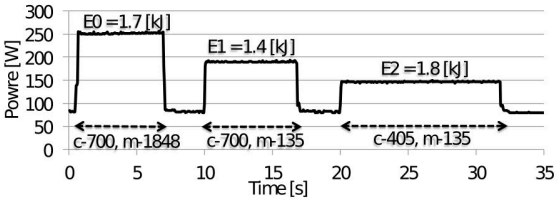


Figure 5.21: Power consumption and execution time of the 512 x 512 matrix addition program

In this example, the memory is mostly idle as all the data can fit into the caches of the GPU, the task is thus compute-bound. Using a GPU with the boost feature, the memory speed could have been kept as low as possible while pushing the core clock to the maximum which would have yielded a lower power consumption for the same processing time. We performed again such a test on our Geforce GTX 660 which has 2 GB of power-hungry GDDR5 RAM. We wrote a compute-bound GPGPU program multiplying big matrices using CUDA and used NVIDIA's proprietary driver to execute it. We then used NVIDIA's coolbits [135] to lower the memory clock and compare the execution time and the power consumption to the original clocks. The results are visible in Table 5.1 and show that 27.8% of energy could be saved while having no performance impact.

In Figures 5.22 and 5.23, we evaluated the relevance of DVFS in an heterogeneous system where both a CPU and a GPU are used to compute data. This test was carried out on an Intel Core i5 2400 processor, a Geforce GTX 480 and the same power sensor used previously to monitor the global power consumption of the computer. The experimental

Core clock	Memory clock	Execution time	Power
1023 MHz	6008 MHz	12.3 s	43.3 W
1023 MHz	1620 MHz	12.3 s	31.3 W

Table 5.1: Power consumption and execution time of compute-bound program depending on the core and memory clocks.

workload is from the Rodinia benchmark suite 2.0.1 [124], a benchmarking suite for heterogeneous computing. In the backprop test, the energy consumption could be reduced by about 28% while only decreasing performance by 1%.

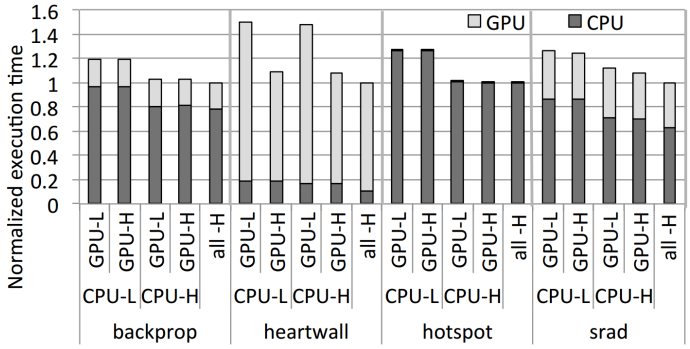


Figure 5.22: Execution time of the Rodina programs

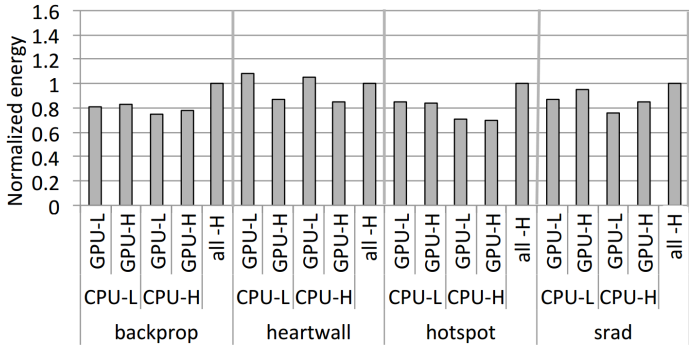


Figure 5.23: Energy consumption of the Rodina programs

In [109], Le Sueur et al. presented that Dynamic Voltage Frequency Scaling was getting counter productive because static power consumption was becoming more prevalent. Three years later, we proved static power consumption can be less than a third of the total power budget in the worst case scenario. Moreover, in [105], we found out that DVFS became more and more effective from one generation of NVIDIA GPU to another. Indeed, in the Tesla family, released in 2006, we managed to save 13% of energy on a test similar to Figures 5.22 and 5.23. The following family (Fermi, 2010) managed to

save 40% while the Kepler family, released in 2012 could save up to 75%. This is in clear contradiction with what the idea presented in [109] which says DVFS is becoming less and less useful. Finally, thanks to the boost features of new processors, it is now possible to change the frequency of processors with a much-higher granularity while performance counters can provide the necessary information to dynamically identify the bottlenecks. The power budget can then be reallocated in order to increase the performance of the processors and accelerators that need more power while slowing down the ones that are not useful to stay in the power budget or to save power.

5.5 Analysing the reclocking policy of a Kepler NVIDIA GPU

In Figure 5.24, we demonstrate the influence of the PMU's performance counters on the DVFS decision on the NVIDIA proprietary driver, version 337.25. To create those figures, we first needed to be able to generate a fake load on any counter while the card was actually idle. We achieved that by forcing the PMU's performance counters to the wanted value by alternately counting every clock cycle or not counting anything. We then needed to get an accurate view of the clock tree to know at which frequency each clock domain runs at. Since Nouveau already has this ability and can be run in the userspace, we decided to modify it in order to make it runnable alongside the NVIDIA's proprietary driver, to parse the clock tree and to output the frequency of every clock domain. Using those two tools, we have been able to observe that NVIDIA does not increase the core clock until the core's activity performance counter reports at least 61% of usage. At this activity level, it takes up to 6 seconds and 17 steps to go from the lowest performance level to the highest one. When increasing the reported activity value, we can observe that these figures drop to 7 steps and about one second when the activity level is 65% and only 3 steps and a tenth of a second when the activity level is 70%. Unfortunately, a low activity level does not declock the GPU unless it is under 10%, in which case it is considered "not in active use".

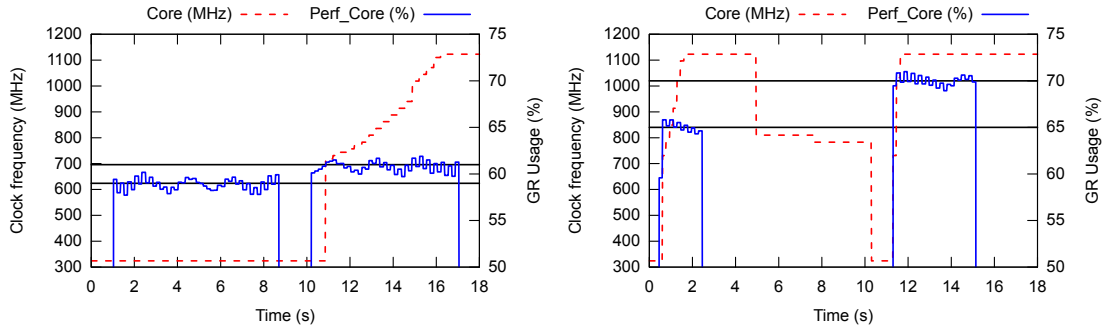


Figure 5.24: Response of the DVFS policy to a high activity level

In Figure 5.25, we show the influence of power consumption on the DVFS reclocking policy. To read the power sensor without risking pertubating the PMU, we decided to create a tool that intercepts the I2C transactions between the external power sensor and NVIDIA's PMU, decode them and compute the power usage from the current and voltage readings sent to the PMU. The power consumption, core's activty level, clocks of the different clock domains and the temperature are then synchronized in one tool outputing a csv file which is then plotted with gnuplot. Since the performance levels are designed so as normal applications cannot consume more power than the TDP, we tried lowering the 124W power budget by modifying the vbios. However, NVIDIA's proprietary driver seems to ignore any power budget under 100W. To make the effect of power on the reclocking policy clear, we decided to modify the shunt resistors' values in the vbios from 5mΩ to 2mΩ which effectively lowered the power budget to 80W. We used the Unigine Heaven OpenGL 4 benchmark [136] in order to generate a load sufficient to reach 110W when not using the power capping technique.

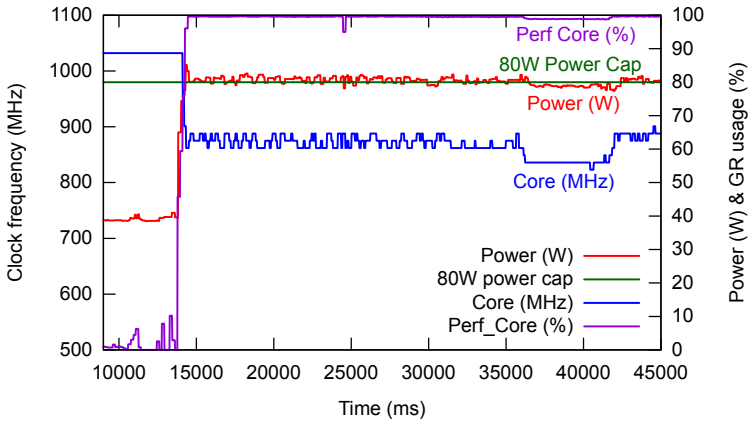


Figure 5.25: Impact of the power usage on the DVFS policy

In the figure, we can clearly see that whenever the power consumption increased over 80 W, the frequency of the core clock dropped sharply and stayed roughly constant until core's activity level dropped a little. This seems to suggest that the reclocking policy takes a decrease in usage level into account only when it is in a reduced-power mode.

The previous two figures seem to suggest that NVIDIA has a power and performance model which allows its DVFS policy to react quickly without overshooting nor having ping-pong effects. Their model seems better than the generic model we proposed in [105] which is precise to about 25% in average when it comes to power predictions but has very high performance prediction errors of up to 65%. This generic model was based on performance counters and uses linear regressions to from a limited set of tests. It was then validated on NVIDIA GPUs of 3 different families. We will not present this model because of its inaccuracy which prevents its usage in a DVFS policy.

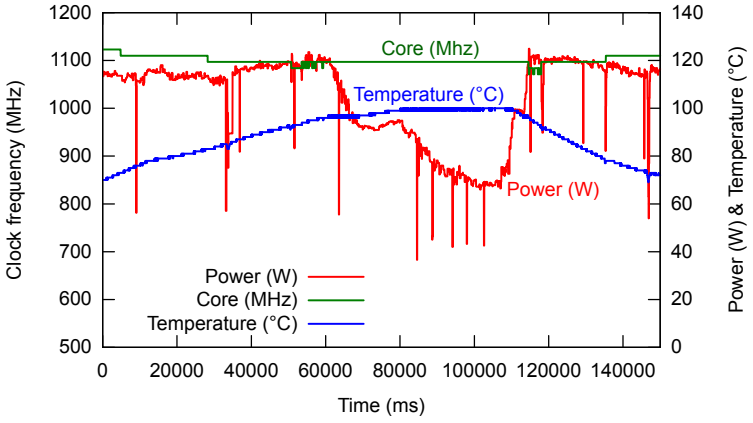


Figure 5.26: The action of temperature on the DVFS reclocking policy

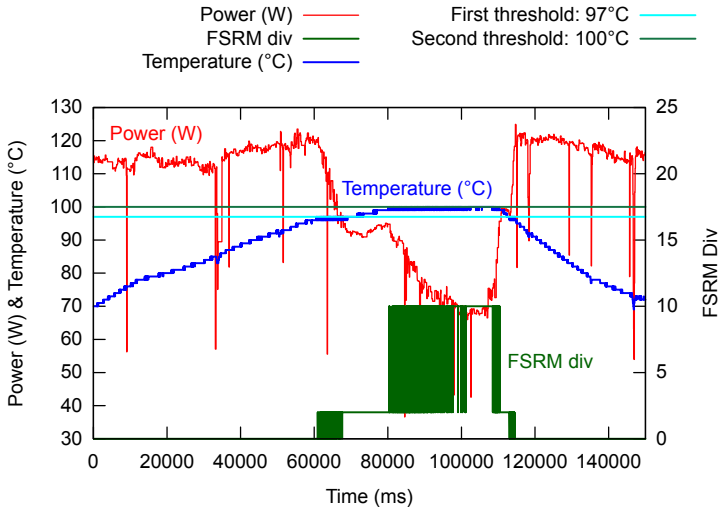


Figure 5.27: The action of temperature on the FSRM

Finally, we looked at the impact of temperature on the DVFS policy. To perform those tests, we completely disabled the fan and used a hair-drier to increase the temperature of the GPU to 100°C. In Figure 5.26 we can see that the core clock got lowered a little when reaching 73°C then 83°C. However, the big power-drop when the temperature reached 97°C and then 100°C is not due to the DVFS reclocking policy, it is due to the action of the FSRM which we introduced in Subsection 5.3.5. As we can see in Figure 5.27, the core clock first got divided by 4 (2^2) when the temperature reached 97°C and then by 1024 (2^{10}) when it reached 100°C. This demonstrates that the FSRM can be very effective at lowering the power consumption of a circuit when needed. If the temperature had kept rising and reached 102°C, the voltage regulator would have been disabled by the GPU to prevent any dommage to the hardware.

5.6 Can modern processors have autonomic power management?

In 2001, IBM proposed the concept of autonomic computing [65] to aim for the creation of self-managing systems as a way to reduce their usage complexity. The idea was that systems are getting more and more complex and, as such, require more knowledge from the technicians trying to maintain them. By having self-managing systems, the user could write a high-level policy that would be enforced by the system itself, thus hiding complexity.

As an example, a modern NVIDIA GPU could perform the following self-functions:

Self-configuration

The GPU is responsible for finding a configuration to fill the user-requested tasks.

Since a GPU is an accelerator, it requires the user to send all its commands through a so-called push buffer. No self-configuration is thus possible for the computation. However, it is possible for the GPU to self-configure enough for it to be able to execute the commands sent by the user. This operation is done partially by the hardware to make the card visible to the host computer. The vbios contained in the card is then executed during the Power-On Self Test (POST) procedure in order to initialiase the clock tree, some engines and set a video mode to display content on the screen.

Self-optimization

The GPU is responsible for knowing how to satisfy the user's needs while consuming as little power as possible.

Power can be saved by lowering the supply voltage and the frequency of the different clock domains (DVFS) without affecting performance dramatically. The performance need and the current bottleneck can be identified using performance counters. The power management unit can then poll on these counters and decide at which frequency should every engine run and at which voltage should the GPU be powered at.

To increase the power savings, it is possible to stop the clock of a transistor block entirely if it will not be used for some time. This can also be followed up with cutting down the power supply of the block to entirely cut the power consumption of this block. The first can be done by the transistor block itself while power gating requires the intervention of the PMU as it needs to save the context.

Self-healing

The GPU is responsible for keeping itself in a good working condition.

Since it has a power sensor and a temperature probe, the GPU can be aware of its over-current and over-temperature problems and react to them by increasing the fan speed and/or lowering the power consumption by downclocking the GPU.

This operation can entirely be done inside the GPU, either by using dedicated hardware like the FSRM or by using the PMU.

Self-protection

Integrity and confidentiality between users can be provided by the GPU in the same way it is done on CPUs, by allocating a virtual address space to each application.

Since GPU cannot usually be preempted while executing a program, it is possible to achieve availability by killing long-running jobs to let other programs use the GPU.

The GPU can also store secrets like HDCP [137] keys internally to encrypt/decrypt the video output on the fly using a dedicated engine called PCRYPT. Such a mechanism can be used to upload encrypted information to the GPU, have it decrypt it and process it. This would make some information unreadable by other clients on the GPU or even the CPU.

Discussion on autonomic processors

Although the aim of autonomic computing is primarily oriented towards lowering human maintenance cost, it can also be extended to lower the development cost. By having self-managing subsystems for non-functional features, integration cost is lowered because of the reduced amount of development needed to make it work on a new platform. Ideally, a complete system could be assembled easily by using unmodified autonomic subsystems and only a limited amount of development would be needed to make their interfaces match.

This approach does not make sense in the IBM-PC-compatible personal computer market as the platform has barely evolved since its introduction in the way that components interact (POST procedure, x86 processors and high speed busses). This renders the development of drivers executed on the CPU cheaper than having a dedicated processor for the driver's operations. However, in the System-On-Chip (SoC) world where manufacturers buy IPs (Intellectual Property blocks) and assemble them in a single-chip system, the dedicated-processor approach makes a lot of sense as there is no single processor ISA and operating system (OS) that the IP manufacturer can depend on. In this context, the slimmer the necessary driver for the processor and operating system, the wider the processor and OS choice for the SoC manufacturer.

This is the approach that Broadcom chose for its Videocore technology which is fully controllable by a clearly-documented Remote Procedure Calls (RPC) interface. This allowed the Raspberry Pi foundation to provide a binary-driver-free Linux-based OS on their products [138]. However, the Open Source driver only uses this RPC interface and is thus not a real driver as it is just some glue code. This led the graphics maintainer of Linux to refuse including this driver in Linux as the code running in Videocore is still proprietary and bugs there are unfixable by the community [139].

Broadcom's Videocore is a good example of both the advantages and the drawbacks of autonomic systems. Adding support for it required very limited work by the Raspberry Pi foundation but it also means the real driver is now a black box running on another processor which cannot be traced, debugged, or modified easily. From a researcher point of view, it also means that it will become much harder to study the hardware and propose new drivers and algorithms. In the case of NVIDIA, this situation could happen if NVIDIA PMU's code was not readable and writable by the host system. Luckily, Broadcom hired a well-known open source developer in June 2014 to write an open source driver for the Raspberry Pi's GPU [140].

5.7 Conclusion

In this chapter, we provided a general introduction to power management and power management features of modern processors. Since power management is often kept secret by processor manufacturers, we have documented most of these features through reverse engineering of multiple NVIDIA GPUs. Some of the features have been re-implemented in the Open Source Linux driver for NVIDIA cards, called Nouveau, which provides a very flexible test-bed for researchers and allow easy reproducibility because this driver is the default one for NVIDIA GPUs on Linux.

We then studied the origin of power consumption in modern processors and identified that temperature, supply voltage and frequency of processors are the biggest parameters for the processors' power consumption. When running at 90°C, our NVIDIA Geforce GTX 660 consumed 38% more power than when running at 34°C. Likewise, when idle at the highest performance level, the GPU consummed 2.2 times more power than when operating at the lowest performance level.

We also analysed the reclocking policy (DVFS) of a modern NVIDIA GPU with regards to performance, temperature and current power consumption. We identified that performance of the main engine is increased when its usage level exceeds 60% and is decreased when its activity drops to 0% or if the power consumption exceeds the rated maximum power consumption (TDP).

We finally discussed about the suitability of modern processors to provide autonomic power management and concluded that it is already possible. It however makes it difficult for applications to predict their performance which can be problematic for real-time applications such as video applications and games. In the future, we can imagine applications asking the processors and their power management drivers to guarantee a certain computing bandwidth and/or a maximum wake-up latency.

Future work will focus on getting stable reclocking support on a modern NVIDIA GPU to experiment with generic and efficient DVFS algorithms. More work will also be done on power- and clock-gating which are the two main power management features which are not entirely understood yet.

Chapter 6

OS: Making real-time power management decisions

Contents

6.1 Introduction

In the previous chapter, we saw that it was possible for processors to behave following the autonomic principles. In this chapter, we introduce how multiple processors or radios can interact with each others.

Indeed, mobile end-user devices such as smartphones and tablets usually have multiple ways of performing the same task. For instance, a smartphone could connect to the Internet using the WiFi adapter or the 3G/4G modem. Likewise, it may perform some calculation on the main CPU or on the GPU using GPGPU. Depending on the current usage of the device, every way can become the most efficient one but, given the very dynamic nature of the user's usage of the device, it is unlikely for a static configuration to always achieve the lowest power consumption while still meeting the user's expected Quality of Service (QoS).

In heterogeneous devices, it is impossible for each sub-device to know when they should be used and what their power budget is. This information can only come from the Operating System (OS) since it is the piece of software that controls all the peripherals and accelerators. The accelerators could then have autonomic power management, as introduced in Chapter 5. In mixed-critical systems, it is also the OS's job to centralise deadline constraints and let the hardware know about the performance needed. Critical jobs must then be executed before their deadline while saving power through DVFS and/or DPM, as explained in [141].

The OS's scheduler is thus in charge of selecting on which processor/accelerator each task should be scheduled on, which network interface should an application use to communicate on the Internet and how to react to the dynamicity of the load generated by the user.

In this chapter, we propose a generic decision flowgraph for power and performance management. This flowgraph allows run-time decision making, even with real-time requirements and is highly reusable thanks to its modular nature. We also implemented a framework around this generic decision flowgraph to minimise the amount of work needed to implement a new power- and performance-management decision engine. This is done by separating the decision process in different blocks that can be re-used later. This eases experimentation during the design phase and the well-defined interface allows dumping debugging textual and graphical debugging information which are used throughout the chapter to showcase our framework. Finally, by using the proposed framework, applications also separate the decision engine from the application which improves its re-usability.

Section 6.2 introduces opportunities for power savings and the state of the art related to run-time power and performance management. The definition of the proposed decision flowgraph is presented in Section 6.3 and its implementation is detailed in Section 6.4. An evaluation of the framework is given in Section 6.5. Section 6.6 concludes and presents the future work.

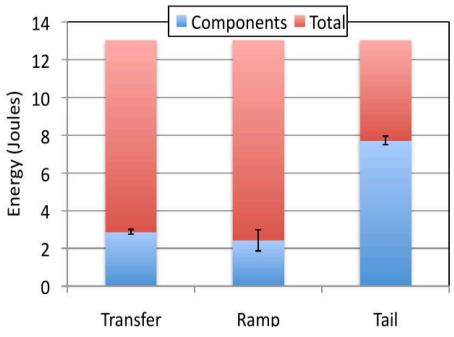
6.2 State of the art

In this section, we both introduce run-time decision engines and their need for power management.

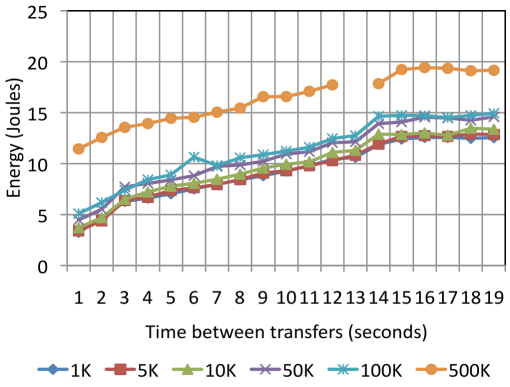
6.2.1 Network interface run-time selection

Mobile end-user devices such as smartphones and tablets usually have multiple ways of accessing the Internet, may it be WiFi (b/g/n), EDGE, 3G, 4G or Bluetooth. All these ways of accessing the Internet are usually provided by two separate radios. The WiFi adapter supporting the WiFi modes b/g/n and Bluetooth while the EDGE, 3G and 4G is provided by another adapter.

In [9], the authors compared the power consumption of the WiFi, 3G and GSM interfaces. The WiFI and 3G measurement have been carried out on an HTC Furze smarthpone while the GSM measurement were carried out with two Nokia phones. The experiment consisted in periodically sending data using each interface and see how the data size and the interval between the transfers influenced the energy consumption of each data transfer. For the GSM and 3G experiments, they break the energy consumption in three parts: 1) *Ramp energy*: energy required to switch to the high-power state, 2) *Transmission energy*, and 3) *Tail energy*: Energy spent in high power state after the completion of the transfer. For the WiFi experiment, the power cost is divided in: 1) *Scan-Associate energy*: Energy spent to scan and associate to an access point, and 2) *Transfer energy*: Energy spent to transfer data.



(a) 3G: Energy components



(b) 3G: Varying inter-transfer times

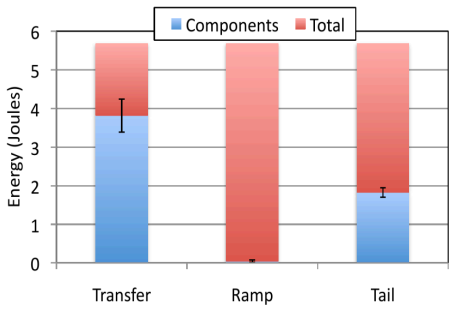
Figure 6.1: 3G Measurements: (a) Average ramp, tranfer and tail energy consumed to download 50K data. The lower portion of the stacked columns shows the proportion of energy spent for each activity compared to the total energy spent. (b) Average energy consumed for downloading data of different sizes against the inter-transfer time. Source: [9]

In Figure 6.1, the authors have shown that the energy cost of a 1K data transmission over a 3G network consumed roughly the same energy as a 100K transfer. This is because most of the power consumption is not due to the transmission cost but rather due to

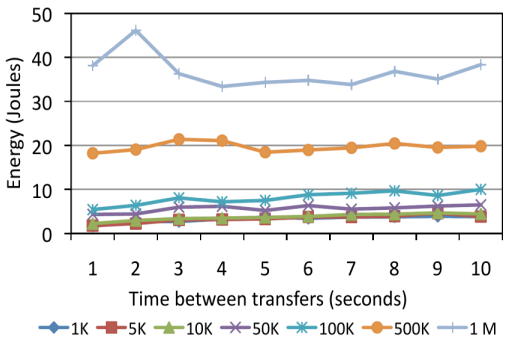
the *Tail energy*. This can be explained because a 3G interface has three states. The idle state is when the interface is not used to download or upload data. In this state, the radio is not connected to the network. From the idle state, the radio can transit to the Dedicated Channel (DCH) state which allows high throughput and a low delay for transmissions. The radio can then transit to the Forward Access Channel (FACH) state which is a lower-power state which is used when there is little traffic to transmit. From the FACH state, the radio interface can either return to the DCH or the idle state.

A high *Tail energy* means that the radio spends too much time in the FACH state instead of returning straight to the idle state. The energy consumption increases sharply with when the inter-transfer time is in [1, 3] seconds. This is because the radio is always in the DCH state. The energy consumption increases less sharply in the inter-transfer [4, 14] seconds range because the radio is put in the FACH mode which has half the power consumption of the DCH state. In the [15, 19] seconds range, the energy consumption reaches a plateau because the radio puts itself into the idle mode which cuts down the power consumption of the radio almost entirely. The time spent in this mode does not increase the energy cost of transmissions.

In Figure 6.2, the same experiment was carried out with very different outcomes. Because GSM is stateless, the radio is put back to sleep as soon as the transmission is over. This explains why the inter-transfers time does not influence the energy consumption of a transmission much. Contrarily to the 3G interface, the energy cost of a transmission is mostly due to the size of the transmission.



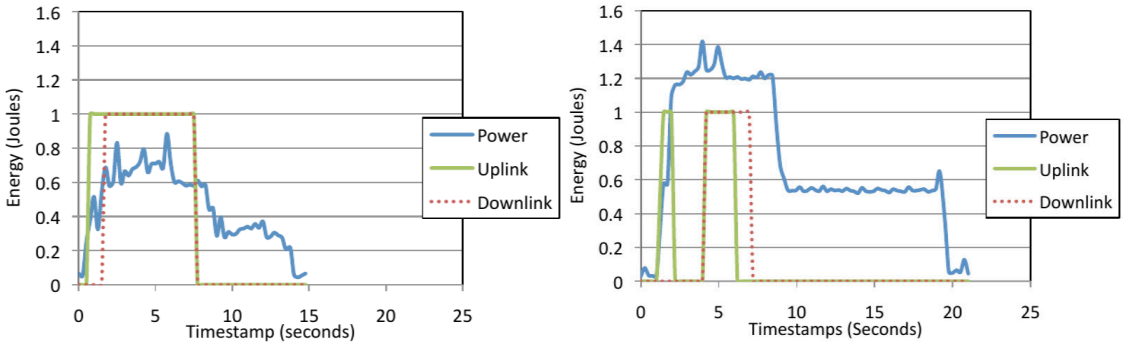
(a) GSM: Energy components



(b) GSM: Varying inter-transfer times

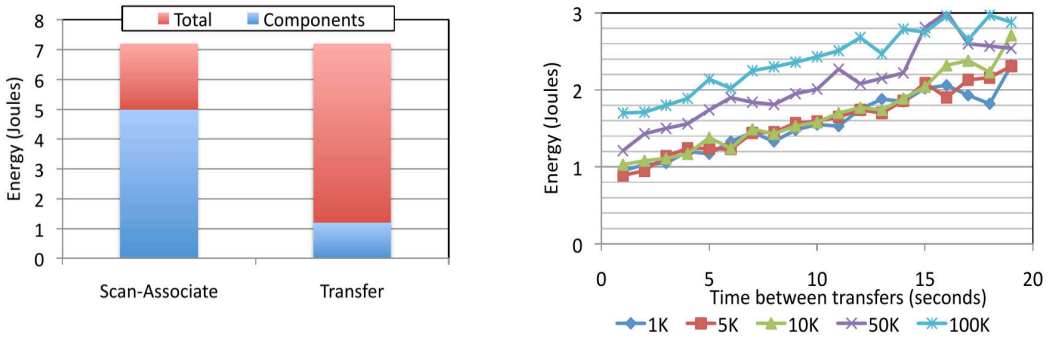
Figure 6.2: GSM Measurements: (a) Average ramp, tranfer and tail energy consumed to download 50K data. The lower portion of the stacked columns shows the proportion of energy spent for each activity compared to the total energy spent. (b) Average energy consumed for downloading data of different sizes against the inter-transfer time. Source: [9]

In Figure 6.3, we can see the power consumption profile (not energy, like the original authors wrote) of a GSM and a 3G interface transferring 50K of data. We can see the same power consumption profile, but the tail time of the GSM transfer is half of 3G's. The power consumption is also much lower for such a small data size, but the 3G transfer was 3 times faster than GSM's.



(a) GSM: Power Profile - 50K **(b)** 3G: Power Profile - 50K

Figure 6.3: Power profiles of 3G and GSM networks. Source: [9]



(a) WiFi: Energy components **(b)** WiFi: Varying inter-transfer times

Figure 6.4: WiFi Measurements: (a) Average ramp, tranfer and tail energy consumed to download 50K data. The lower portion of the stacked columns shows the proportion of energy spent for each activity compared to the total energy spent. (b) Average energy consumed for downloading data of different sizes against the inter-transfer time. Source: [9]

In Figure 6.4, the authors used the same experiment again for the WiFi interface and found out that most of the power cost of transmitting 50K of data was due to scanning to find the right access point and connecting to it. Once this was operation was over, the transfer size impact was not as high as the found in GSM or 3G. Interestingly, the power consumption increases linearly with the inter-transfer time. The authors claim this is due to the high cost of keeping the interface connected (3 to 3.5 Joules per minute).

In Figure 6.5, the authors focused on the impact of data size and compared the different network interfaces. The WiFi interface has been split in two: *WiFi + SA* is the cost of scanning, connecting to the access point and sending the data whereas *WiFi* is only about sending the data. The offset between the two is thus constant but shows how GSM can be more efficient than WiFi in case the WiFi interface is not already associated.

Unsurprisingly, WiFi is more efficient than GSM in most usages except when idle because of the short distance between the emitter and the access point. However, if no

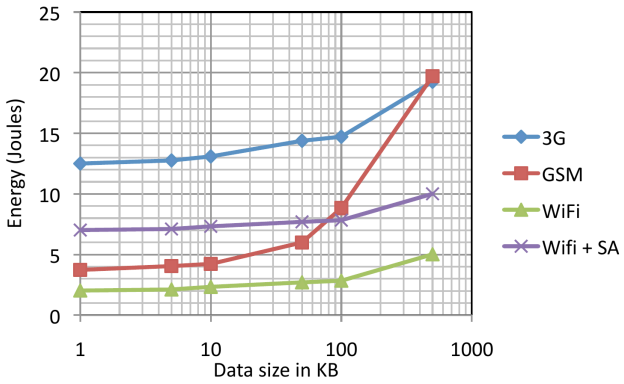


Figure 6.5: WiFi versus 3G versus GSM measurements: Average energy consumed for downloading data of different sizes against the inter-transfer time. Source: [9]

WiFi connection is possible, either because there are no access points or because the user is moving too fast, then a 3G connection will most likely become the most power efficient for usages such as browsing or downloading emails. GSM is more efficient for an idle scenario where the users wants to keep his/her connections opened but is not actively using the connection. It thus important to be aware of the current network usage and the power consumption and performance characteristics of the available network interfaces in order to select the most efficient interface fulfilling the needed QoS.

6.2.2 Processor/Accelerator run-time selection

In this subsection, we study the different types of asymmetry found between processor cores or accelerators. We then evaluate how we can take advantage of this asymmetry at run time to lower the power consumption or boosting performance.

ARM's big.LITTLE

As we explained in 5.2.3, one processor can either be very fast or very power efficient because transistors need to be etched in different ways to achieve one or the other. ARM has tried to get the best of both worlds by increasing the number of cores in their processors. Some of the cores are performance-oriented while the others are low-power. The LITTLE set of cores is actually a Cortex-A7 which is a simple ARM processor while the big set of cores is actually a Cortex-A15, which executes instructions out of order and has twice the amount of pipeline stages. Every core can be powered down (power-gated) or have its clock modulated to lower the power consumption or increase performance (DVFS [109]). ARM markets this technology under the name big.LITTLE [142].

From an operating system point of view, all the cores of a big.LITTLE processor are the same. However, usual schedulers assume a Symmetric MultiProcessing (SMP) system where all processors are the same. Since a big.LITTLE processor has asymmetric performance and power-consumption characteristics, the scheduler cannot make a good decision which negates the improvement made possible by the hardware. Such a scheduler does not exist yet in Linux but support for it is slowly being implemented [143][144].

Figure 6.6 hints about the power savings achievable for different types of applications when run on the right big.LITTLE core.

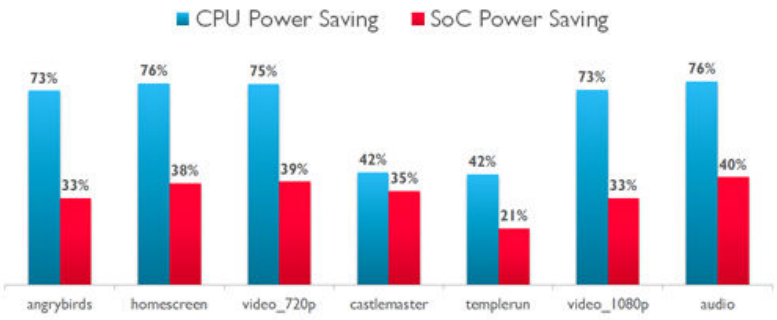


Figure 6.6: Power savings achieved when running applications on the right core. Source [10].

A simple scheduling heuristic could consist in scheduling tasks on the least amount of LITTLE core possible in order to power gate the others. If one task becomes CPU-bound, it should be migrated to a big core in order to improve its performance. This heuristic is based on the assumptions that consolidating tasks consumes less power than spreading them on more cores running at lower frequencies. This seems accurate on desktop or server processors but may not be true for low-power processors with a low static power consumption. In [145], the authors managed to save 10% of power consumption by spreading tasks and keeping the clock low instead of consolidating tasks. An accurate power and performance model in the scheduler is thus needed to make optimal decisions.

A power and performance modelisation of a big.LITTLE board has been carried out in [146] with a static performance level (1GHz) and voltage (1.05 V) for both set of cores. All the cores were power gated except one big and one LITTLE core. The authors found out through their benchmarks that the LITTLE core consumed a constant 1.44 W at full charge while the big core consumed from 4.2 to 5.15 W depending on the application. The performance improvement from the LITTLE core to the big core ranged from 45 to 130%. In their benchmarks, the authors found out that even at full charge, either core could be the most efficient one. However, migrating a task from one core to the other is quite expensive. Migrating from the LITTLE to the big core takes 2.1 ms while the reverse operation takes 3.75 ms. This is why being able to predict the performance and power consumption of a task on a core type based on the current performance and power consumption on another core type could allow the OS to dynamically select the most appropriate core type for the task. This is however not possible in real life yet because tasks are dynamic and there are multiple cores executing multiple tasks at the same time. This could change if ARM develops a hardware power consumption model like Intel's in order to predict its power consumption per task.

In [147], the authors took another approach and tried to select the most efficient processor type in a browsing scenario based on the content of the web pages. They

analyzed 5000 web pages to perform regression modeling of both performance and power consumption. The model was then used to identify the best core type and frequency to minimise power consumption while still loading in less than a certain time. The authors managed to save up 83% of energy in average compared to a performance-oriented policy while increasing by 4.1% the number of websites not loading in the given delay. When compared to Linux's CPUFreq DVFS strategy, it managed to save 8.6% of energy while increasing performance by 4.0%.

Non-Uniform Memory Access

Big.LITTLE may become more complex in the future by becoming a Non-Uniform Memory Access (NUMA) architecture where not all the central memory is accessible at the same speed/latency for every core. In those architectures, a NUMA node is composed of a set of cores and a memory attached to it. An example of a NUMA architecture is visible in Figure 6.7, generated by hwloc [11].

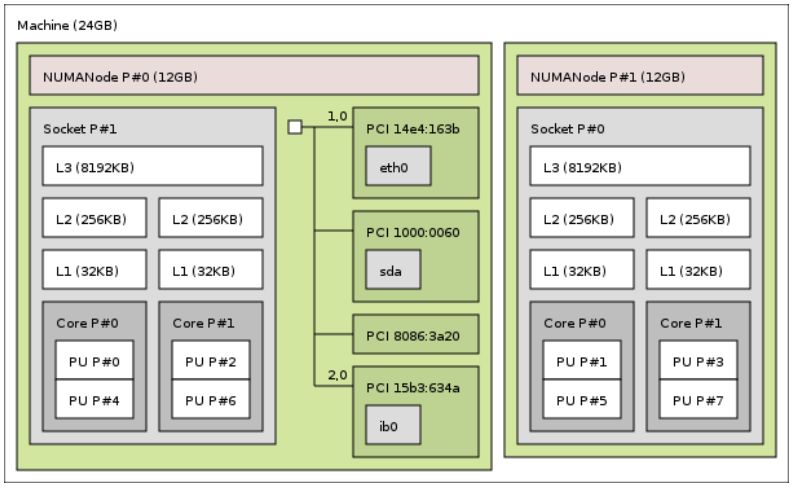


Figure 6.7: A simple NUMA architecture with two sets of 2-cores processors attached to 12GB of RAM each. Generated with hwloc [11].

In such an architecture, a process is most-efficient when all its data and threads are located in the same NUMA node. However, when multiple processes/threads need to access the same data which does not fit into one NUMA node, data and/or threads need to be migrated to lower the access cost and increase performance, as suggested by [148].

Some frameworks such as OpenMP and Intel Threading Building Blocks have been proposed to help developers to create multi-threaded applications in multi-core environments and make them scale well when the number of cores increases. Improvements have been suggested to take into account the NUMA information [149] in those frameworks.

However, these frameworks do not address the problem of having multiple independent applications running concurrently in a situation where all the data and processes can not

fit in one NUMA core. In this situation, the OS should migrate threads and data in order to maximise the usage of the local memory. Currently, the OS can only know which process requested a memory buffer and which processes have access to it, by tracking mmap calls on shared-memory file descriptors. Unfortunately, the OS does not know how often this buffer is accessed because it is the processor's job to walk the page table and get the correspondence between the virtual and the physical address.

Processors manufacturers could however add a usage counter for each page which would get incremented by the Memory Management Unit (MMU) every time a page is accessed. In order to limit the overhead of this method in memory bandwidth usage, the Translation Look-aside Buffer (TLB), which is a hardware cache to improve the performance of page walking, could count the memory accesses to each page it is currently tracking and increment the corresponding value in memory when the page gets evicted. The overhead could be further reduced for both memory and memory bandwidth usage by lowering the granularity of the technique from the page-table (4KiB) to the page directory (4 MiB) one.

Periodically, the OS could then compute the number of accesses per second for each process/thread to each memory page/directory. Using this information, it could identify the performance bottleneck when it comes to memory accesses and move the memory pages that are most-often used closer to their users or move their users closer to the them. To lower the overhead even further, this operation could be done only when performance counters would show a high usage of the interconnection link with the other NUMA nodes.

We did not find any paper talking about the power savings achievable by making placing processes/threads and data on the right NUMA node. However, since accessing memory from another NUMA node is slower than accessing local memory, the performance is not optimal. This means that the program will use more CPU time to execute itself which lowers the time spent by the CPU in sleep mode and most likely increases the average power usage.

NVIDIA's Optimus

In 2010, NVIDIA introduced the Optimus technology in laptop computers. It allows running graphical applications on Intel's integrated GPU (IGP) or NVIDIA's discrete GPU depending on the performance need of the application.

In Optimus's whitepaper [12], it is said that NVIDIA introduced a routing layer for DX calls (used by 3D applications), DXVA calls (used by video playback applications) and CUDA calls (GPGPU application). NVIDIA has a profile for most applications using the GPU, pushed to their clients through a web server they manage. In the application profile NVIDIA stores whether the GPU "can add quality, performance, lower power, or functionality." We can guess that they built a testing farm with multiple Optimus laptops, gathered power consumption and performance metrics for each application and decided which GPU should be used in every supported configuration.

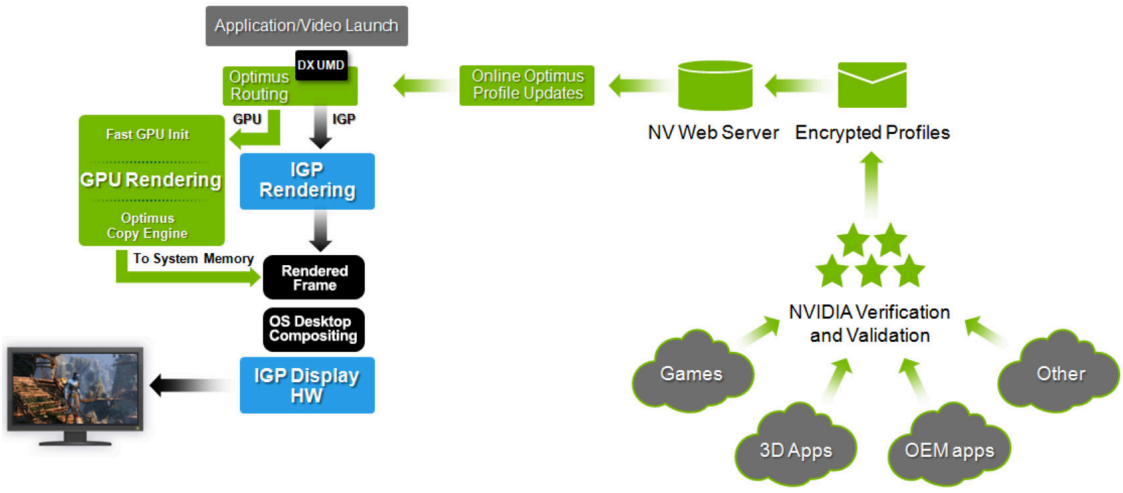


Figure 6.8: NVIDIA Optimus Flow. Source: [12].

If the discrete GPU is chosen, it is first turned on before allowing the driver to configure it for the application. Once rendering is completed in a texture (image), this texture is copied directly from the IGP's memory before being composited (drawn on the screen) by the IGP. For performance reasons, texture are usually stored in a format that best suits the memory they use. These so called "tilling" formats are different from vendors to vendors. The NVIDIA discrete GPU thus had to add support for the tilling format of Intel's IGPs to be able to copy application's output buffers to the IGP's memory without using CPU processing power. NVIDIA named this DMA engine "Optimus Copy Engine". The name given by Nouveau developers is PCOPY. An overview of Optimus for 3D and video playback applications is shown in Figure 6.8.

Our experiment using the Nouveau driver have shown an idle power consumption of 17.334 W when the discrete GPU is powered up and using the Nouveau driver, while it consumes 9.257 W when the discrete GPU is powered off. Disabling the discrete GPU can thus save a substantial amount of power and multiply by two the battery-life of our test laptop (Dell Latitude E6520). This power increase is however higher than it could be because Nouveau does not do any sort of power or clock gating by default. Further experiments are necessary.

StarPU

The research community proposed a runtime called StarPU [150] to express the parallelism of a compute-related application by specifying *codelets* and a dependency graph. Codelets have multiple implementations to fit each architecture it is meant to run on. StarPU will then handle the task dependencies and schedule the codelets on the accelerator the most appropriate processor or one that that is currently not executing anything. It also manages memory and data transfers to get the highest performance by selecting the fastest memory possible for the processor running an instance of the codelet while also limiting the memory transfers that increase latency and use memory bandwidth.

The most appropriate processor for a given codelet is selected by learning the execution time of each codelet is learnt at run time and on every processor/accelerator type because data influences a lot the execution time of a complex codelet [151].

Supported targets for the StarPU runtime include GPUs supporting GPGPU via CUDA and/or OpenCL, Multicore CPUs (Intel x86, AMD64 or PowerPC). It also has experimental support for cell processors such as the one found in the Playstation 3 and is planning on support the new Intel platforms Intel SCC and Intel MIC/Xeon Phi. This runtime is cross-platform and can be executed on Linux, Mac OS X and Windows.

Although the StarPU runtime is doing the right thing, it is difficult for the programmer to deal with multiple programming frameworks and compilers. Right now, CUDA or OpenCL code is loaded from an external file and probably compiled at launch time. StarPU wrapped most of the boilerplate code to make it as easy as possible but it still impedes the readability of the program. To enhance this readability, better compiler support is necessary. Some work is currently underway to reduce code cluttering by adding annotations to the code and let a compiler plugin generate the boilerplate code for us [152]. This effort has however been made for GCC which has no backend for GPUs, contrarily to LLVM. One solution could be to use Clang as a C/C++ frontend, add annotations to specify codelets and let clang generate the LLVM code then the machine code using existing LLVM backends (x86, x86_64, PPC, ARM, NVIDIA, AMD (GPU)). This would allow specifying all the codelets in the same source code and compile it for every target. Codelets could also be re-implemented to better suit to the specificity of each platform. Thanks to the annotations, the StarPU runtime could still identify the different tasks and execute them like they currently do.

A runtime such as StarPU could be enhanced and tuned to provide performance or power efficiency to compute-intensive tasks depending on what the user is doing or his/her performance need. We hope this runtime will become easier to use with time and will be adopted outside of the High Performance Computing (HPC) community.

6.2.3 Dynamic handovers

Since a smartphone/tablet user's activity is dynamic, being able to migrate the network traffic to the most efficient interface and migrating programs on the most efficient processor or accelerator can yield substantial power improvements compared to a static launch-time configuration, as we saw in ARM's big.LITTLE and the NUMA architectures.

It is however not always possible to migrate computing tasks. For instance, Optimus does not seem to be able to migrate applications between the discrete GPU and the IGP. This is because Intel's and NVIDIA's internal architecture is different. It is thus not really possible to pause an application on one GPU, copy its context on the other GPU, reload the context and continue the execution. Even if it was possible, it would still require copying hundreds of megabytes of textures from one GPU to the other and recompile all the shaders (graphics programs running on the GPU). This would result

in a context switch lasting seconds instead of microseconds. This explains why NVIDIA preferred testing applications themselves and create profile for them instead of launching them on the wrong GPU which would either result in poor performance or high power consumption.

StarPU solves the migration problem by segmenting the problem into chunks that take a relatively short time to execute. It then wait for a processor or an accelerator to become available before scheduling another chunk on it. This is suitable to perform scientific computation but not really for games or usual applications that are not compute-oriented.

Before MultiPath TCP (MPTCP) [153], it was also not possible to seamlessly support data connection handovers without the external network infrastructure supporting it. This means that switching network interfaces would close all the TCP connections. MPTCP is very promising for multihomed devices as it allows seamless handovers of TCP connections if both the client and the server support it. This will enable selecting the most power-efficient network interface for the current usage of the user without needing a proxy. The MPTCP protocol is however not final yet, but it is implemented nonetheless by Apple for Siri in iOS 7 [154] or in an experimental tree of Linux.

Power and performance decisions are not easy to make because of the network handover and processor migration costs. It may indeed be more power-efficient to stick to a less instantly-efficient mode that would avoid a ping-pong effect which would be both damaging to the user experience and to the battery life. The decision also needs to take into account the current state of each network interface.

Numerous articles are found in the literature about autonomic network selection. However, they all use an application- and hardware-specific decision making module that needs to be re-implemented when a new radio is being used because accurate models are heavily hardware-, driver- and protocol-dependent. Indeed, our generic power and performance model [105] for NVIDIA GPUs was ten times less precise than a card-specific one [124]. That is also the approach taken by Intel for RAPL [130]. It is thus very important to separate these models while still being able to compare their decisions. That is why we are proposing a generic flowgraph for run-time and performance management.

6.2.4 Run-time power and performance management generic flowgraph

Decision engines are usually used in the literature for network interface selection in multihomed mobile devices, consolidating workloads in data centres [155] or making dynamic power/performance trade-offs in CPUs/GPUs through dynamic voltage-frequency switching [109].

Most articles on network selection in a multihomed environment usually aim at selecting the interface that will best suit the QoS. Some decision models are based on user-defined rules [156] or policy-based priority queues [157]. Other articles on the same

topic incorporate more metrics in the decision process such as the available bandwidth or the radio signal strength [158] while some also include power considerations [159][160].

A generic mathematical framework is also proposed [161] which takes into account the current signal strength, the cost of using the network access technology, and the client power consumed for the particular access technology. This model does not take into account the different states of the radio or the type of traffic generated by the applications. Simulations were carried out with MATLAB but no evaluation on the impact on CPU and RAM have been carried out.

In [162], the authors proposed to use Genetic Algorithms [163] to find the network interface that will minimise both the average power consumption and the bandwidth dissatisfaction. Applications seem to be required to know their current bandwidth needs while the power consumption of the radio is modelled as being linear with its bandwidth usage which does not take into account that radios cannot be put to sleep straight away after emitting data. Depending on the number of generations, the execution time varies from about 50ms (10 generations) to 10s (2000 generations).

PISA [164] proposed to separate the application-specific part of an optimisation engine from the non-application-specific part and defined interfaces for them to communicate. This modularisation allows re-using the optimisation algorithms and share them with other researchers. PISA is language-independent which makes it potentially suitable for real-time applications, depending on the algorithms used. It is however not a framework and thus lacks debugging helpers.

We are not aware about any previous work taking into account the real-time and low performance-impact requirements for performance and power management. The real-time requirement is important because decisions must be taken at a fast and constant rate to react to changes in the user's activity and its surrounding. The low performance impact requirement matters because the resources taken by the decision engine cannot be used to produce useful work for the end user and increase the power consumption, which decreases the potential power savings made by the decision engine.

Decision engines found in the literature are usually written in high-level languages such as MATLAB or linear-optimisation frameworks that are hard to bound in execution time and RAM usage. These languages usually have a high performance impact on the CPU and RAM usage compared to non-managed languages such as C. Genetic algorithms have also been used and have, by nature, no execution time boundaries since they should never converge. The best solution can however be selected at any generation which makes it suitable for real-time requirements. To make the impact on CPU and memory acceptable, a slow decision rate is mandatory which can be acceptable in some scenarios. For instance, spending one second of computation time every 20 minutes would produce an average impact on the CPU of less than 0.1%.

In this chapter, we propose a flowgraph for generic and run-time decision making that follows the modularisation approach proposed by PISA for multi-variable optimisation. This flowgraph has been implemented in a framework written in C which gets the best potential for performance and efficiency. This framework focuses on allowing real-time decisions for power and performance management while remaining generic-enough to not constraint the decision-engine programmer. It also eases the development of a decision engine thanks to its already-existing modules for the different stages and thanks to its advanced debugging capabilities.

6.3 Definition of an abstract decision flowgraph

In this section, we propose the definition of an abstract and generic decision flowgraph that is compatible with real-time requirements.

A decision flowgraph is needed when there are more than one independent way to achieve the same goal, yet, they do not impact the power consumption and performance in the same way. For instance, in the case of a multihomed mobile device, we could want to take a decision whether to use the WiFi network interface or the cellular one. Taking a decision requires a performance and power model (HW model) of both interfaces. With these models, it should be possible to find the optimal configuration to balance the needed performance with power consumption at any given time.

The hardware (HW) models are supplied with the user's performance/QoS needs continuously so as it can self-optimise. Instead of feeding models with the immediate QoS constraints to the models, we propose to feed them with a prediction of what will be the evolution of the constraints and their expected variance.

Once a HW model outputs a decision along with its impact on performance and power consumption, they should be compared with each others. In order to do so in an abstract way, we propose that a score on every output metric of every HW model should be generated and then aggregated into one score per HW model. All these information are gathered in the same data structure and sent to the decision-making part that will decide which HW model should be selected.

Our abstract decision flowgraph is composed of 5 stages:

- Metrics: Collecting metrics / performance needs;
- Prediction: Predicting the evolution of the metrics;
- HW models: Proposing the best decisions possible;
- Scoring: Scoring the output of the local decisions;
- Decision: Selecting the best HW model.

This decision engine flowgraph is considered generic because it does not force using a single model with different parameters, unlike other decision engines in the literature. This decision making model improves interchangeability and real-time-worthiness of a decision engine. An overview of the abstract decision flowgraph is presented in Figure 6.9. Each stage will be discussed in the following subsections.

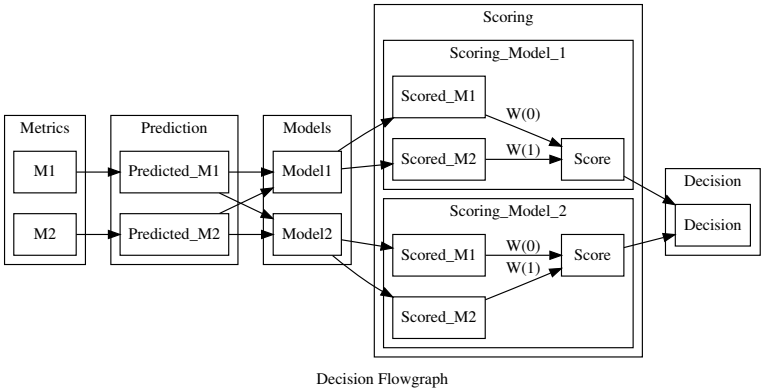


Figure 6.9: Overview of the abstract decision flowgraph

6.3.1 Decision flowgraph

A flowgraph represents a decision process. It includes the 5 stages discussed before and holds properties like the decision refresh rate. It can also run along-side other decision flowgraphs to, for example, allow an application to select the most efficient network interface while selecting the right performance level on the CPU.

6.3.2 Metrics

The role of the input metrics is to describe the current state of the system, including the performance requirements. They are not limited in numbers and should be updated periodically at a fast-enough rate to represent the current state and its variability. Poorly-chosen metrics will produce poor predictions at the next stage which will result in a poor decision quality. Examples of metrics could be the incoming and outgoing packets in the network stack or the CPU usage.

6.3.3 Prediction

The objective of the prediction stage is to supply the expected evolution of the system's state for a given time span. The needed time span depends on how often a decision is taken and how long it takes for a decision to be applied.

Prediction models may have multiple inputs of different types and generate multiple predictions of different types. For instance, a prediction model could take as an input the network packets received and predict the evolution of the reception throughput of the network interface.

Predictions act as constraints and can be metrics-based (run-time information on the state of the system) or static constraints. Static constraints can be used to express power or performance constraints such as latency that the HW model should take into account during its decision making.

The output of prediction models is composed of three graphs per prediction. The first graph represents the expected evolution of the predicted metric while the other two respectively represent the lowest and highest expected outputs for the metrics. These three graphs allow representing both the average and the expected variance at any point of the time span of the prediction.

Using graphs over mathematical formulas allows for a much more precise and real-time-friendly intermediate representation for predicted metrics. First of all, interpreting mathematical formulas at run time can be very CPU-intensive and linear optimisation has execution time that is hard to bound. Moreover, graphs are much easier to generate, read and score for a computer. They also represent the real expected evolution and, to some degree, the density of probability at any point of the prediction.

Some prediction models are proposed in the next section.

6.3.4 HW Models

HW models of a flowgraph should all describe an independent way to carry out the task that is needed by the user. They are user-provided, take the predicted metrics as an input and output the expected impact on the predicted metrics/constraints. Every output metrics of the models should be in the predicted metrics/constraints otherwise it is impossible to score it.

HW models have to be independent so that selecting one does not change how others calculate as this would require HW models to be aware of each others and this would break the modularity of the flowgraph.

In the case of network interface selection, there should be one HW model per network interface. All HW models would take the predicted network throughput along with the RF occupancy and latency constraints. HW models should then try to produce an optimal solution by selecting the right performance mode at the right time to have the lowest power consumption possible while still meeting the throughput requirement along with the latency and RF occupancy constraints. The HW models would then output graphs telling the evolution of the expected power consumption, RF occupancy and network latency in order to be scored.

The reason why there should be one HW model per interface and not per technology is because one network interface may support different technologies, such as the WiFi adapter also supporting Bluetooth, and they would not be independent.

6.3.5 Scoring

The scoring stage is meant to score the different HW models, before sending the result to the decision engine. The HW model's score is bound in $[0, 1]$ and is calculated based on the score of sub-metrics. Every sub-metric is assigned a weight that defines the importance of each metric ($W(0)$ and $W(1)$ in Figure 6.9). These weights need to be defined by the application. Scoring blocks can be user- or framework-provided.

6.3.6 Decision

The final stage of the decision process is selecting the most appropriate HW model and switching to it. The decision stage receives the scores of every HW model and metric, the HW models' output and the predictions. It is then responsible for evaluating the transition costs on performance and power consumption so as to avoid ping-pong effects that would be damaging to both the battery life and the QoS.

6.4 Implementing the decision flowgraph in a framework

In Section 6.3, we proposed a flowgraph for a generic run-time decision engine. We have implemented this flowgraph into a framework in order to test it and make writing decision engines easier. We named this framework "Real-Time Generic Decision Engine" (RTGDE). Its implementation is available publicly [165].

This framework should not impede the implementation of a real-time decision process due to the framework's pipeline nature and the fact it barely does any processing at all except passing information from stage to stage. It should thus be possible to create a soft real-time decision engine as long as the predictions, HW models, scoring and decision algorithms used can have their execution time bound.

In this section, we detail the most important design constraints we had at the different stages of the decision pipeline.

6.4.1 Metrics

Run-time metrics collection and decision suggests multiple threads must be used to separate both processes. A cross-threads synchronisation is thus needed to allow the producer (metric collection) thread to send information (metrics values) to the consumer (prediction) thread. The use of a ring buffer, along with the usage of a mutex per metric, solves this communication problem. It is shown on Figure 6.10.

6.4.2 Predictions

Predictions take the metrics attached to them and output three graphs which are respectively the average, lowest and highest prediction over time. Graphs are used instead of formulas to ease development and make it easier to reach real-time guarantees.

The RTGDE framework aims at shipping multiple generic prediction engines. Currently, we have implemented the following ones:

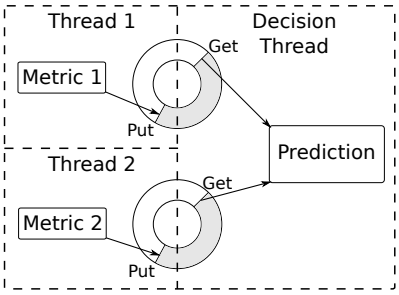
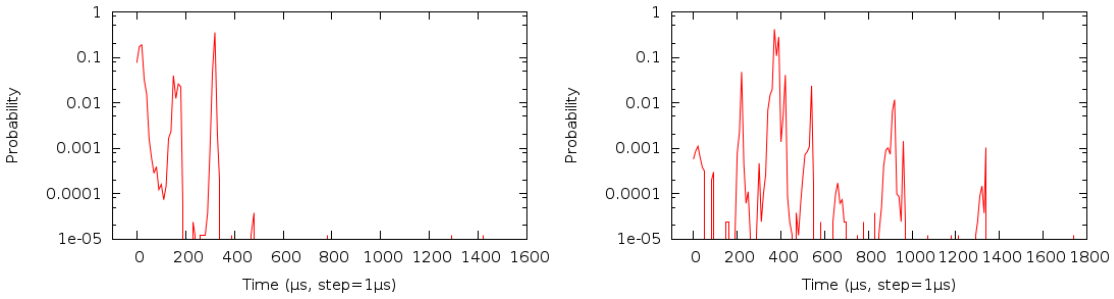


Figure 6.10: Using ring buffers to implement asynchronous communication between metrics collection and metrics prediction

- Simple: Equivalent to no prediction at all, it just outputs the latest value of the metrics;
- Average: Computes the average and the variance of the metric, outputs the average, the lowest and the highest predictions based on a confidence level and the variance if the metrics follows a Gaussian;
- FSM: Considers the metric follows a Finite State Machine (FSM) with associated properties such as power consumption or throughput. The FSM prediction learns the density of probability to transit from one state of the FSM to an other and uses this knowledge to output the expected evolution of the metric. This prediction technique works for periodic signals but requires a very high sample rate at the metrics.

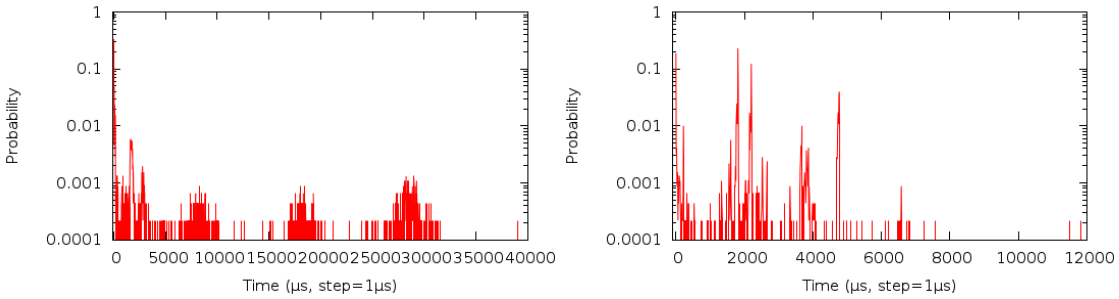
To evaluate the need for an FSM prediction, we looked at GPU usage in different scenarios. We proposed two states, the GPU would either be active or idle. To get the GPU's state, we continuously polled at 3.8 MHz (as fast the PCIe port would let us poll) on NVIDIA's status register for the main engine (PGRAPH.STATUS, address 0x400700, bit 0).



(a) IDLE ↦ ACTIVE transition **(b)** ACTIVE ↦ IDLE transition

Figure 6.11: GPU Usage: Probability density of transitioning from one state to another while running a highly predicable GPU load (glxgears)

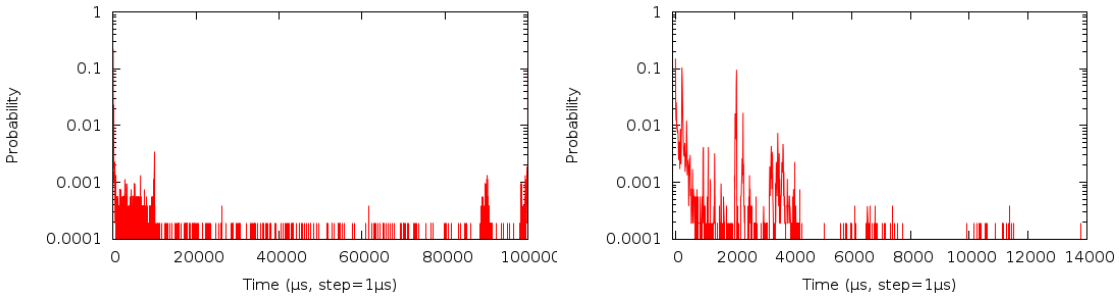
In the first scenario, shown in Figure 6.11, we ran a very simple and predicable 3D application called glxgears. We can see several spikes in the density of the switching probability from one state to another. This denotes a high regularity in the switching period and means the FSM prediction method is appropriate for such an input metric and application.



(a) IDLE ↦ ACTIVE transition **(b)** ACTIVE ↦ IDLE transition

Figure 6.12: GPU Usage: Probability density of transitioning from one state to another in a web browsing scenario

In a second scenario, shown in Figure 6.12, we analysed the state transitions when watching a youtube video in 360p. The ACTIVE ↦ IDLE transition's density of probability have more diracs indicating a predicable rendering time. However, contrarily to Figure 6.11, the IDLE ↦ ACTIVE transition's density of probability is more diffused but shows 6 different bumps. This may suggest that either there are several periodical events triggering GPU usage with their update frequency not being harmonics and in phase, or there is a periodical event (rendering frames at 25 FPS) along with one or multiple a-periodical events.



(a) IDLE ↦ ACTIVE transition **(b)** ACTIVE ↦ IDLE transition

Figure 6.13: GPU Usage: Probability density of transitioning from one state to another in a web browsing scenario

Finally, we repeated the experiment in a web browsing scenario, shown in Figure 6.13. Unsurprisingly, given the aperiodical nature of the GPU load when web browsing, the

IDLE $\mapsto$ ACTIVE transition's density of probability is very spread. The GPU's processing time was also more spread than in previous figures.

As can be seen from the previous figures, the FSM prediction only works when monitoring a single process that can be modelled by a finite state machine, like in Figure 6.11. When multiple processes are running concurrently, the process with the shortest period creates aliasing. If the signals are not in phase, they create what seems like white noise in the transition's density of probability which makes any prediction useless. The implementation of this prediction technique is however not finished but the results are encouraging-enough to make it worth finishing up the implementation.

Since every application has different needs, it may not be accurate to use generic prediction models. It is thus important to allow for user- along with framework-supplied prediction models. We decided to make it as easy as possible to do so. This means the internal interface should be exposed to the application in order to interface with it.

Constraints are considered as static predictions and define how to score the HW model's output metrics that are linked to it. For instance, the power consumption constraint of a model can be set so as the maximum score is achieved when the power consumption is 0 mW, the minimum score when drawing 3 W and the average score when consuming 1 W. Score is attributed linearly between these points.

Applications may require the prediction to be dumped along with the metrics data that was used to generate it. Predictions are then dumped as a Comma-Separated Values (CSV) file and a user-defined callback is called to allow the application to process this data. Figure 6.14 was generated using gnuplot [166], called from the user-defined callback of the pcap example located at '/tests/pcap/' in [165].

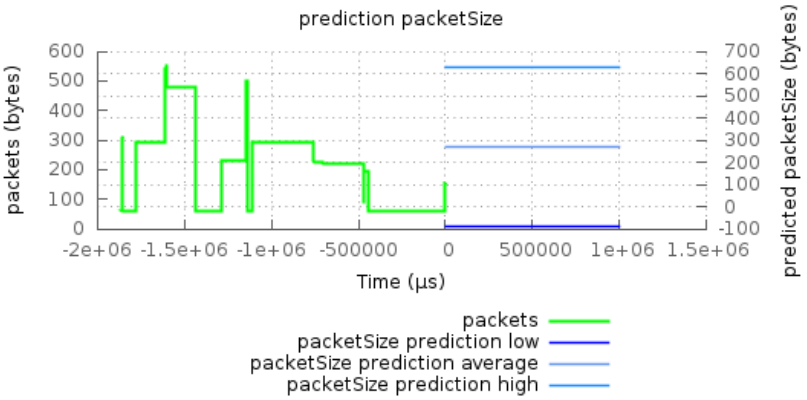


Figure 6.14: Example of the output of the "average" prediction method

In this Figure, each point of "Packets" represents a received packet (size and time of arrival). The packetSize prediction low/average/high represent the output of the prediction stage concerning the packet size. In this example, it is not expected to change during the prediction period so the lines are flat.

6.4.3 Flowgraph

Prediction and HW models are attached to a flowgraph. Whenever the flowgraph is run, all the predictions algorithms generate the inputs for the HW models which are then called serially with a copy of the input metrics. They could however be run in parallel on multi-core processors to lower the execution time of the flowgraph.

6.4.4 Scoring

There is currently only one scoring algorithm implemented in the framework. The equation used to compute the HW model's score is given in Equ. 6.1 where $S_m(i)$ is the calculated score of the metric i while $W(i)$ is the weight given to the metric i and n is the number of metrics to be scored.

$$Score = \frac{\sum_{i=0}^{n} S_m(i) * W(i)}{\sum_{i=0}^{n} W(i)} \tag{6.1}$$

The function $S_m(i)$ is calculated by integrating the probability that the proposed result on the metric i is lower (or higher, depending on the metric) than its prediction at time t ($p_i(t)$), over time 0 to the prediction time-span (T).

$$S_m(i) = \frac{\int_{t=0}^{T} p_i(t).dt}{T} \tag{6.2}$$

Like for predictions, an application may require the scoring output to be dumped to a CSV file before calling a user-defined callback. Figure 6.15 is an example of the output of the default scoring method on the power consumption metric.

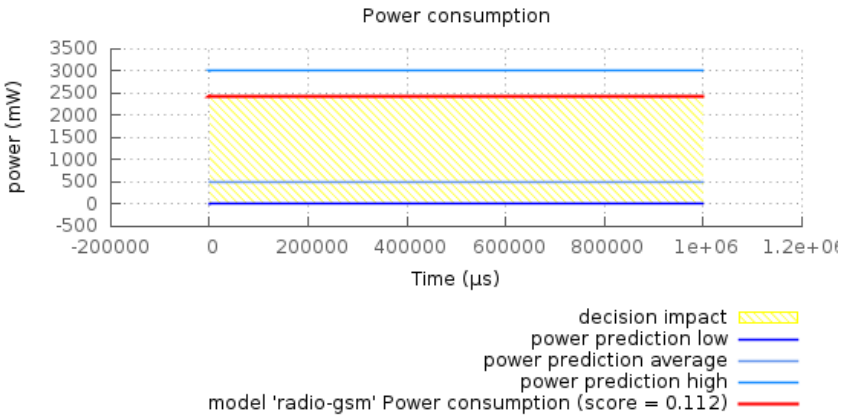


Figure 6.15: Example of the output of the default scoring method

This figure is akin to Figure 6.14, except it is a power consumption constraint instead of a packet size prediction. We can also find an additional line (red) which represents the expected power consumption of the "radio-gsm" HW model. Finally, the decision impact, shown in light yellow, represents the difference between the ideal power consumption (power prediction low) and the expected power consumption.

6.4.5 Decision

We provide as much information as possible to the decision algorithm. This includes predictions and HW models outputs along with their scores.

Once a decision is taken, a user-defined callback is called with the result of the decision and all the information that was used to take it. As this call is currently made from the flowgraph thread, the application should not block.

6.5 Evaluation

In this section, we evaluate the genericity of the flowgraph and our framework by implementing two common use-cases for decision engines in green networking. We then evaluate the performance of the framework.

6.5.1 Network selection WiFi / GSM

As most of the decision engines in the literature are used for selecting network interfaces based on QoS and/or power efficiency, we decided to evaluate the framework by implementing a simple WiFi / GSM network interface selection.

The throughput requirement and expected packet count are predicted in real-time by sniffing the network traffic in the network stack, using libpcap [167] (the library used by tcpdump and Wireshark). In this experiment, the network activity was generated by browsing the web. The following constraints are set (Highest/Median/Lowest score):

- Power consumption: (0, 500, 3000 mW);
- RF spectrum occupancy: (0, 50, 100%);
- Latency: (0, 5, 10 μs).

For every decision, 2 predictions (Packet Size and Packet count) and 3 constraints (Power consumption, Radio Frequency (RF) occupancy and network latency) are generated. Both HW models then generate 3 outputs each for every decision. Given an update rate of 1 Hz, we get 11 CSV file every second. It is thus not practical to include all of them in this thesis. Examples of outputs would be Figure 6.14 (output of the prediction for the "packetSize" metric) or Figure 6.15 (output of the scoring block for the "power consumption" constraint).

The WiFi and GSM radios are using the same HW model but with different parameters. The parameters are the name of the radio, its bitrate, the energy cost and delay to switch from reception (RX) to transmission (TX) and TX to RX and then, the power consumption when receiving and when transmitting. Those parameters are very simple and are not accurate as they do not take into account GSM and WiFi link quality. They are however sufficient to prove the genericity of the decision flowgraph and the framework as we are only demonstrating how the framework can be used and not evaluating the decisions taken by the engine.

The scoring technique is the one proposed earlier with the metric "Power consumption" (Figure 6.16) given a weight of 20, "Emission latency" (Figure 6.17) given a weight of 5 and "RF occupancy" (Figure 6.18) given a weight of 3. These weights should be defined according to the user's strategy and motivations. It is out of the scope of this thesis.

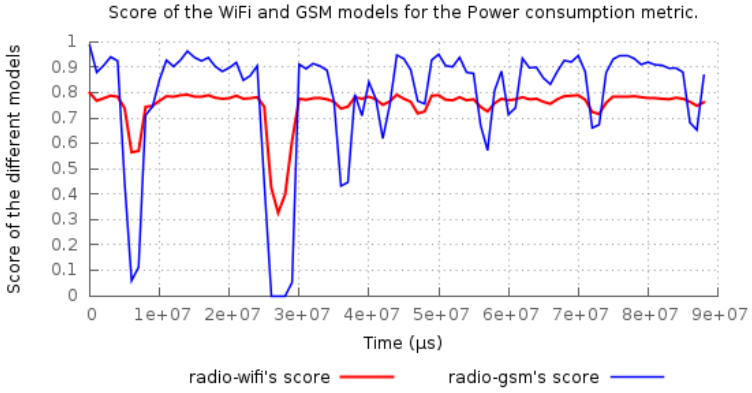


Figure 6.16: Evolution of the power consumption score of the two HW models

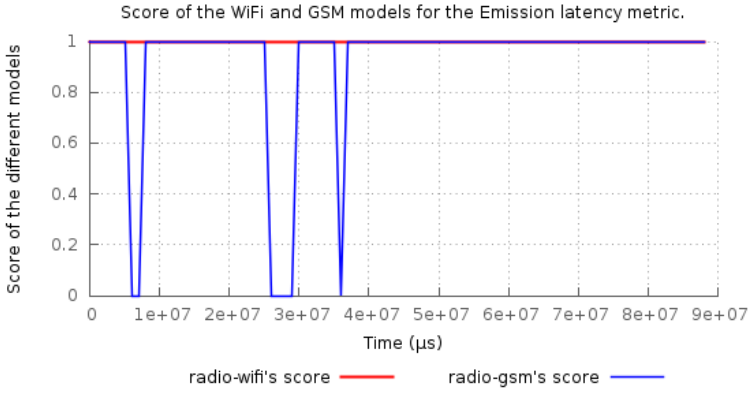


Figure 6.17: Evolution of the emission latency score of the two HW models

The decision to use the GSM model over the WiFi model is given if the following conditions are fulfilled:

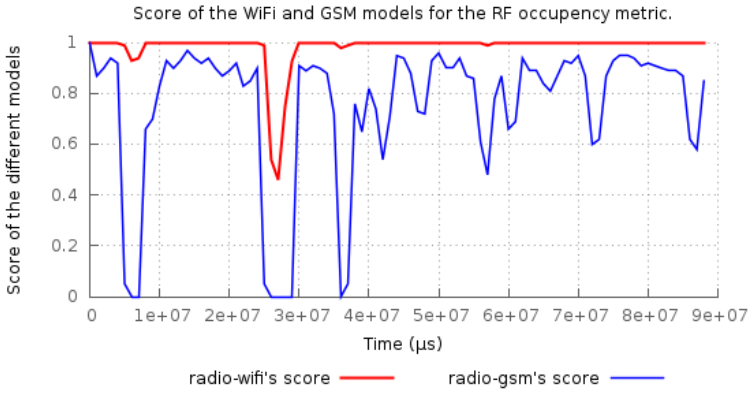


Figure 6.18: Evolution of the RF occupancy score of the two HW models

- The average score of the GSM model in the last 30 seconds is higher than the WiFi model's score;
- The emission-latency average score of the GSM model in the last 30 seconds is higher than 0.9. This means the GSM's throughput is sufficient for the current load;
- The latest network interface switch happened more than 10 seconds ago.

As stated before, this will not yield a good decision but it is sufficient to prove the flexibility of the approach. The decision policy should be set according to the user's strategy. The final score of both HW models along with the decision result is shown in Figure 6.19.

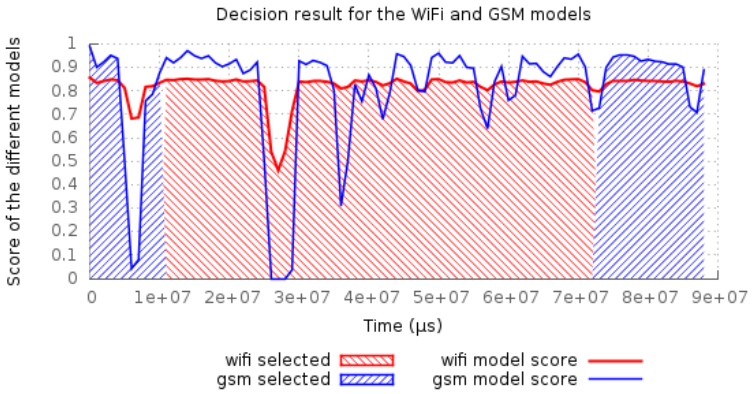


Figure 6.19: Evolution of the score of the two HW models along with the decision

All these graphs have been generated at run-time using the logging and callback capabilities of the framework.

In this evaluation, we proved it is possible to implement a network interface selection at run-time using our proposed generic flowgraph.

6.5.2 Selecting performance levels and processors

The second evaluation of the framework we performed was CPU performance level and core selection on a smartphone equipped with a big.LITTLE processor [143], where half the cores are optimised for speed (BIG) and the other half is optimised for power consumption (LITTLE). The decision engine's role is to select which set of cores and performance level should be used for the applications currently running on the smartphone. It's input metric is the CPU usage which gets predicted using the "average" prediction system. as illustrated by Figure 6.20.

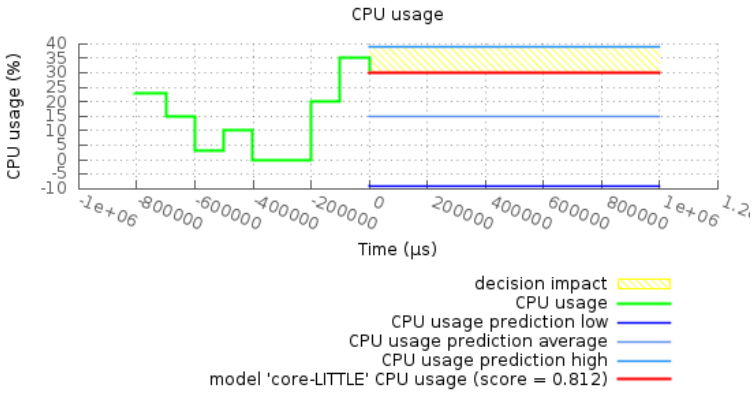


Figure 6.20: Evolution of the CPU usage metric and one prediction

In this figure, we can see in green the evolution of the CPU usage in the last 800 ms. We can then see the result of the prediction stage with the low, average and high predictions shown in the different shades of blue. The result of the model is shown in red and the impact on the impact on the metric is shown in yellow. The score of the model for this metric is 0.812.

Each set of cores (BIG or LITTLE) has 3 different performance levels. A performance level is defined by:

- Static power consumption: idle power consumption;
- Dynamic power consumption: power consumption added when 100% active;
- Performance: Maximum performance relative to the fastest performance level of the BIG core.

The LITTLE cores have the following performance levels:

- 0: Static = 10 mW, Dyn. = 200 mW, Perf. = 0.1;

- 1: Static = 15 mW, Dyn. = 300 mW, Perf. = 0.2;
- 2: Static = 25 mW, Dyn. = 500 mW, Perf. = 0.3.

The BIG cores have the following performance levels:

- 0: Static = 100 mW, Dyn. = 800 mW, Perf. = 0.25;
- 1: Static = 125 mW, Dyn. = 900 mW, Perf. = 0.63;
- 2: Static = 150 mW, Dyn. = 1000 mW, Perf. = 1.0.

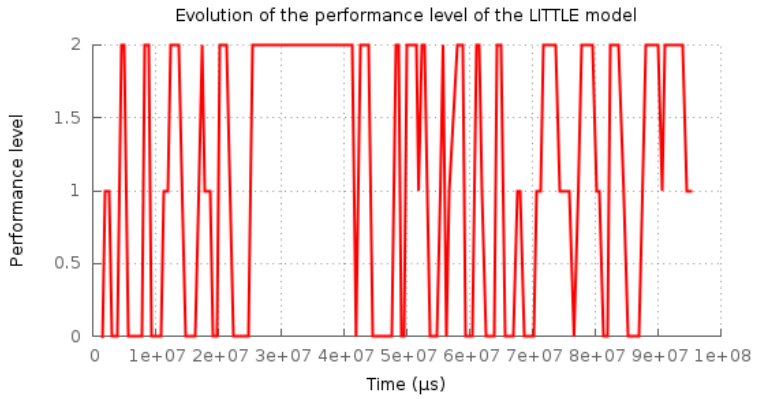


Figure 6.21: Evolution of the performance level used by the LITTLE model

We propose to use two HW models, one for the BIG set of cores and one for the LITTLE set of cores. BIG and LITTLE cores are handled using a single HW model with different parameters (name and performance levels). Performance levels are selected by both HW models while the set of cores to be used is selected by the decision process. In Figure 6.21, we can see the selected performance levels for the LITTLE model across the experiment with 0 being the slowest performance level, 1 the median one and 2 the fastest one. Figures 6.22 and 6.23 respectively represent the evolution of the score for the performance impact / CPU usage and the power consumption score. The power consumption score comes from a static constraint set to 5/125/1000 mW (low, median, high).

The power consumption constraint is given a scoring weight of 10 while the performance impact score is given a weight of 13. The weights need to be adjusted according to the user's strategy.

The decision policy is entirely based on the score of the two HW models. If the current model is LITTLE and if the score of the BIG model is 20% higher than LITTLE's score then the BIG model is selected. On the other hand, if the current model is BIG and the LITTLE's score has been 10% over the BIG's 5 times without the LITTLE's score

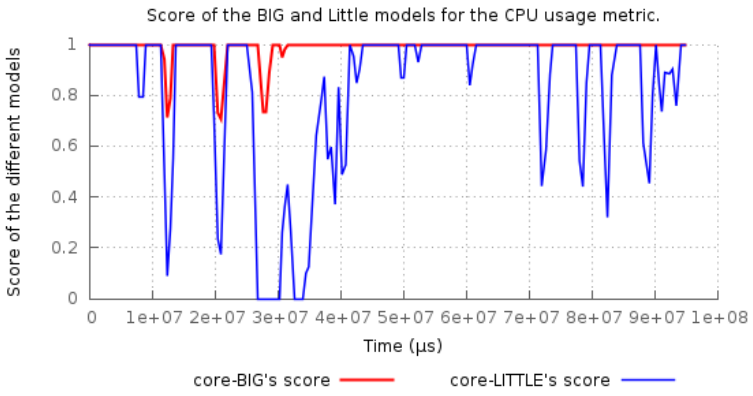


Figure 6.22: Evolution of the CPU usage score for the two HW models

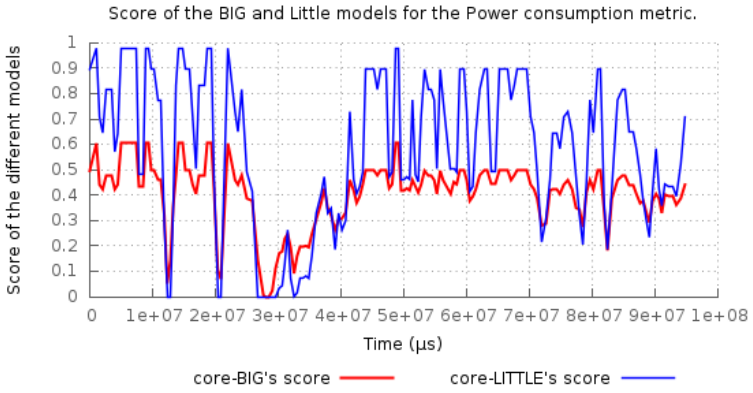


Figure 6.23: Evolution of the power consumption score for the two HW models

dropping lower than 20% of the BIG's score, then the LITTLE model is selected. A decision is taken twice per second which is slow enough for performance- and power-cost of the migration to be negligible. Figure 6.24 presents the evolution of the score of the two HW models along with the proposed decision.

6.5.3 Performance evaluation

The previous experiment, without CSV outputs and graphs generation, used approximately 296 kB of RAM on our system (Arch Linux x86_64) with a further 796 kB of libraries shared with other programs (libc, libglib, ld, libm, libpthread, libpcre, libXau). The actual memory footprint of RTGDE library is however just 36 kB and the volume of data created by the framework for this experiment is lower than 56 kB. Most of the remaining memory footprint of this experiment comes from the libgtop that was used to collect the CPU usage metrics, the libc and the stack allocation for the two threads of the process.

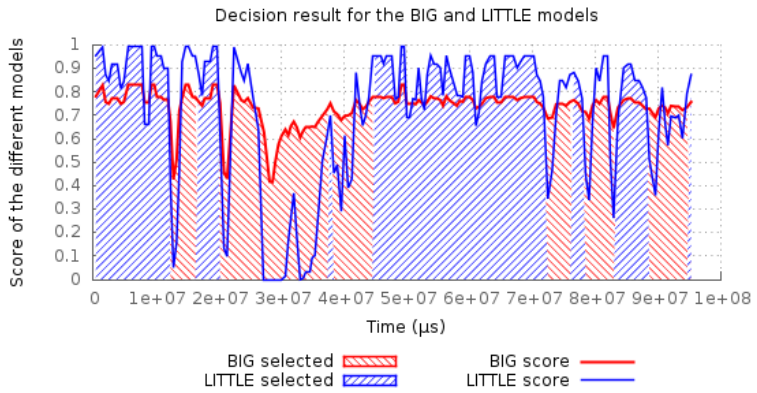


Figure 6.24: Evolution of the score of the two HW models along with the decision

We also experimented with the CPU usage impact of the RTGDE framework in this evaluation and found that we had to increase the decision rate from 2 to 100 Hz to produce a visible impact on CPU performance of 2% on one core of our i7 860. One iteration of the decision process takes 43.95 μs in average with a standard deviation of 13.19 μs.

We decided to compare the memory usage and real-time worthiness of our proposition compared to MATLAB as it has often been used by researchers to implement decision engines [161][160]. We compared our proposition on Linux using GNU Octave which is an open source alternative compatible with MATLAB. To compare our proposition and an implementation made in Octave, we wrote the minimum application possible that executes code one thousand times per second in our proposition and in Octave, respectively found in [165] under the name "test _minimal.c" and "matlab_min.m".

To test the real-time worthiness of the two solutions, we compared the delay between the moment when the application requested to resume execution, called scheduling jitter, and the standard deviation of this jitter over 30 seconds when our testing Linux system was idle. The results highly depend on the CPU scheduler but also reflect any additional overhead the language is putting on real-time constraints. They are visible in Figure 6.25 where we can see that Octave's jitter is almost 50% higher than the one found in the C language used to implement our framework. Octave's standard deviation in the scheduling delay is more than 60% higher than C's.

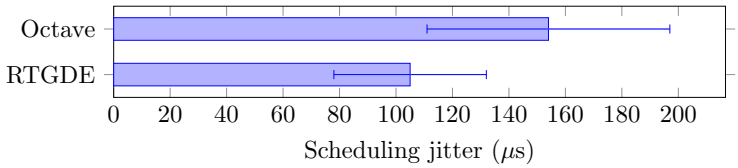


Figure 6.25: Jitter in the scheduling of decision process on Linux. Data collected during 30 seconds when the system was idle.

In the second test, we measure the memory usage of both solutions and split it in two categories. The "shared" memory usage comes from the different libraries used by both solutions, named this way because they can be shared in memory with other processes. On the contrary, the "private" memory usage is memory which is exclusively used by the program. The results of this test can be found in Figure 6.26 where we can see that Octave's private memory usage is 25.3MB along with 7.4MB worth of libraries that could be shared with other processes. On the other hand, our proposition has a private memory footprint more than 150 times lower (168kB) than Octave's while its shared memory footprint is more than 15 times lower (1.4MB).

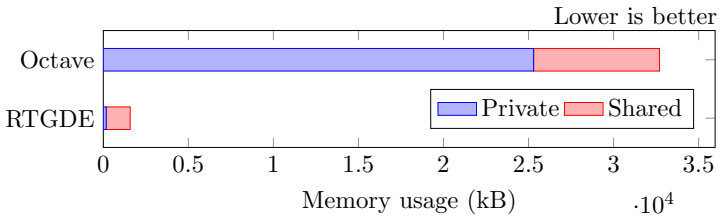


Figure 6.26: Minimum memory usage when using RTGDE or Octave to implement the decision engine

A comparative performance evaluation of MATLAB and C++ was carried out for plot generation and it was found that C++ was faster and more versatile [168]. C being a sub-set of C++, it should share the same characteristics.

Through this evaluation, we demonstrated that it is possible to implement a complex performance level selection at run-time on an heterogeneous CPU architecture by using our proposed flowgraph and framework. We also demonstrated that the framework's CPU and memory footprint can be negligible, even for mobile devices, contrarily to solutions found in the state of the art which are mostly based on MATLAB.

6.6 Conclusion and future work

In this chapter, we motivated the need for an OS-driven power and performance management system, even when using autonomic components, as defined in Chapter 5. The role of this OS-driven power management system is to model the power and performance of the processors, accelerators and network interfaces found in the computer in order to fulfil the user's task as efficiently as possible while still meeting the wanted performance. Energy savings of up to 80% can be achieved by selecting the right processor type in a big.LITTLE scenario or by selecting the right network interface in a typical smartphone with a WiFi, GSM and 3G interface.

The most accurate power and performance models for network interfaces and processors are very dependant on the hardware they model. In order to compare them, their inputs and outputs need to be standardised, as proposed by PISA [164]. However, PISA has not been designed for run-time decision making. It can be used to implement efficient power

and performance models but it lacks different stages that allow for decision making in real-time such as collecting metrics in different threads.

In this chapter, we proposed a generic flowgraph for run-time decision making that allows code re-usability by specifying different stages and their interfaces. It also allows creating decision engines using many different power and performance models as long as they share the same output interface. We then implemented this generic flowgraph into a framework and evaluated it by successfully implementing run-time algorithms for network interface and CPU type selection in a big.LITTLE environment.

On Linux, the RTGDE framework we proposed requires less than 58kB of storage memory, uses less than 100 kB of RAM for typical flowgraphs and has virtually no impact on CPU performance when not generating gnuplot graphs in real time. This is to be compared with other solutions used in the state of the art such as MATLAB that can use up to 150 times as much RAM as our proposition. The low CPU and memory requirement makes this decision flowgraph and framework very suitable for mobile devices. The implementation of this framework has been achieved in about 4000 lines of C code with only one mandatory dependency to pthread and is available publicly [165] under the Open Source MIT license.

Such a framework could be used to write an efficient power management daemon that would tell the Operating System's kernel which network interface and which set of core should be used at any time. Power and performance models could be added to the daemon by hardware manufacturers and power efficiency hobbyists thanks to its open nature. The daemon could then probe which hardware is available, load the right modules, collect the right metrics and select the most efficient way of performing the user's task.

This flowgraph is meant for system-level power and performance management such as migrating all applications to the same cores or selecting the most appropriate network interface for the whole system. This flowgraph is not suitable for scheduling multiple tasks onto multiple resources. Such scheduling is currently left to the Operating System and will be investigated in future work.

Other future work include adding more generic prediction algorithms, evaluating the flowgraph on more scenarios and writing tools to help generating the code for setting up the prediction engine. Thanks to the modular nature of the framework, a website could also be written to allow researchers to share their prediction algorithms or HW models for radios, CPUs or GPUs to allow direct comparisons between models or scoring systems in research or to limit the amount of code that needs to be written to set-up a prediction engine. Currently, researchers have to implement HW models for every hardware used in their decision-making. Predictions could also be scored by RTGDE and be used to automatically select the most-accurate prediction system. The score could also help debugging poor prediction issues.

Chapter 7

Proposing a green network for disaster relief

Contents

In this chapter, we discuss how we can combine the different contributions we described in this thesis to create an energy-efficient heterogeneous wireless network to help a rescue team to organise itself in a disaster relief scenario to save civilians.

7.1 Defining the scenario

In a disaster scenario such as the one that happened in New Orleans after the hurricane Katrina (2005), most of the infrastructure is not usable anymore. The roads are blocked and the electricity, water and telecommunication infrastructure are gone.

Upon arrival to the disaster site, the rescue team first requires an aerial view of the the area to identify roads that need to be cleared to facilitate the access to buildings with potential survivors who need to be rescued. The aerial view also needs to be matched to preexisting maps and should be made available to most members of the rescue team. These maps are meant to be annotated by the rescuers and updated in real time. Example of annotations could be the level at which a building has been inspected, the number of

survivors trapped or the potential hazards found in different areas (explosives, chemical leaks, others).

The rescuers' terminal used to access shared data should be a tablet because of its big screen that allows discussing with multiple people and because of its versatility. It should also allow rescuers to text, voice or video chat with each others. It should also be available at all time for a minimum of 12 hours and be easily replaceable without loss of data when the battery is drained or if it gets broken.

Victims trapped on their rooftops or in the ruins of buildings should be able to contact the rescue service as fast as possible in order for it to prioritise the rescue missions based on their severity and their number of victims. Victims may also need first-aid training over the phone along with medical equipment until a rescue team arrives. Finally, some victims may like to record a testament and farewell messages for their friends and family if they feel like they are not going to make it out alive.

7.2 Defining wireless nodes : network and system architecture

In a disaster scenario, preexisting communication infrastructure may not be available anymore, either because victims would jam all the communication services by trying to communicate to their friends and families or because the disaster cut down power and/or the optic fibres.

In such a situation, rescue missions should be allowed to re-use the licensed spectrum of non-vital services in order to provide the most reliable communication possible by decentralising them and selecting the least crowded frequency bands. This would provide the communication resilience needed by the rescue service to operate safely and efficiently. The list of usable bands in such a scenario should be selected by the state's telecommunication regulation organism and made available to the rescue service.

To further increase the reliability of the network, rescuers' tablets should be able to communicate with each others in a peer-to-peer fashion to form a meshed ad-hoc network. To extend the range of the transmissions, vehicles could have a high-power radio with a multi-kilometres range. Likewise, when some rescuers are cut from the network or are going to be, a relay drone could be flown to make sure the network stays connected while waiting for a terrestrial relay to be deployed.

Software-defined radios could allow the rescue teams to communicate with each others without needing to pre-allocate spectrum that may or may not be available everywhere in the disaster zone. Indeed, the contributions from Chapter 4 allow the different network nodes to find each others and to react to spectrum perturbations to keep communications going and exporting several services on different spectrum bands.

To minimise latency, long-distance relay nodes should agree on using the same central frequency for communicating. This allows transceivers with different bandwidth capabilities to communicate at the maximum rate possible as they can use the entire bandwidth of the slowest radio. As frequency bands closer to the central frequency can be used by more devices, when a packet needs to be sent, a frequency band as far away as the central frequency should be allocated. The actual distance will depend on the current spectrum usage and the receiver/transmitter capabilities which are known thanks to the beaconing system. Relay nodes should also select the next central frequency that should be used in case the frequency band gets perturbed too much by radios external to the network. In case of a total jamming of the frequency band, relays can re-initiate a search for surrounding relays before agreeing a new central frequency again. Here is an example of beacon for a relay node:

```
<beacon_frame>{ node_id=xxx, tx_pwr=30dBm,
[
  { {band_relays}, len=1.0, period_offset=0.0 },
],
period=1000ms, cur_period_offset=0.3 }
```

Tablets should periodically be available on the relay's central frequency to allow long distance communications. This frequency band could also be used by two tablets to allocate a frequency band on demand when they need to exchange a lot of data. The rest of the time, the tablets' radio should be off to save power. Synchronisation between nodes is made possible thanks to the beaconing system introduced in Chapter 4.

Depending on the number of relays between the emitter and the receiver, the bandwidth and latency of the transmission will vary. A route thus needs to be selected according to the needed QoS. Such a network could thus accommodate the rescuers' need for constant connectivity, text/voice/video communication and live update of the maps and annotations on it.

The central command should be able to locate rescuers to dispatch the closest rescuers when an emergency such as a cardiac arrest of a victim arises. This means the rescuers' tablets should make their position known periodically. This traffic can be used by the network layer to gain some insight about the network topology without being proactive.

A geographic routing protocol such as GRP should then be used to find the shortest path from one node to the other. The localisation of the nodes can be acquired in a cross-layer fashion by a relay by analysing the packets containing the current GPS location of a node. This is possible because the relays should know their position and mobile users know the position of the central command. The GPS position packet of a node will thus always find its way to the central command and will force the relay nodes along the way to update their routing tables. A relay that did not receive a GPS position packet from a node for a long time should thus redirect packets to a relay closer to the central command.

To further improve the relays' routing decisions, we propose that they should be aware about which relays they can contact to access every node in the network along with their respective theoretical maximum bandwidth and maximum latency. To keep the relay topology up to date, the relays could rely on the periodic beaconing system to detect when a new relay has become active or when a neighbouring relay has suddenly become inactive. When a relay detects a new node or detects that one of its neighbour has disappeared, it should broadcast this information to all the relay nodes for them to update their routing table.

7.2.1 Node type 1 : Aerial drones

Figure 7.1: SenseFly Swinglet CAM: An autonomous aerial imaging drone [13]

When arriving on a disaster site, rescuers should first map the surrounding area to evaluate the extent of the damages and to avoid putting the the rescuers' life in danger. The fastest and simplest way of doing so is by launching a wireless aerial drone, like the one seen in Figure 7.1, that will autonomously map the area. This new map could be added as a layer to maps prior to the disaster for comparison and to re-use street names when possible instead of coming up with new names for every street. Photos could be downloaded in real time from the drone to reconstruct a 3D model of the area [169]. Drones may keep on patrolling to look for survivors and fire outbreaks.

The drone may also carry a GSM base station to allow victims to contact the rescuers using their cellphones like they would in a more typical emergency. Drones could also broadcast text messages to all cellphones in their vicinity to inform them of the current situation and give them instructions. More importantly, active cellphones trying to authenticate onto this mobile GSM base station could allow mapping and tracking the position of the civilians. This could also speed up the discovery of unconscious victims, the recovery of deceased people and their identification. The OpenBTS project [170] is already able to provide a software-radio-based GSM base station that could achieve this goal with the help of wireless telecommunication providers to allow their users to automatically connect onto those.

Long-term stationary drones such as balloons or circling planes could also be used as a relay in the mesh ad-hoc network in order to increase the reliability and bandwidth

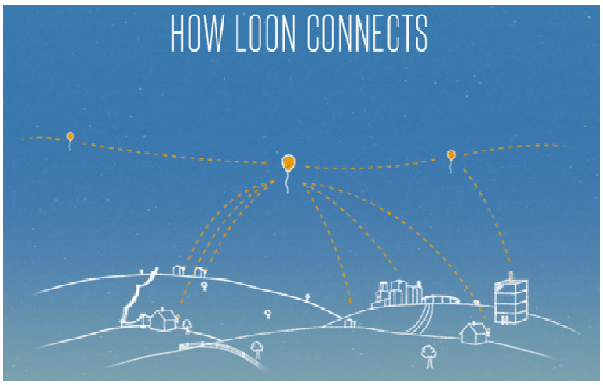


Figure 7.2: Google Loon Project: Providing Internet access to a large area [14]

for long-distance calls in the disaster area. Google has been working since 2013 on using balloons flying at 20km of altitude (twice as high as commercial jet flights) to bring Internet everywhere around the globe. This project, named project Loon [14] and shown in Figure 7.2, is being tested successfully in New Zealand.

Drones could also be used for urgent package delivery such as medical supplies. A rescuer could ask for the delivery through his/her tablet, a drone would then fly to a supply tent to be fitted with the package. The drone would then fly to the rescuer who issued the request by first going towards the GPS location from which the request was made. When the drone would become in range of the rescuer, the drone will request a landing/dropping zone on the rescuer's tablet. This is possible because the drone and the rescuer's tablet could find each others and because they share a map that allows the user to point where the drone should deliver the package. Because the drone knows its position and the current topology, it could also deliver the package at the safest position close to the rescuer if this one is too busy to answer the drone's request. An example of such a drone is presented in Figure 7.3.

Figure 7.3: Amazon Prime Air: A commercial drone delivery proposed by Amazon [15]

This urgent delivery service could also be directed to civilians that can not yet be reached for the rescuers. The drone could identify the cellphone that requested the delivery and drop the package close to its current location.

Thanks to out node discovery mechanism, drones could avoid collisions between themselves by broadcasting their flight direction and current GPS position like commercial air planes do. Rescuer vehicles such as helicopters should also be fitted with such a system to make sure no drones would cross its path. This discovery mechanism running on software radios also provides a cognitive behaviour to relays by allowing them to dynamically select white spaces for their communications.

7.2.2 Node type 2 : A fully autonomic wireless tablet

Figure 7.4: An example of rugged tablet that could be used by rescuers [16]

The second node type is the rescuer's mobile terminal we referred to as "tablet" earlier. It is used by rescuers to view up-to-date maps, access map annotations (safety, work to be done, potential victims and their needs, etc...), and communicate with other rescuers and victims by text, voice or video. It is also used to communicate with wireless sensors of all sorts and should finally be able to take pictures and videos for press and archive purposes. The tablet would resemble the one shown in Figure 7.4.

The tablets should also be able to communicate with other tablets either directly if they are close-enough, or by using relays. In the absence of a relay, other tablets may relay data to and from the nearest one terrestrial or aerial relay.

Since tablets are not personal, may run out of battery or get damaged, data such as video or audio recordings, should be duplicated on multiple devices until it reaches a cloud service which can guarantee its availability. This should however be avoided as much as possible to reduce the battery drain on other tablets.

Software radios are the perfect candidate to fulfil all these network requirements. Indeed, just like for the relays, we need tablets to be able to identify collaborating nodes in their surroundings. They also need to be able to evade crowded or perturbed frequency bands while also being able to change all their PHY and MAC parameters to match their receiver' and QoS's requirement. This is critical to be able to communicate with wireless nodes using hardware radios. Finally, the beacons we proposed in Chapter 4 allow radios to be put in a sleep mode most of the time when not in use.

In the following example beacon, the radio is not really used by the rescuer which means it can save power by powering itself down. The radio has a cycle of 1 second where 50 ms (5%) are used to communicate with the wireless sensor network that will be defined in the node type 3, 100 ms (10%) are used to be available to the relay network in case an incoming connection happens and then it can sleep for 850 ms:

```
<beacon_frame>{ node_id=xxx, tx_pwr=10dBm,
[
  { {band_WSN}, len=0.05, period_offset=0.0 },
  { {band_relays}, len=0.1, period_offset=0.05 },
],
period=1000ms, cur_period_offset=0.08 }
```

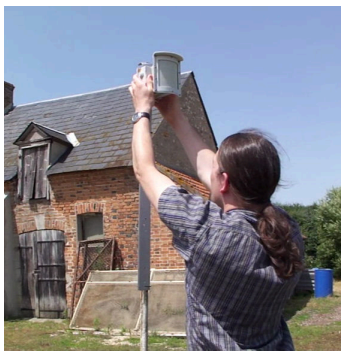
In this other example beacon, the radio is used at its maximum capacity to communicate through a relay. The radio is thus available to the relay for 95% of the cycle.

```
<beacon_frame>{ node_id=xxx, tx_pwr=10dBm,
[
  { {band_WSN}, len=0.05, period_offset=0.0 },
  { {band_relays}, len=0.95, period_offset=0.05 },
],
period=1000ms, cur_period_offset=0.16 }
```

7.2.3 Node type 3 : An extremely low-power wireless sensor node

(a) Deployment of a PIR sensor

(b) Deployment of a seismic sensor



(c) Deployment of a SPIRIT sensor

Figure 7.5: Example of wireless sensor nodes that could be used in a disaster area

The third and final node type is a low-power wireless sensor node. It's role is to enhance the safety of rescuers by monitoring the air and look for potential looters who may enter dangerous areas and gravely hurt themselves.

This node type can enhance the safety of rescuers by monitoring the air in search of dangerous gazes, may they be explosive or toxic. Indeed, in a flooding scenario involving tidal waves such as when a tsunami happens or when a levee breaks, gas bottles from

a clearly advertised site may be transported over multiple kilometres to seemingly non dangerous areas. These bottles may then start leaking and releasing potentially-toxic gas in the atmosphere. At the right concentration, the gas may even detonate and potentially kill some rescuers or innocent by-standers.

One air-monitoring node should be placed onto each rescuer for his/her immediate protection. These nodes would communicate with the rescuers' tablet. However, to lower their power consumption, they would only start transmitting on a pre-established frequency when there is a problem to find the rescuer's tablet. Other transmitters are unlikely to be able to jam the transmission because of the proximity between the node and its receiver. Tablets should thus be available on this band periodically. Finally, the node could also beep more or less loudly to inform immediately the rescuer about the danger level. The number of beeps could also indicate the type of danger.

If a node does not detect any above-normal concentration of any gas, the radio should be put to sleep constantly. When it does, it should enable its radio in receive mode to check if surrounding sensors are not sending alerts. If this is the case, the node should emit a special beeping pattern to let the rescuer know that a potential danger is here. The node should however not send any packet until the concentration reaches a dangerous level so as this pre-danger warning does not spread too far away from the leak's source. This solution creates a very simple correlation mechanism between mobile sensor nodes without consuming power at all when such correlation is unnecessary.

Looting is a frequent problem in a disaster site and it is difficult for law enforcement to prevent looters from entering the disaster area. Keeping them out of danger zones should become a priority a few days after the disaster. Secondly, detecting looters in residential areas could help law enforcement to prevent theft. A wireless sensor network composed of multiple wireless nodes and heterogeneous sensor types such as infrared barriers (SPIRIT) and seismic sensors (both seen in Figure 7.5) could detect looters and identify their type.

Deploying the network should be extremely easy. Nodes should thus be made as small as possible and no configuration should be required aside from assigning them an area and then adding them in the Command and Control application at their GPS coordinate and orientation. This can easily be done by the rescuer installing the wireless sensor by using his/her tablet during the deployment.

Because this node type has a very predicable network behaviour, there is no need for advanced cognitive behaviour that would require any special hardware. Since the network should be silent most of the time, nodes willing to communicate can emit a long preamble. If other nodes check for the presence of a preamble with a period slightly lower than the preamble time, they will always be able to detect transmissions and receive them. If no preamble is present, the radio can be put back to sleep immediately to save power. This saving however comes at the cost of latency and throughput.

Our contributions introduced in Chapter 3 allow for an easy deployment, reduce the number of messages exchanged to increase the battery life while our correlation mechanism would reduce false positives and negatives and would also autonomously be able to detect faulty sensors and isolate them. Our introspection mechanism coupled to our GUI called Diase also allows network administrators to check that the network is operating properly. Finally, an automatic response such as sounding an alarm, playing an automated message and/or taking pictures of the intruders is also possible with our contributions to scare the looters off.

Tablets may detect this network thanks to the nodes heartbeats needed by the correlation node to make sure all sensors are still available. A tablet could then join the network by following the same procedure the sensor nodes used to add themselves to this wireless sensor network.

7.3 Radio frequency : Continuously selecting the best RF bands

In Figure 7.6, we introduce a waterfall view of the RF spectrum as seen by a relay in the proposed network.

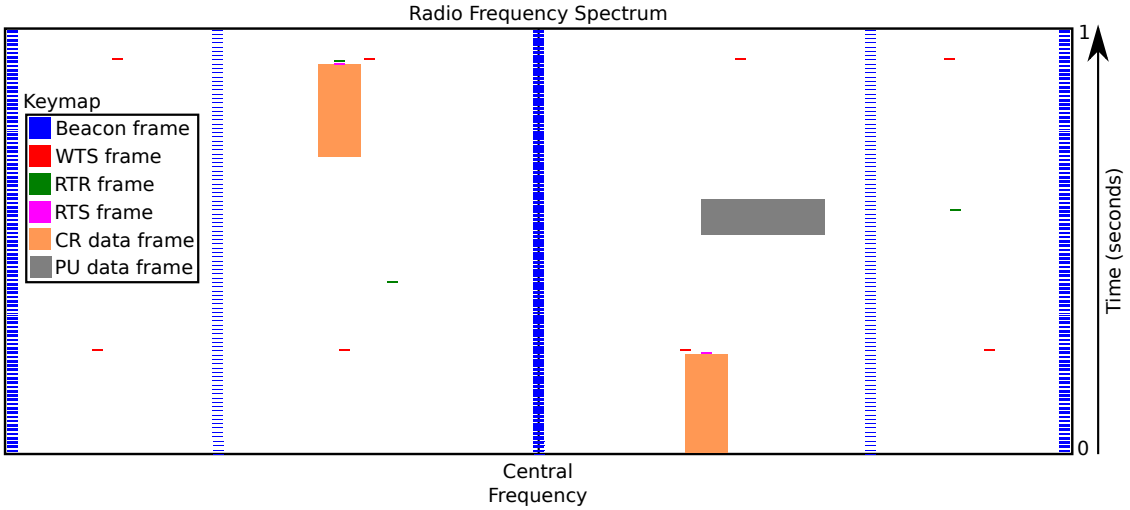


Figure 7.6: Example of the spectrum usage done by relays in the proposed network

In blue, we can see the beacons of 4 different nodes, 3 of which have a 25 MHz bandwidth while the last one has a 15 MHz bandwidth. All the nodes send 3 beacons at a time every 10 ms. In red, green and purple, we can see the MAC-layer controlled frames we proposed to allocate spectrum. These frames can be followed by an orange frame containing the actual data frame. A relay may not receive the RTR frame if it is far away from the receiver. Likewise, it may only receive the RTR frame if it is too far away from the emitter. Some non-collaborating nodes that could potentially be primary users may also use the spectrum to send frames. These frames are shown in dark grey.

When the frequency band used by the relay nodes becomes too congested by non-collaborating nodes, nodes should start to dedicate some time to sensing the other bands allocated to non-vital services to extract usage statistics on the bands' type of usage (periodicity, length and width of the transmissions).

The selection process of the RF band to be used for communication can be done using our contribution from Chapter 6. The bands' statistics are first fed to a prediction stage to detect periodic behaviours. Likewise, statistics of the average bandwidth necessary are collected. Models are fed with the bands' usage statistics along with the average bandwidth and the maximum latency allowed to guarantee a good end-to-end QoS. The band model will then check if the band it is associated to can support both the wanted bandwidth and latency. The scoring system will compute each band's score based on their width in MHz, expected average bandwidth and latency, and the variance of those three parameters. The decision stage is responsible for deciding whether changing the current band would make sense or not.

A node can not decide to change band unilaterally, this decision should be made by all the relay nodes. When a node decides a change of band should be made, it should send an alert containing the expected performance/usage ratio for both the bandwidth and the latency metrics for both the current band and the list of candidate bands. This alert should be received by the surrounding nodes that should try to find an alternative route if they often use this node for communicating with other nodes.

The alert should also be received by a network-wide correlation node that will decide to change bands when a criticality level gets higher than a given threshold. This threshold is calculated based on the wanted end-to-end QoS. The contribution of a node to the current criticality level should depend on how many routes are affected by this poor performance and how many routes can not reach the expected QoS. When the criticality level is reached, the correlation node uses the information found in the alerts that contributed to reaching the threshold and try to find the most appropriate band for all the nodes that emitted an alert. The correlation node should then propose this new band to every relay node in the network. The solution we proposed to detect the need for a different frequency band for the relay is very similar to the contribution found in Chapter 3.

Upon receiving such a message, relay nodes will need to update their beacon to reserve some time for sensing the proposed band and collecting statistics. This allows the network to stay available during the transition. After a timeout specified by the correlation node that depends on the urgency of the switch, relay nodes should send their statistics to the correlation node that will take its decision to either use this band or try the next best one.

When the correlation node decides which band should be used by the relay nodes, it should send a message to all the relay nodes telling them they should switch their band in a certain amount of time. This delay should be high-enough to make all the nodes

receive the message before it expires. Nodes should follow the procedure proposed in Chapter 4 to announce a band switch by changing its beacon.

If a node did not receive the message in time and did not receive a beacon announcing the change, it will still be able to find the other node in the band that was last proposed by the correlation node. If it still can not find the other nodes, it should spend its time on sensing to find them again. Based on our simulations in Chapter 4, it should be able to find them in a matter of seconds in the worst case scenario.

In the event a single band can not be used across the network, the network should be split in two or more sub-networks. Each sub-network should use a single band and have the equivalent of edge routers that can spend half their time on a band and half the time on the other in order to keep the network connected. The performance impact of such a split will depend on the number the volume of data that needs to be exchanged between the two sub-networks and the latency needed. The minimum performance impact can however be computed by the correlation before the split happens when the correlation node computes the beacon of the edge routers.

Finally, the node which is in average the closest to every other node of the sub-network should be elected as the correlation node.

7.4 Hardware requirements : Performance and availability

Rescuers should focus on saving lives, not on making sure their tools have enough power left to complete their tasks. They should be able to know that no matter what they do with their tablet, they will still be able to go through the day with it. It is thus necessary to modulate performance and power consumption to allow rescuers to trust their device will be available for the rated time of 12 hours.

A battery capable of supplying the tablet for 12 hours when it is using all of its peripherals, CPU, and network resources constantly is not realistic because it would require it to be much heavier than what would be needed in most situations. A suitable battery large-enough to accommodate most of the rescuers' needs while being light-enough should thus be selected. The average power consumption (P_{avg}) should then be made lower than the capacity (in Joules) of the chosen battery (C_{bat}) divided by the wanted battery life (T_{bat_life}, in seconds), as can be seen in Equ. 7.1.

$$P_{avg} \leq \frac{C_{bat}}{T_{bat\_life}} \tag{7.1}$$

Likewise, a relay should guarantee a certain availability and predictability in its failure time to allow rescuers to change its batteries or fly a drone to its location before it becomes unavailable.

Other reasons for which we need to be able to cap the power consumption of the tablet is to make sure the tablet stays in its Thermal Design Power (TDP) and that it does not consume more power than the battery is rated to be able to output safely. The TDP of the table may actually be quite low due to its rugged nature that adds layers of elastic materials that do not conduct heat as well as traditional solid casings.

A Graphical User Interface (GUI) could be provided with the tablet to let rescuers or technicians diagnose poor-performance issues involving applications draining the power, although no rescuer should ever be forced to know about it in order to use his device. This GUI would display the power/energy usage of the different services along with the repartition of its energy consumption on the CPU, GPU, modem, screen display and sound card. A simpler version of this GUI has been developed by Intel under the name powertop [171].

In this section, we evaluate how the network interface and the processors can react to a power draw too high and vary their performance to cap it.

7.4.1 Network

Not every node type requires network performance management in order to meet their availability goal. For instance, the power cost of keeping the drone in the air being far greater than the power consumption of the wireless radio. We can thus consider leaving the radio always on for a maximum availability and to lower the forwarding delay while not impacting considerably the flight time of the drone.

Likewise, terrestrial human transport vehicles, used for relaying, should not worry about their network power consumption as their batteries are usually very large and can be recharged on demand by starting the combustion engine of the vehicle.

This is however not the case for terrestrial relays since all their energy consumption comes from transmitting and receiving data. These relays need to be deployed on the highest ground possible to extend their coverage. To ease their transport and their deployment, they must be relatively small and light which limits the size of the battery they can have. To lower the needed battery capacity, these relays could be fitted with solar panels to recharge the battery during the day.

When a relay consumes power at a rate that will not allow it to fulfil its availability contract, it should take measures to lower its power consumption. It can start by asking surrounding nodes to find alternative routes. If the power consumption is still too high, the relay should alert the command and control that it will start throttling transmissions passing through it by delaying or dropping non-critical packets at first and then powering down the radio periodically if it was still not enough. Finally, the relay should advertise its end of life 2 hours before it happens to let the opportunity to rescuers to flight a relay drone to its location until a rescuer changes the relay's battery or the sun recharges the battery the next morning.

This graceful degradation of performance brings predictability to network transmissions which is more important than immediate throughput and latency as it brings reliability. This mechanism could also be used by tablets serving as relays at the difference that the tablet's operator should be prompted if the tablet should keep on relaying data or not when relaying packets exceeds the allowed relay power budget.

Such a network architecture could thus accommodate the rescuers' need for constant connectivity, text/voice/video communication and live update of the maps and annotations on it while guaranteeing some degree of availability for the entire time the relays are supposed to be active.

7.4.2 Processors and accelerators

A big contributor to power consumption in the tablet is computational power needed to performs calculations and display content on the screen. This computational power is provided by both the CPU and the GPU which can both be limited in their power consumption at a high rate, as shown in Chapter 5.

Low-power wireless sensor nodes can mostly predict exactly when and what they will need to do at any time due to a single program running with almost no user inputs. They can thus limit their power consumption in the code of the application by coalescing data transmissions or by enabling peripherals such as the radio right before they are needed. This enables the node to be programmed to get the lowest power consumption possible while having a predictable performance cost. This allows the application programmer to statically make sure the node's availability will be met because he/she controls most of the variables.

Unlike low-power wireless sensor nodes, a rescuer's activity on his/her tablet is not predictable in advance. Indeed, it has multiple applications which have different power and performance needs that could be used concurrently or not. It is thus not possible to statically guarantee the availability of the device using the techniques talked about for the lower-power wireless sensor nodes which required a single application with full control on the node and very few user inputs.

One solution for the tablet could simply be to limit the power consumption of the tablet so it never exceeds P_{avg} by allocating power budgets for the CPU, GPU and the radio and letting them make sure they stay in budget, as proposed in Chapter 5. However, some applications may never be able to run if they consume more power than that. Instead, it makes more sense to make sure that the average power consumption since we started the tablet is lower than P_{avg}. Upon reaching 90% of P_{avg}, the tablet should start throttling applications and services based on their priority so as the instantaneous power consumption would tend towards P_{avg}. A power budget should thus be allocated to both services and sub-devices instead of sub-devices only. A service is potentially composed of multiple processes. We propose using a system such as Linux's Cgroups [172] to group processes into one service. Cgroups are a form of very lightweight virtual machine.

Because of the high dynamicity of the workload, an accurate estimation of the energy consumption of each service is required. This means processors and accelerators should have a fast and accurate hardware model of their power consumption that takes into account the context switching and wake-up costs to provide this accurate reading.

Once a service exceeds its power budget, the frequency of the processors and accelerators running it should be lowered until reaching the most efficient frequency. If this is still not enough to limit the energy usage of the service to its budget, the service should be throttled by forcing the CPU and/or GPU to do something else (or go to sleep if no other service requires to be run) for some time.

Sharing resources such as network throughput or computational time fairly or according to a policy between multiple services can be achieved by respectively using QoS algorithm such as DiffServ [173] or a fair scheduler such as Linux's Completely Fair Scheduler [174]. Per-service network bandwidth and CPU usage caps are already possible in Linux thanks to the Cgroups. We have however been unable to find similar algorithms for limiting the power consumption of services through a scheduling decision.

The processors and accelerators found in the tablet should self-optimise to lower the power consumption at all time, as proposed in Chapter 5. It is also the operating system's job to make sure applications run on the most appropriate processor/accelerator based on the application's performance need and the incurred power consumption by the application on the system. For instance, the OS is responsible for moving tasks on the right core type in a big.LITTLE processor scenario, as proposed in Chapter 6, which can yield up to 75% power saving for the CPU.

7.4.3 Other peripherals

Other peripherals such as the display's backlight can be responsible for a large part of the power consumption of the a rescuer's tablet. To save some power, multiple techniques are already exist in current smartphones and tablets. For instance, the backlight power is controlled by the ambient brightness to make sure it stays visible. Also, a proximity sensor disables the screen when the screen is not visible, either because it is up-side down on a table or it is held up to one ear.

Likewise, sound's volume could be adjusting by the ambient sound volume. When there is little sound around, the volume could be kept low. On the contrary, if the ambient sound is loud, the volume should be increased to make sure the rescuer can hear it. The advantages of doing so are two-fold. Power can be saved by outputting just the right amount of power to be heard. The second advantage being that an automatic volume management will make sure the no rescuer will accidentally leave the volume on low which could result in him/her not being reachable because he/she would not hear the alert.

7.5 Conclusion

In this chapter, we created a scenario that showcase our contributions and how they interact with each others. In this disaster-relief scenario, rescuers require constant communication among each others and need tools to better locate and help victims. In such a scenario, the rescuers' time is a scarce resource and moving from one part of the disaster area to another can take a lot of time due to a damaged road infrastructure. We proposed a network architecture composed of multiple wireless radio nodes ranging from low-power wireless sensor nodes to flying drones. These wireless nodes mostly self-manage and have a predicable availability thanks to our contributions. This local power management produces a global improvement on the entire network's availability.

Our propositions can seem a little futuristic but, as we found out, all the hardware to make it a reality has been available for several years already. Our PHY/MAC signalling protocol allows nodes to find each others and react to changes in the RF spectrum usage to keep being available. Our in-network data processing scheme enables substantial power savings, increasing the battery life of intrusion detection systems meant to protect both rescuers and civilians. Our autonomic hardware-level power management contribution as well as the OS-level decision engine enable significant savings for the rescuers' tablet while also guaranteeing a level of availability for a pre-defined time-frame which translates a global improvement in the entire network's available. The decision engine has also been used to create statistics of usage of the RF spectrum by both our network and external users which are then used to identify which bands are best-suited for the current usage of the RF spectrum. More research is however needed at every layer to enhance the autonomic behaviour for power management and increase the nodes' availability guarantee.

Prototypes of the proposed autonomic tablets could soon be possible using a Tegra K1-based tablet and a lot of reverse engineering to be able to access the software radio modem [116].

Chapter 8

Conclusion

Contents

8.1 General conclusion

The environmental impact of Information and Communications Technology (ICT) has been raising exponentially to equate the impact of the airline industry. The green computing initiative has been created in response to this observation in order to meet the 15%-30% reduction in green-house gases by 2020 compared to estimations made in 2002.

In this thesis, we studied power-saving techniques in wireless networks and how they interact with each others to provide a holistic view of green networking. The thesis goes down the network stacks before going up the hardware and software stack.

Contributions have been made at most layers in order to propose an autonomic wireless network where nodes can work collaboratively to improve the network's performance, globally reduce the RF spectrum usage while also increasing their battery life.

We now summarise the contributions found in each chapter of this thesis:

Application & Network: decentralising data processing

In Chapter 3, we discussed the power cost of sending and receiving messages. To lower this cost, fewer and smaller messages should be sent. An effective way to do so is to localise communications to reduce the number of hops a message has to go through before reaching its destination. Localising communications means decentralising them as much as possible.

Thanks to our in-network correlation scheme of heterogeneous and redunded sensors, we managed to drastically localise transmissions for applications monitoring spatio-temporally-correlated events. This drastically reduces the number of messages sent which increases the battery-life of the network. An interesting side effect of this approach is that it made automatic response to an event by the network trivial to implement in an efficient way thanks to the data-centric routing mechanism we used and the in-network reasoning capabilities of our system.

Such a drastic reduction in the number of exchanged messages and the decentralisation of the data processing however makes it nearly impossible for the network operator to verify that the network is working properly. We proposed a mechanism to make it possible for the network operator to fetch an history of all the meaningful events that happened in a recent past. This mechanism is also used in the network to allow detecting malfunctioning sensors at a relatively low ROM (code) and RAM (data) cost by also introducing a proposed reputation mechanism.

The complete proposition exhibits multiple autonomic traits such as:

- Self-configuration: Ad hoc network routing and discovery of the sensors in an area allow the reasoning functions to compute criticality thresholds on the fly;
- Self-optimising: Learn about the false positives and false negatives rates of sensors and adjust thresholds based on them to limit emissions;
- Self-healing: Find another route and reconfigure the thresholds when a node becomes unavailable. Automatically evict faulty sensors.

PHY/MAC: Efficient spectrum sharing

In Chapter 4, we introduced the concept of cognitive radio networks that aims at allowing nodes to understand their RF environment and select the best frequency band and modulation to communicate with any other surrounding node, cognitive or not.

Creating a truly-cognitive radio requires a direct access to the spectrum which is possible using a software-defined radio (SW radio). Such a radio can listen to a wide frequency band and modulate/demodulate one or multiple transmissions happening inside this band, thus allowing for an efficient sharing of the spectrum while also allowing radios to find each others quickly. It thus also allows tuning the PHY parameters and the MAC protocol depending on the current QoS needs.

Since cognitive radios (CRs) are very agile and may need to communicate with multiple radios on different frequency bands, we proposed a mean of synchronisation between CRs. This proposition allows CRs to know where and when in the RF spectrum surrounding nodes will be available. This allows CRs to discover surrounding CRs and makes it possible for them to synchronise their hopping pattern to guarantee a maximum delay and a probable minimum throughput. The discovery of a new node can happen in less than a second in realistic scenarios using $400 SW radios.

Our second proposition allows CRs to negotiate the allocation of spectrum and select the best modulation for a transmission. This proposition is an improvement over IEEE 802.11's CSMA/CA by providing advanced cognitive and autonomic abilities to spectrum management

Both propositions increase the autonomic abilities of wireless nodes:

- Self-configuration: Allow CR discovery at any time, update the hopping-pattern at run time to become reachable by other CRs;
- Self-optimising: Optimise the hopping sequence to improve performance with the CRs the CR is primarily communicating with, then keep their hopping patterns in sync even in the presence of clock drifting;
- Self-healing: React to (un-)intentional jamming by leaving the selected frequency band and selecting a less-crowded one without shutting down communications and without prior agreement among the CRs.

Hardware: Autonomic power management

In Chapter 5, we introduced the power management features found in modern hardware which are sufficient to implement a fully-autonomic power management running on the processor itself.

Indeed, hardware is increasingly complex and becomes harder and harder to configure and optimise for every situation. Furthermore, some optimising decision such as clock gating are so short-lived that they cannot possibly be taken by the main CPU at a fast-enough speed without increasing the overall power consumption or without impacting the performance of applications running on the CPU.

To optimise both the power consumption and performance, modern hardware has introspection capabilities such as performance counters which can provide information about how much each part of the processor has been used during the last few hundred milliseconds. Using this information, it is possible to tune the performance of the processor to improve performance of the parts of the processors that are currently the bottleneck while slowing down the parts that are less used. This operation can be done using the Real-Time Operating System running inside modern CPUs and GPUs which has all the capabilities to perform those operations.

Based on our reverse engineering, the following autonomic traits can be implemented:

- Self-configuration: Processors can read a microcode from an embedded ROM when powered-up and execute it to perform the required initialisation of the processor before it is able to execute any code;
- Self-optimising: Performance counters can be used to optimise the performance and lower the power consumption of the processor;

- Self-protection: Hardware contexts can isolate users in their own virtual memory address space. Processors' cryptographic abilities can be used to decode encrypted information and keep it secret from any user.
- Self-healing: Processors can react to over-current and over-temperature by automatically lowering their operating clock frequency to lower their power usage.

OS: Real-time power management decisions

In Chapter 6, we demonstrated the need for a run-time decision engine for power management. Indeed, in modern computers such as smartphones, there are multiple ways of carrying out the same task. Each way can be the most efficient one in some conditions and the worst one in other conditions. It is thus very important to be able to react quickly to changes in the usage pattern in order to constantly select the most efficient way. The decision engine should also be able to learn from the usage pattern and avoid the so-called ping-pong effect that can actually be damaging to both performance and power consumption.

Our proposition thus enables the following autonomic traits:

- Self-optimising: Select the most efficient way of carrying the current task based on the prediction of what is thought to be going to happen in the near future;
- Self-healing: Learn from previous transitions by gathering statistics and building up knowledge to avoid making the same mistakes over and over again.

Proposing a green network for disaster relief

In Chapter 7, we showcased all our contributions together in a disaster relief scenario where all the communication infrastructures have been destroyed. Together, our propositions allow the network to guarantee its availability while nodes work collaboratively to provide the best network performance possible. Locally, nodes follow the autonomic computing vision by automatically managing both the RF spectrum and the power usage.

This chapter also demonstrates the modularity of the different autonomic behaviours. Indeed, we introduced three types of wireless nodes (sensors, relays and tablets) and have shown that they all use the same autonomic features, but not always in the same way. Some features may also not be needed at all depending on the node type. An example would be the sensor node not requiring any autonomic behaviour in the hardware because one application controls the whole node. We have also shown that our software architectures are generic-enough to be applicable for many types of decision making. For instance, the run-time decision engine can be used without modifications to locally monitor the spectrum and check that the select frequency band is still sufficient to guarantee the needed QoS. The contributions of Chapter 3 can then find a global consensus to select the most-suitable frequency band(s) across the network.

8.2 Future work

Due to time constraints, we have not yet addressed the following challenges:

8.2.1 Decentralised data processing in Wireless Networks

In our contribution to decentralise data processing in a Wireless Sensor Network, we did not study the impact of adding redundancy in the area-level centralisation to increase the reliability of the solution. The impact could be kept low as all the nodes in an area are supposed to be able to communicate with each others. It however requires modifications to both the routing algorithm and the MAC protocol.

In our experimentation, we used a centralised version of a Publish/Subscribe routing algorithm to demonstrate our solution. It is however widely inefficient in a large area network as even messages that could stay local have to be routed to the Pub/Sub broker and then routed back to the neighbouring node interested in it. To improve the power savings of our data-processing decentralisation proposition, we would like to develop a fully decentralised Publish/Subscribe routing algorithm that would make sure that messages never get forwarded unless they have to.

With this decentralised version of Pub/Sub routing, the node emitting an alert could directly send the alert to all the nodes which are interested in it. Even if all the nodes of an area are able to communicate with each others, the alert has to be duplicated to every node interested in it because of limitations in our MAC protocol. To reduce the number of emissions, it should be possible for all the correlation nodes of an area to receive the same alert message. We thus want to study the feasibility of a local multicast support at the MAC-layer which would support acknowledgements to make sure that every receiver actually got the message. The list of all the receivers that need to receive the message is available at the network level, in the Pub/Sub's subscriber list. The alert should thus be emitted as long as not all the receivers have acknowledged it. This solution will possibly require to add a unique ID to the message so as a node that already acknowledged reception of the message would not be forced to re-emit one when the alert is re-sent because of another node. Another possibility would be to drop any receiver that already acknowledged the message from the re-emission list. The main challenge of implementing such a proposition lies in the very limited amount of RAM found on sensor nodes.

8.2.2 Cognitive Radios Network

Still at the MAC-layer but on cognitive radio networks, we would like to propose better cognitive abilities to better adapt to the RF environment and to the QoS of the applications running on the network.

For example, nodes requiring an extremely-short latency and predicable transmission times but requiring a limited bandwidth may permanently allocate a narrow frequency band. This would satisfy the application's QoS while not really impacting the ability for other nodes to communicate. Such narrow bands should however be allocated close to each others and close to primary users if applicable in order to avoid fragmenting the spectrum which would hinder opportunistic CR communications using a higher bandwidth.

As cognitive radio nodes are not very limited in memory usage, it is possible for them to use any cross-layer information to improve the quality of their routing decision in low-traffic wireless networks. Example of cross-layer information that can be leveraged is the GPS coordinates of nodes sent to centralisation point by mobile nodes like explained in Chapter 7.

8.2.3 Hardware

After working on the network and applications, we would also like to improve on the hardware side.

We would first like to study the impact of encryption on wireless nodes when using the hardware-encryption capabilities of TI's CC430 [175]. The hardware capabilities are limited to a single symmetric algorithm (AES-128) but they should provide a substantial reduction in code size, performance and power consumption that we would like to quantify.

We then would like to propose a generic DVFS policy that detects the bottlenecks inside the GPU/CPU and increase the clock of the associated clock domain while also diminishing the clock of the domains which are not as needed. Such fine-grained information about block usage can be retrieved very rapidly using performance counters. The policy would also take into account the temperature to keep it as low as possible to keep the power consumption down. Finally, the policy would make sure the processors do not consume more than their assigned power cap in order to guarantee a minimum battery-life and/or the safety of the hardware.

Finally, we would like to improve our run-time decision engine to dynamically rate the effectiveness of our prediction mechanism. This scoring can be done by comparing the result of previous predictions and check with what actually happened. With such a system, it becomes possible to detect the most appropriate prediction system at run-time. This would allow to improve the accuracy of our predictions which would lead to a better decision which, in turn, would result in a better QoS enforcement and a lower power consumption.

8.2.4 Green networking with Energy-harvesting nodes

In this thesis, we considered that our wireless nodes were powered with a battery which may have been continuously recharged using energy harvesting. We however did not consider the case where a wireless node would be powered only by a small capacitor recharged using energy harvesting. Such nodes may need to perform work in bursts when their power input is low and may perform heavier computations when their power input is higher or when the capacitor is full. This further increases the complexity of operating the network which increases the need for an autonomic behaviour at every layer.

Future work will evaluate the suitability of our propositions for such type of nodes and propose new ways of collaborating in an autonomic fashion to keep the administration of the network as simple as possible.

Appendix A

List of publications

Contents

In this appendix, we introduce the list of publications we made in international conferences and workshops. The full list of poster presentations and talks is however available online at http://phd.mupuf.org.

A.1 Book chapters

A.1.1 Green Networking

Editor Francine Krief

Publisher ISTE & Wiley

Contribution I contributed about 14 pages to the chapter 6 called "Autonomic Green Networks" where I talked about self-protection.

Description This book focuses on green networking, which is an important topic for the scientific community composed of engineers, academics, researchers and industrialists working in the networking field. Reducing the environmental impact of the communications infrastructure has become essential with the ever increasing cost of energy and the need for reducing global CO2 emissions to protect our environment. Recent advances and future directions in green networking are presented in this book, including energy efficient networks (wired networks, wireless networks, mobile networks), adaptive networks (cognitive radio networks, green autonomic networking), green terminals, and industrial research into green networking (smart city, etc.).

Note The book is also available in French under the title "Le green networking : Vers des réseaux efficaces en consommation énergétique", published by Lavoisier.

A.2 International conferences

A.2.1 On optimizing energy consumption: An adaptative authentication level in wireless sensor networks

Authors Martin Peres, Mohamed Aymen Chalouf, Francine Krief

Conference GIIS 2011: The 3rd IEEE International Global Information Infrastructure Symposium 2011

Abstract Nowadays's authentication methods are essentially based on cryptography or any other kind of secret information. These methods are particularly efficient but they are, in certain cases, very power-hungry. As energy is a scarce resource in wireless sensor networks, we propose a new approach that consists in utilising the physical properties of the transmission medium in order to calculate, when receiving a frame, a confidence rating that we attribute to the source of the frame. This information is important in order to decide if it is necessary to authenticate the source using asymmetric cryptography. In the case where sensors are static and communications are slightly perturbed by the environment, the proposed rating can add another layer of control which enables sensors to check the origin of messages. This paper will also study how collaboration between the network's nodes further enhances detection of malicious or third-party nodes.

A.2.2 Overcoming the Deficiencies of Collaborative Detection of Spatially correlated Events in WSN

Authors Martin Peres, Romain Perier, Francine Krief

Conference ICAIT 2012: The 5th IEEE International Conference on Advanced Infocomm Technology

Abstract Collaborative WSN are known to efficiently correlate sensors' events to detect spatially-correlated events with a low latency. However, because of the drastic reduction of messages sent to the network administrator, it is difficult for him/her to understand in what state the network is. In this paper, we propose a modality-agnostic collaborative detection of spatio-temporally correlated events that drastically lowers power consumption by both reducing and localising communication. This contribution can then be used to expose better auditing capabilities that overcome the problems usually found in collaborative networks. These capabilities can in turn be used by both the administrator and the network itself to, for instance, automatically react to faulty sensors. Experiments validate the interest of our proposal in an intrusion detection scenario.

A.2.3 Power and Performance Characterization and Modeling of GPU-accelerated Systems

Authors Yuki Abe, Hiroshi Sasaki, Shinpei Kato, Koji Inoue, Masato Edahiro, Martin Peres

Conference IPDPS 2014: IEEE International Parallel & Distributed Processing Symposium 2014

Abstract Graphics processing units (GPUs) provide an order-of-magnitude improvement on peak performance and performance-per-watt as compared to traditional multi-core CPUs. However, GPU-accelerated systems currently lack a generalized method of power and performance prediction, which prevents system designers from an ultimate goal of dynamic power and performance optimization. This is due to the fact that their power and performance characteristics are not well captured across architectures, and as a result, existing power and performance modeling approaches are only available for a limited range of particular GPUs. In this paper, we present power and performance characterization and modeling of GPU-accelerated systems across multiple generations of architectures. Characterization and modeling must be tightly coupled to conclude what GPUs can optimize power and performance, and how they are predictable. We quantify impact of voltage and frequency scaling on each architecture with a particularly intriguing result that a cutting-edge Kepler-based GPU achieves energy saving of 75% by lowering GPU clocks in the best scenario, while Fermi- and Tesla-based GPUs achieve no greater than 40% and 13%, respectively. Considering these characteristics, we provide statistical power and performance modeling of GPU-accelerated systems simplified enough to be applicable for multiple generations of architectures. Our finding is that even simplified statistical models are able to predict power and performance of cutting-edge GPUs within errors of 20% to 30% for any set of voltage and frequency.

A.2.4 PHY/MAC Signalling Protocols for Resilient Cognitive Radio Networks

Authors Martin Peres, Mohamed Aymen Chalouf, Francine Krief

Conference SoftCOM 2014: The 22nd IEEE International Conference on Software, Telecomunications and Computer Networks

Abstract Our society relies more and more on wireless communication technologies while most of the RF spectrum has already been allocated by the states. As a result, unlicensed bands are becoming crowded which makes it difficult to create a reliable network without using more spectrum than really necessary. Allowing radio nodes to seamlessly switch between different frequency bands without prior synchronisation would allow the creation of a truly resilient radio network capable of avoiding the frequency bands used by nodes that are not part of the network. In this paper, we propose using software-defined radios in order to sense the surrounding RF environment to find the most suitable bands for communication. We also propose a PHY-layer and a MAC-layer signalling protocols to provide a seamless way of discovering other nodes and selecting the parameters that will be used for communicating with them. Our first experimentation results are very promising towards defining a resilient cognitive radio network.

A.2.5 A run-time generic decision framework for power and performance management on mobile devices

Authors Martin Peres, Mohamed Aymen Chalouf, Francine Krief

Conference UIC 2014: The 11th IEEE International Conference on Ubiquitous Intelligence and Computing

Abstract Nowadays, mobile user terminals are usually composed of multiple network interfaces and multiple processors. This means there are usually several independent ways of satisfying the immediate Quality of Service (QoS) expected by the user. Dynamic decision engines are used for selecting the most power-efficient interface and performance states in order to satisfy the QoS while increasing the battery life. Current decision-engines are not generic and need to be mostly re-written when a new processor or radio comes out. This is because there are no generic decision-making framework for real-time power and performance management that would allow creating re-usable bricks that could be shared by multiple decision engines. In this paper, we propose such a framework to ease the implementation of a run-time decision making with real-time requirements and a low memory/CPU footprint to increase the reliability of mobile devices.

A.3 International Workshops

A.3.1 Power and Performance Analysis of GPU-Accelerated Systems

Authors Yuki Abe, Hiroshi Sasaki, Martin Peres, Koji Inoue, Kazuaki Murakami, Shinpei Kato

Conference HotPower'12: 2012 Workshop on Power-Aware Computing and Systems, co-located with the 10th USENIX Symposium on Operating Systems Design and Implementation.

Abstract Graphics processing units (GPUs) provide significant improvements in performance and performance-per-watt as compared to traditional multicore CPUs. This

energy- efficiency of GPUs has facilitated use of GPU in many application domains. Albeit energy efficient, GPUs still consume non-trivial power independently of CPUs. It is desired to analyze the power and performance charateristic of GPUs and their causal relation with CPUs. In this paper, we provide a power and performance analysis of GPU-accelerated systems for better understandings of these implications. Our analysis discloses that system energy could be reduced by about 28% retaining a decrease in performance within 1%. Specifically, we identify that energy saving is particularly significant when (i) reducing the GPU memory clock for compute- intensive workload and (ii) reducing the GPU core clock for memory-intensive workload. We also demonstrate that voltage and frequency scaling of CPUs is trivial and even should not be applied in GPU-accelerated systems. We believe that these findings are useful to develop dynamic voltage and frequency scaling (DVFS) algorithms for GPU-accelerated systems.

A.3.2 Reverse engineering power management on NVIDIA GPUs - Anatomy of an autonomic-ready system

Authors Martin Peres

Conference OSPERT 2013: 9th annual workshop on Operating Systems Platforms for Embedded Real-Time applications held in conjunction with the 25th Euromicro Conference on Real-Time Systems (ECRTS13)

Abstract Research in power management is currently limited by the fact that companies do not release enough documentation or interfaces to fully exploit the potential found in modern processors. This problem is even more present in GPUs despite having the highest performance-per-Watt ratio found in today's processors. This paper presents an overview of the power management features of modern NVIDIA GPUs that have been found through reverse engineering. The paper finally discusses about the possibility of achieving self-management on NVIDIA GPUs and also discusses the future challenges that will be faced by researchers to study and improve on autonomic systems.

Appendix B

List of Software Projects

Contents

During the course of this thesis, we developed multiple software projects or contributed to already-existing ones. They are listed here and categorised by the chapter of the thesis they are related to:

B.1 Decentralising data processing in surveillance networks

We developed 3 main projects when researching on data processing decentralisation in surveillance networks, found in Chapter 3.

The source code of the following projects can be found publicly [176] under the GPLv2 license.

B.1.1 Reasoning services

The main contribution of this chapter consists in the autonomic reasoning services described in Chapter 3. It currently lacks the automatic eviction of faulty sensors but has all the other features (correlation, history, sensor reputation, ...). The contribution can be found in the folder *diaforus/application/reasoning*.

Note The released code does only include the work of LaBRI and is insufficient to run on its own as it lacks the freeRTOS, UIP, Pub/Sub and CoAP implementations along with the platform drivers. Lastly, the toolchain we used was buggy and did not allow us to define default values for variables. We thus had to re-initialise them at boot time.

B.1.2 Build network

The role of the *build_network.py* is to generate the firmware of every node in the network. It takes an input file such as the one defined in Appendix C and parses it. Next, it generates the firmware of each node by first generating the configuration of a node in C header files (stored in the *config/* directory) and then compiling the code using the right toolchain (for real/simulated hybrid networks). This python script can also run the network when nodes were compiled in the simulation mode (x86).

B.1.3 Diase

One of the biggest project of this chapter was definitely the DIAforus Simulation Environment (DIASE). It allows deploying, validating and monitoring the DIAFORUS network and is meant to be used on a tablet. Screenshots of the environment and how it operates can be found in Appendix D.

B.2 Efficient spectrum sharing

We developed four projects when researching efficient spectrum sharing, found in Chapter 4. They are currently in an experimental state but we are slowly improving on them as time permits to make it possible for us to implement the proposed PHY/MAC protocols.

The source code of the following projects is available publicly [89] under the GPLv2 license.

B.2.1 GR-GTSRC & Spectrum-viz

This project started as a way to analyse the spectrum in real-time to fill the Radio Event Table introduced in Chapter 4. The code is implemented as a GNU-Radio block that would instantiate decoders and encoders on the flight to respectively demodulate incoming signals and modulating outgoing messages.

It also contains a visualisation tool called *Spectrum-viz* which allows debugging and demonstrating the abilities of *GR-GTSRC*. This project was used during our mid-term demonstration for the ANR (the French research agency that funded the Licorne project).

B.2.2 Hachoir_UHD

Hachoir_UHD is a simplified version of the *GR-GTSRC* GNU-radio block that allowed us to experiment with PHY-parameters detection and real-time communications. The latter has been done using two bladeRF, a PSK modulation and Linux's network stack to generate the messages. Frames were received and injected in the network stack by using a TUN network interface.

It has also been used as a basis for a research project on creating a box capable of centralising the data coming from unknown sensors in its surrounding. The identification of the sensor is done using PHY-layer parameters along with link-layer frame decoding.

This project is the most usable one and can often be used successfully to analyse low-speed communications such as the ones from car keys or smart meters.

B.2.3 Non-real-time PHY simulator

This simulator has been developed to evaluate our PHY-layer signalling protocol and generate the figures found in Chapter 4. Its main interest is that it simulates the network as fast as possible and not in real-time in order to make it possible to run the simulation millions of time with random parameters to get meaningful results.

B.3 Defining an autonomic low-power node

In this section, we introduce the three main projects we have developed when working on the hardware-side of power management. The source code for each project will be given in the following subsections.

B.3.1 Nouveau

We participated to the Open Source driver for NVIDIA cards developed mostly without the help from NVIDIA, written in C, licensed under the MIT/X11 Open Source license and part of the Linux kernel.

We believe having a production-ready Open Source platform for NVIDIA GPUs is important for research as it allows full access to the GPU and also enables performance comparisons with the proprietary driver. Our participation mainly concerns thermal management, fan management, vbios parsing, PDAEMON and clock tree reclocking. Since 2010, this amounts to 90 patches in the kernel tree.

The source code is available in Linux [177] or in an out-of-the-tree version [178] that can be run as a process instead of a kernel driver.

B.3.2 Envytools

The envytools project has been started by Nouveau developers to study the hardware and analyse traces of the proprietary driver. It allows documenting the hardware in both machine-readable and human-readable form. The project is located on github [97] while the human-readable documentation can be found on readthedocs [179].

Our contribution to this project are mostly focused in vbios management tools (nvagetbios, nvafakebios and nvbios), performance counters documentation and automated reverse engineering tools (nvacounter), clocks and memory timings (nvatimings and nvamemtimings) and i2c transaction spy (nvaspyi2c). We also documented PTHERM in great length on the Tesla generation along with writing hardware tests to verify the result of the reverse engineering more easily.

B.3.3 PDAEMON tracing

In Figures 5.24, 5.25, 5.26 and 5.27, we analysed the reclocking policy implemented by NVIDIA. Creating these figures required to track multiple parameters at the same time such as the temperature, the frequency of every clock, the FSRM state and the power consumption. We also needed to generate fake loads to evaluate their impact on the configuration of the GPU.

We developed such tools and released them under a GPLv2 license. The source code can be found publicly [180] although it will likely require modifications to run on your hardware.

B.4 Making real-time power management decisions

B.4.1 RTGDE

We implemented the decision engine proposed in Chapter 6 along with multiple unit tests, gnuplot examples and test scenarios.

The project has been written in C and has been tested on modern Linux distributions. It should also be portable to other Operating Systems without problems.

The source code is available publicly [165] under the MIT license.

Appendix C

Diaforus : An example deployment file

The following code listing is an example of deployment XML file we talked about in Chapter 3. This is an hybrid deployment containing both simulated and real wireless nodes. The simulated node run the area-level reasoning code while the real nodes host sensors.

Listing C.1: deployment.xml

```
1 <?xml version='1.0' encoding='utf-8'?>
2 <!DOCTYPE network SYSTEM '../network.dtd'>
3 <network failure_handling="false" pubsub_reliable="true">
4     <build>
5         <!-- path: filepath to the make binary (optional)
6              args: parameters you can specify for make (optional)
7              root: the directory that contains the Makefile
8              output: the binary file generated by the project
9         -->
10        <make output="diaforus_final_code" root="build/" target="host"
              args="-j8" />
11        <make output="diaforus_final_code.jtag.s3" root="Debug/"
              target="aps3" args="-j1" />
12        <debug pubsub="true" uip="false" sicslowpan="false" rpl="false
              " wavesim="false" coap="false" reasoning="true" />
13        <application name="REASONING" />
14    </build>
15
16    <coap_groups>
17        <group name="reasoning1">
18            <resource name="intrusion_duration" />
19            <resource name="latency_mode" />
20        </group>
21        <group name="reasoning2">
22            <resource name="hist_analyze_per" />
23            <resource name="criticality_lvl" />
```

```
24              <resource name="history"/>
25              <resource name="alert_alarm_ratio"/>
26          </group>
27          <group name="rpl">
28              <resource name="rpl_neighbors"/>
29              <resource name="rpl_rank"/>
30              <resource name="rpl_num_routes"/>
31              <resource name="rpl_num_tra_mess"/>
32              <resource name="rpl_num_rec_mess"/>
33          </group>
34          <group name="allnodes">
35              <resource name="num_tra_packets"/>
36              <resource name="num_rec_packets"/>
37              <resource name="nodeid"/>
38          </group>
39      </coap_groups>
40
41      <!-- Zone 1 -->
42      <node target="host" simulation="true" id="20" area="1" name="node_
            root_reasoning_1">
43          <firmware name="node_20"/>
44          <phy x="910" y="865" range="900"/>
45          <energy startup="1600" alarm="100"/>
46          <rpl>
47              <restrict_neighbors>
48                  <neighbor node_id="1"/>
49                  <neighbor node_id="2"/>
50                  <neighbor node_id="3"/>
51                  <neighbor node_id="4"/>
52              </restrict_neighbors>
53          </rpl>
54          <pubsub enable="true"/>
55          <coap enable="true">
56              <group name="reasoning2"/>
57              <group name="rpl"/>
58              <group name="allnodes"/>
59          </coap>
60          <!-- The parameters to control the reasoning (level 1 and
                level 2) -->
61          <reasoning latency_mode="green" max_intrusion_duration="10"
                min_intrusion_duration="5">
62              <!-- This node is the reasoning node -->
63              <reasoning_node log_analyse_period="1440"/>
64          </reasoning>
65          <sensors/>
66      </node>
67
68      <node target="aps3" simulation="false" id="2" area="1" name="node_
            pir_2-1">
69          <firmware name="node_2.jtag.s3"/>
```

```
70          <phy x="836" y="813" range="900"/>
71          <energy startup="1600" alarm="100"/>
72          <rpl>
73              <restrict_neighbors>
74                  <neighbor node_id="1"/>
75              </restrict_neighbors>
76          </rpl>
77          <pubsub enable="true"/>
78          <coap enable="true">
79              <group name="reasoning1"/>
80              <group name="rpl"/>
81              <group name="allnodes"/>
82          </coap>
83          <reasoning latency_mode="green" max_intrusion_duration="10"
                min_intrusion_duration="5"/>
84          <parameter value="NO_DEMOBOARD" name="HW_MODALITY"/>
85          <parameter value="-120" name="M_FREQ_OFFSET"/>
86          <sensors>
87              <modality type="PIR">
88                  <sensor period="200" reemission_delay="3001" type="
                        digital" id="0" delay="0">
89                      <gpio pin="2"/>
90                      <value abs_threshold="1" rel_threshold="1" invert=
                            "false"/>
91                      <position x="763" y="785" z_rotation="-110"/>
92                  </sensor>
93              </modality>
94          </sensors>
95      </node>
96
97      <node target="aps3" simulation="false" id="3" area="1" name="node_
            pir_3-1">
98          <firmware name="node_3.jtag.s3"/>
99          <phy x="956" y="799" range="900"/>
100         <energy startup="1600" alarm="100"/>
101         <rpl>
102             <restrict_neighbors>
103                 <neighbor node_id="1"/>
104             </restrict_neighbors>
105         </rpl>
106         <pubsub enable="true"/>
107         <coap enable="true">
108             <group name="reasoning1"/>
109             <group name="rpl"/>
110             <group name="allnodes"/>
111         </coap>
112         <reasoning latency_mode="green" max_intrusion_duration="10"
                min_intrusion_duration="5"/>
113         <sensors>
114             <modality type="PIR">
```

```
115                 <sensor period="200" reemission_delay="3001" type="
                        digital" id="0" delay="0">
116                     <gpio pin="2" />
117                     <value abs_threshold="1" rel_threshold="1" invert=
                            "false" />
118                     <position x="905" y="738" z_rotation="-110" />
119                 </sensor>
120             </modality>
121         </sensors>
122     </node>
123
124     <!-- Zone 2-->
125     <node target="host" simulation="true" id="1" area="2" name="node_
            root_reasoning_2">
126         <emblem source="c2.png" />
127         <firmware name="node_1" />
128         <phy x="1138" y="863" range="900" />
129         <energy startup="1600" alarm="100" />
130         <rpl prefix="1180:0000:0000:0000">
131             <restrict_neighbors>
132                 <neighbor node_id="20" />
133                 <neighbor node_id="2" />
134                 <neighbor node_id="3" />
135                 <neighbor node_id="4" />
136                 <neighbor node_id="21" />
137                 <neighbor node_id="22" />
138             </restrict_neighbors>
139         </rpl>
140         <pubsub enable="true" broker="true" />
141         <coap enable="true">
142             <group name="reasoning2" />
143             <group name="rpl" />
144             <group name="allnodes" />
145         </coap>
146         <gateway port="1235" />
147         <!-- The parameters to control the reasoning (level 1 and
                level 2) -->
148         <reasoning latency_mode="white" max_intrusion_duration="10"
                min_intrusion_duration="5">
149             <!-- This node is the reasoning node -->
150             <reasoning_node log_analyse_period="1440" />
151         </reasoning>
152         <sensors />
153     </node>
154
155     <node target="aps3" simulation="false" id="21" area="2" name="node
            _pir_1-2">
156         <firmware name="node_21.jtag.s3" />
157         <phy x="1174" y="632" range="900" />
158         <energy startup="1600" alarm="100" />
```

```
159         <rpl>
160             <restrict_neighbors>
161                 <neighbor node_id="1" />
162             </restrict_neighbors>
163         </rpl>
164         <pubsub enable="true" />
165         <coap enable="true">
166             <group name="reasoning1" />
167             <group name="rpl" />
168             <group name="allnodes" />
169         </coap>
170         <reasoning latency_mode="white" max_intrusion_duration="10"
                min_intrusion_duration="5" />
171         <parameter value="NO_DEMOBOARD" name="HW_MODALITY" />
172         <parameter value="-240" name="M_FREQ_OFFSET" />
173         <sensors>
174             <modality type="SEISMIC">
175                 <sensor period="200" reemission_delay="5001" type="
                        digital" id="0" delay="0">
176                     <gpio pin="3" />
177                     <value abs_threshold="1" rel_threshold="1" invert=
                            "false" />
178                     <position x="1121" y="614" z_rotation="80" />
179                 </sensor>
180             </modality>
181         </sensors>
182     </node>
183 
184     <node target="aps3" simulation="false" id="22" area="2" name="node
            _pir_2-2">
185         <firmware name="node_22.jtag.s3" />
186         <phy x="1339" y="568" range="900" />
187         <energy startup="1600" alarm="100" />
188         <rpl>
189             <restrict_neighbors>
190                 <neighbor node_id="1" />
191             </restrict_neighbors>
192         </rpl>
193         <pubsub enable="true" />
194         <coap enable="true">
195             <group name="reasoning1" />
196             <group name="rpl" />
197             <group name="allnodes" />
198         </coap>
199         <reasoning latency_mode="white" max_intrusion_duration="10"
                min_intrusion_duration="5" />
200         <parameter value="WITH_SPIRIT" name="HW_MODALITY" />
201         <parameter value="-240" name="M_FREQ_OFFSET" />
202         <sensors>
203             <modality type="SPIRIT">
```

```
204                     <sensor period="500" reemission_delay="3001" type="
                            digital" id="0" delay="0">
205                         <gpio pin="0"/>
206                         <value abs_threshold="1" rel_threshold="1" invert=
                                "false"/>
207                         <position x="1273" y="585" z_rotation="0"/>
208                     </sensor>
209                 </modality>
210             </sensors>
211         </node>
212 </network>
```

Appendix D

DiaSE : DIAFORUS' Simulation Environment

Contents

In this appendix, we demonstrate the capabilities of DiaSE, the simulation environment for the DIAFORUS network which has been presented in Chapter 3.

D.1 Deployment

The first view, shown in Figure D.1 allows deploying a network with the mouse. Nodes should be added by selecting the antenna icon and clicking where it is wanted on the map.

Likewise, adding sensors is done by selecting the wanted type, clicking on the map where it should be deployed, orient the sensor by moving your mouse before clicking again. Connecting sensors to their node is done by changing the nodeID parameter of the sensor.

Emblems can be assigned to nodes to better-represent their function. For instance, in this figure, the gateway and the alarm have emblems while the other nodes use the default antenna icon.

It is possible to visualise the connectivity of a node by putting the mouse's cursor on top of the node. It is also possible to see the whole network connectivity by telling so in the *Display* menu.

Figure D.1: Deploying a network using Diase

D.2 Running the network

The *Network* menu contains the different modes supported by Diase (fully-simulated, hybrid or fully hardware). The menu can be seen in Figure D.2.

Figure D.2: Running options for the network (building, running the gateway, running a simulation)

Before running the network, it is necessary to build it. To do so, press the *Build* option in the *Network* menu. The result is shown in Figure D.3.

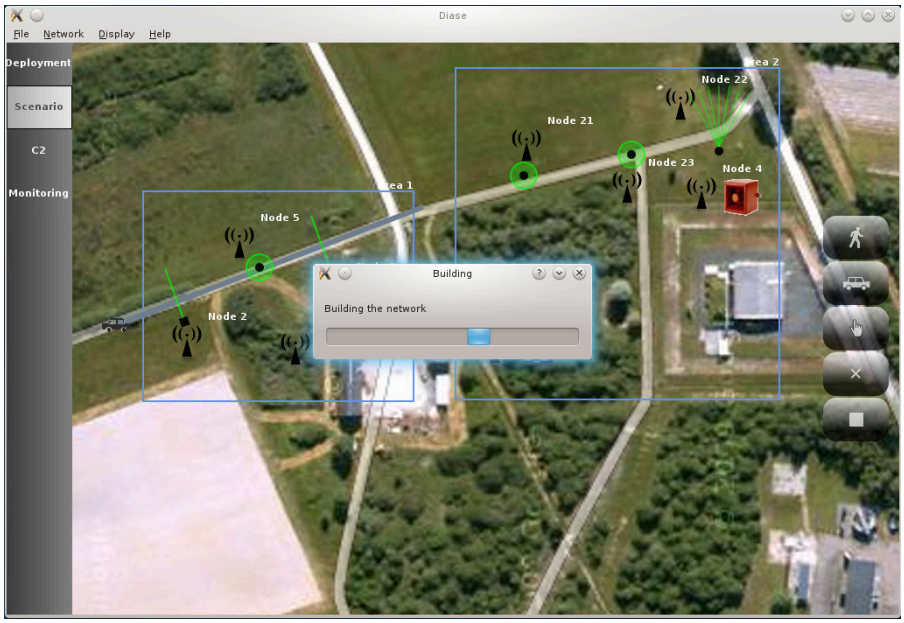


Figure D.3: Building the network

We further demonstrated how to use Diase by running a fully-simulated network and then opening the *Nodes output* console. In Figure D.4, we show the logs produced by our physical-layer simulation.

We then show in Figure D.5 the debug output of a correlation node.

D.3 Scenario

Now that the network has been compiled and is running, we can simulate different kind of intruders (cars and pedestrians).

The user has to create a path using his/her mouse by first clicking on the hand icon then left-clicking to create the different segments of the intrusion path. We the path is finished, the user should press the right button of the mouse.

Once a path has been created, the user can attach a pedestrian or a car to it by selecting the corresponding icon and then clicking at the beginning of the path it should be assigned onto.

The car simulates a fast intrusion and is shown in Figure D.6.

The pedestrian simulates a slow intrusion and is shown in Figure D.7.

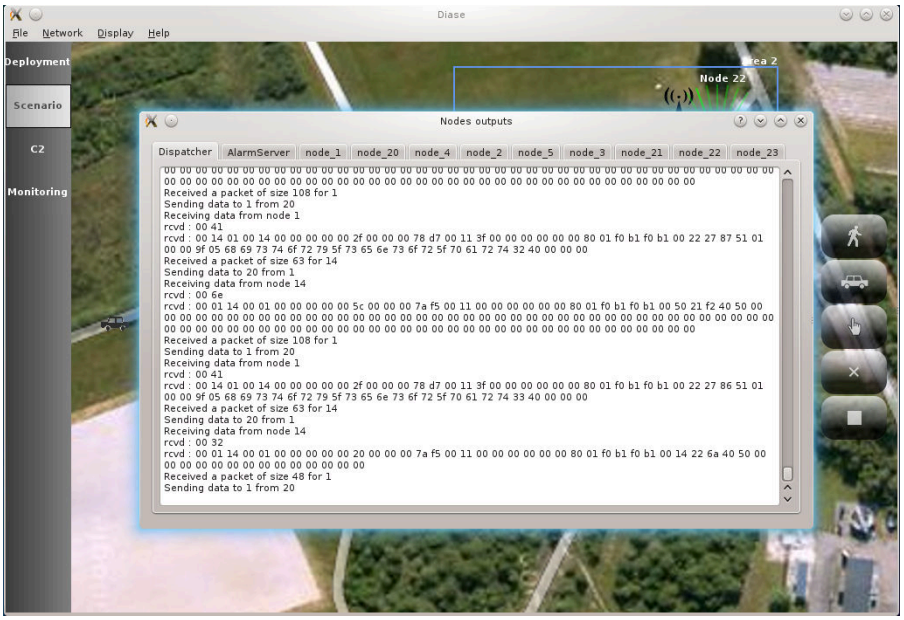


Figure D.4: Simulating the communication medium with our dispatcher

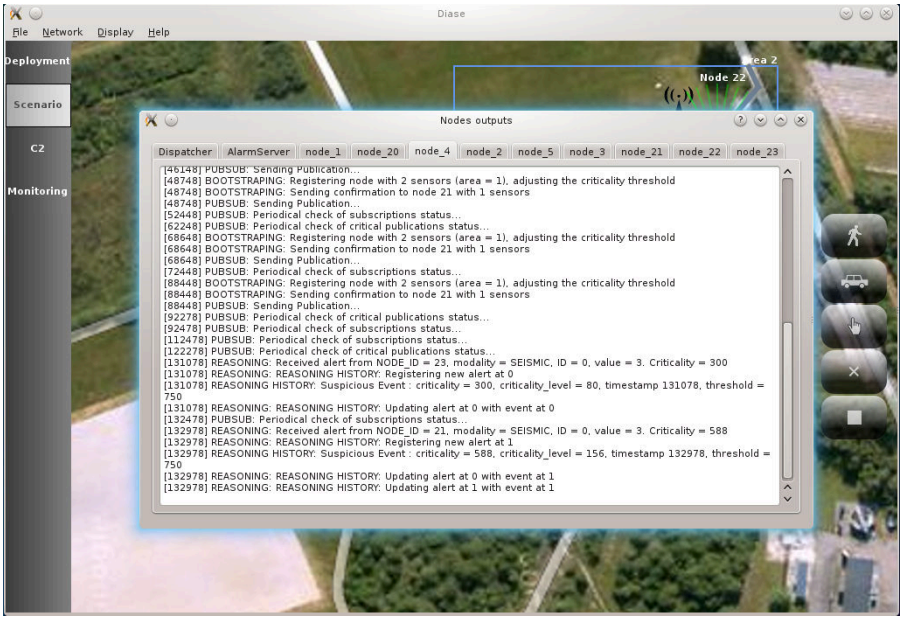


Figure D.5: Simulating a correlation node

D.4 C2 : Command & Control

One of the most important panel of the Diase environment is the Command & Control (C2) mode.

Figure D.6: Simulating the intrusion of a car in the DIAFORUS network

Figure D.7: Simulating the intrusion of a pedestrian in the DIAFORUS network

In this mode, the network operator can visualise the current state of alarm of all the areas by checking if it is drawn in red or not. The sensors that contributed to the past alarms are also drawn in red.

To acknowledge an intrusion, the operator needs to click in the area he/she wants to acknowledge. This resets the state of the area.

Figure D.8: Command & Control mode : Visualisation of an alarm in an area along with which sensors participated in it

The past intrusions can be found in the *Alarms logs*. They contain the time at which it happened, the area concerned and the list of sensors involved. Finally, it also contains the intrusion time which is computed as the difference between the first sensor that detected the intrusion and the last one that did. This enables the operator to get a sense of the speed of the intrusion and allow him/her to react appropriately.

The C2 mode can be seen in Figure D.8. In this figure, we can see that the area 2 is reporting an intrusion thanks to its red color. The sensors that contributed to the intrusion are nodes 21 and 23. The alarms log shows that multiple alarms have been received for the same intrusion because the pedestrian that is currently in the area walks very slowly and because the sensors are not able to calculate the distance between the sensor and the intruder(s).

D.5 Monitoring

Diase allows the network operator to inspect the state of every node in the network to check what is going on. To do so, Diase uses the REST protocol CoAP to retrieve the resources and display them.

In Figure D.9, we can see the left panel allows monitoring a new element. In this figure, we can see how the operator can select which node he/she wants.

In Figure D.10, we can see how the operator can select the resource he/she is interested in displaying before being asked whether to display it in a textual or graphical representation (not shown in any screenshot). The list of resources is not directly exported

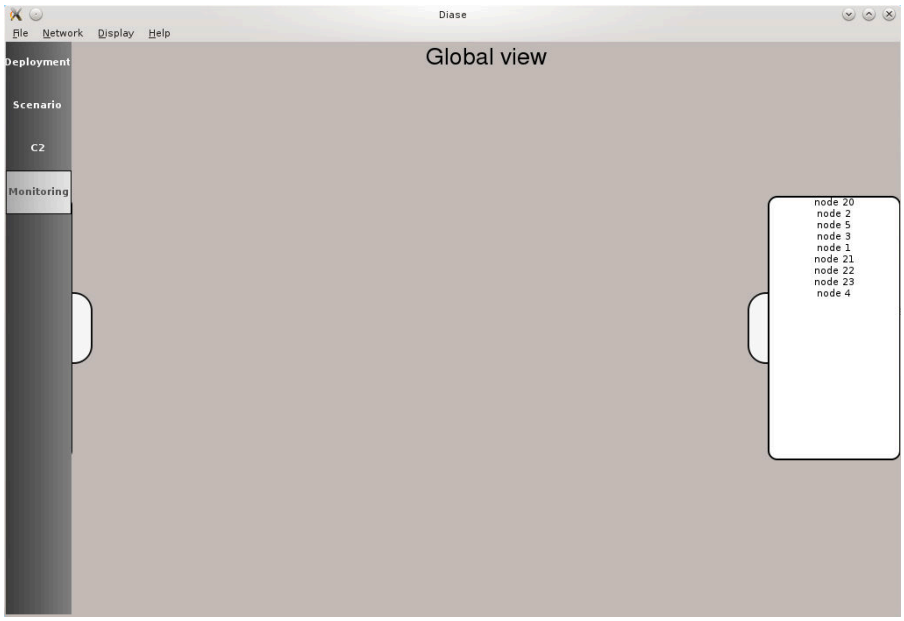


Figure D.9: Monitoring mode : Listing the nodes that export CoAP resources

by the node, instead, the node only reports which group of resources is available. The content of this group can be found in the network xml file (Appendix C).

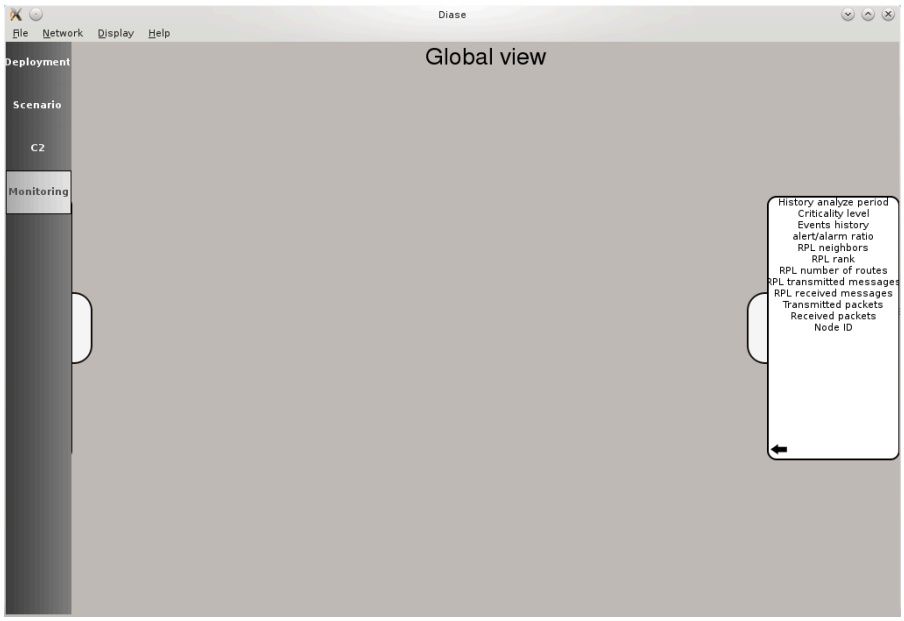


Figure D.10: Monitoring mode : Selecting which resource should be added to the current monitoring view

The way the resource should be presented is defined in a local file called diase.ini which allows the operator to specify the format of the CoAP resource, how to display it and how often it should be polled. An example can be seen in the listing *diase.ini*.

Listing D.1: diase.ini

```
[intrusion_duration]
name="Avg. intrusion duration"
type=short
drawing=text
interval=10000

[history]
name="Events history"
type=multipart
multipart\1\keys=multipart[1].type[0]
multipart\1\values=multipart[1].type[1],multipart[2].type[0],multipart
    [2].type[1]
multipart\2\name=history_part1,history_part2
multipart\2\type=int,byte
multipart\3\name=history_sensor_part1,history_sensor_part2,
    history_sensor_part3
multipart\3\type=short,byte
multipart\size=3
drawing=plugin::bargraph::history
interval=5000

[rpl_num_routes]
name="RPL number of routes"
type=byte
drawing=text
interval=10000

[rpl_num_tra_mess]
name="RPL transmitted messages"
type=shortarray
shortarray\1\name="DIO"
shortarray\2\name="DAO"
shortarray\3\name="DIS"
shortarray\4\name="DAO/ACK"
shortarray\size=4
drawing=text bargraph
interval=10000
```

In Figure D.11, we can see different resources shown in both a textual and graphical representation. The graphical representation is interactive and can be zoomed in and out while also allowing scrolling back and forth in time.

In Figure D.12, we can see two examples of graphical resources as displayed by Diase.

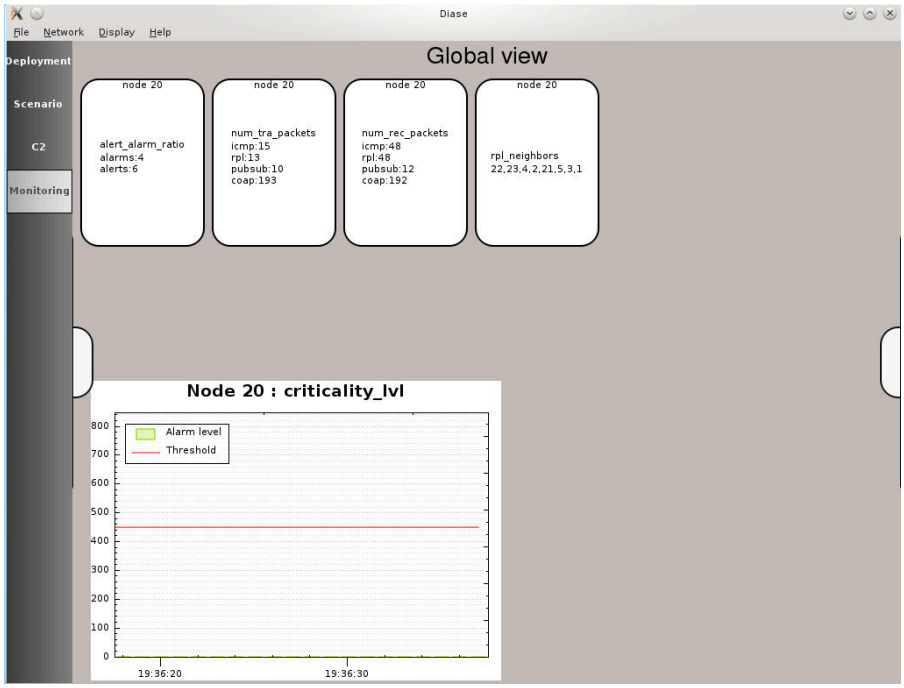
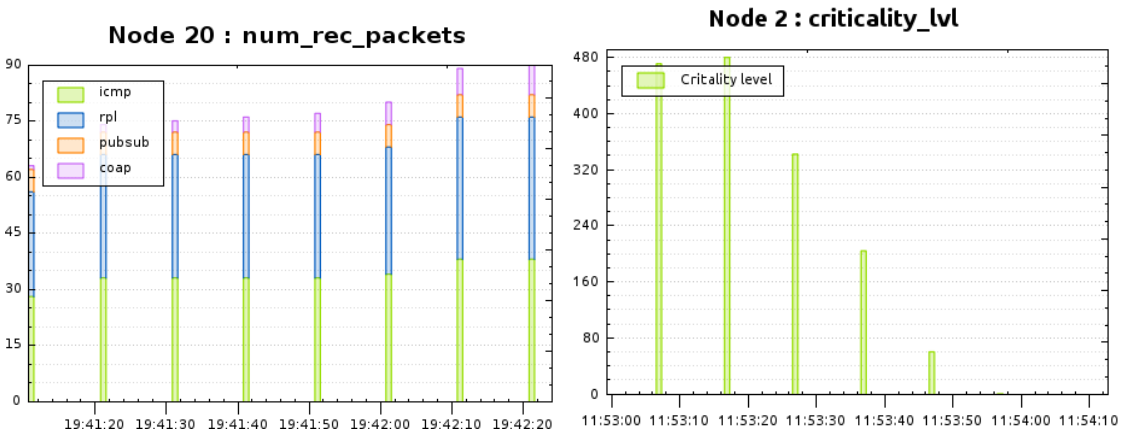


Figure D.11: Monitoring mode : Example of a monitoring view for a node with both textual and graphical representations



(a) The number of type of packets received **(b)** Evolution of the criticality level of Node 2

Figure D.12: Two examples of the real-time graphical representation of the CoAP resources in Diase

Appendix E

Real-life deployment of the DIAFORUS Network

The results presented earlier in Chapter 3 have been obtained using a simulated network. However, in the DIAFORUS ANR project, we ported this network on real nodes and validated the simulation results. The algorithms used for the simulation are the ones used on the real nodes. This means our proposal is feasible on standard sensor nodes.

E.1 Hardware used

Sensors and actuators used

We used two military sensors from Thales[1] because they were one of our industrial partners in this research project along with Coronis[2] which provided the sensor nodes.

In our real-life testing, we used 4 types of sensors:

- an infrared unidirectional barrier from Thales (Figure E.1a);
- a 60° infrared barrier called SPIRIT from TechNext (Figure E.1c);
- a seismic sensor from Thales (Figure E.1b);
- a manual switch, for simulating a seismic sensor on tabletop demos.

The only actuator used during this demonstration was a buzzer to simulate an alarm when the correlation node would detect an intrusion.

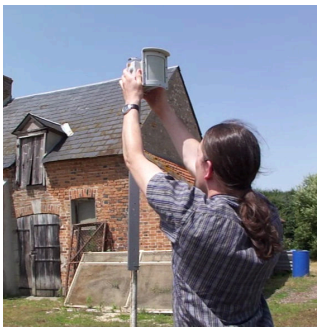
Sensor nodes

For our real-life test, we used the sensors nodes developed by Coronis. These nodes contain an APS3 core from Cortus [181], a 32 bit micro-controler with an Harvard architecture oriented towards power efficiency and small code size. We had 4 kB of RAM and 48 kB left for the code which could also hold some data although with a performance hit. The node is visible in Figure E.2.

[1] http://www.thalesgroup.com/en
[2] http://www.coronis.com

(a) Deploying a Thales infrared barrier

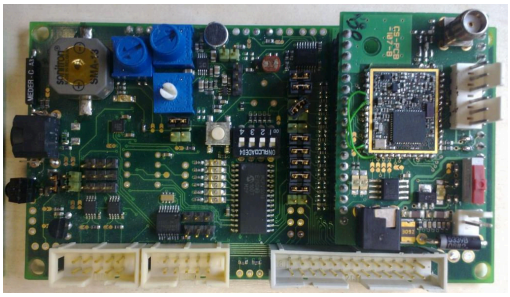
(b) Deploying a Thales seismic sensor



(c) Deploying a 16-beam infrared sensor

Figure E.1: Deployment of some sensors for protecting a farm in a military field

(a) Four sensor nodes with an unidirectional infrared barrier attached



(b) A close-up on the sensor node, when attached to its demonstration board

Figure E.2: Deployment of some sensors for protecting a farm in a military field

The node's transceiver implements the Wavenis/wave2m standard [37]. It can operate on the 433, 868, 915 and 2400 MHz bands. Its bitrate ranges from 4.8 to 100 kbps. It's modulation is a GFSK and it also supports Frequency Hopping Spread Spectrum (FHSS) to increase the link reliability.

E.2 Tested scenarios

We tested the versatility of our proposition using 6 operational scenarios proposed by Thales Telecommunications. The videos of our deployment scenarios are available on Youtube[1].

E.2.1 A - Short-loop response

Operational need Detecting an intrusion and triggering an automatic response without the intervention of the operator. The operator is however warned about the intrusion.

[1] https://www.youtube.com/user/ANRDIAFORUS/videos?sort=da&flow=grid&view=0

Deployment Two unidirectional infrared barriers are deployed along a path leading to a restricted area. Another node receives the information about the intrusion and automatically sounds an alarm.

Result When a simulated pedestrian walks on the restricted path and crosses the first infrared barrier, nothing happens. When he/she continues and crosses the second barrier, an alarm sounds a few seconds later without the intervention of the operator. The operator however received an alarm on Diase, in the Command & Control (C2) mode. The result can be seen in Figure E.3 and on Youtube, at http://youtu.be/oVN-ncoBVNE.

Figure E.3: Testing the short-loop scenario on real sensor nodes

E.2.2 B - Inter-area communication

Operational need Some areas may not be of immediate interest to an operator because they are often used by civilians. The situation of these area is however useful to increase the vigilance-level of an adjacent area that has a restricted access. The situation of adjacent zone thus allows an area to detect an intrusion faster by requiring less intra-area correlation while still limiting false positives.

Deployment We deployed two areas, an area with a restricted access and an area of interest with no access control. The restricted-access area has two sensors, a SPIRIT and a simulated seismic sensors. The area of interest has two infrared barriers.

Result At first, a pedestrian needs to cross both sensors in the area of interest to trigger an alarm. But if the pedestrian tripped both sensors of the area of interest, then he/she will be detected after tripping only one sensor in the restricted-access area. The result can be seen in Figure E.4 and on Youtube, at http://youtu.be/y8uvsAPZ3zw.

E.2.3 C - Per-area sensitivity settings

Operational need Areas further from the zone to protect are not as latency-sensitive as areas closer to the zone to protect. The further an area is from the zone to protect,

Figure E.4: Testing the inter-area communication to decrease the intrusion detection latency

the more correlation it can operate before sending an alarm to the operator to limit false positives. On the contrary, areas closer to the protected zone should require less correlation to lower the detection latency at the expense of false positives.

Deployment We deployed two areas, an area in the protected zone and an area of interest. The protected area has three sensors: two infrared barriers and a simulated seismic sensor. The area of interest has two simulated seismic sensor and a SPIRIT.

Result A pedestrian crossing the area of interest needs to trip the two seismic sensors and the SPIRIT before an alarm is sent. In the protected area, only two sensors are necessary before an alarm is sent to the operator. The result can be seen in Figure E.5 and on Youtube, at http://youtu.be/ws_s0SvBae8.

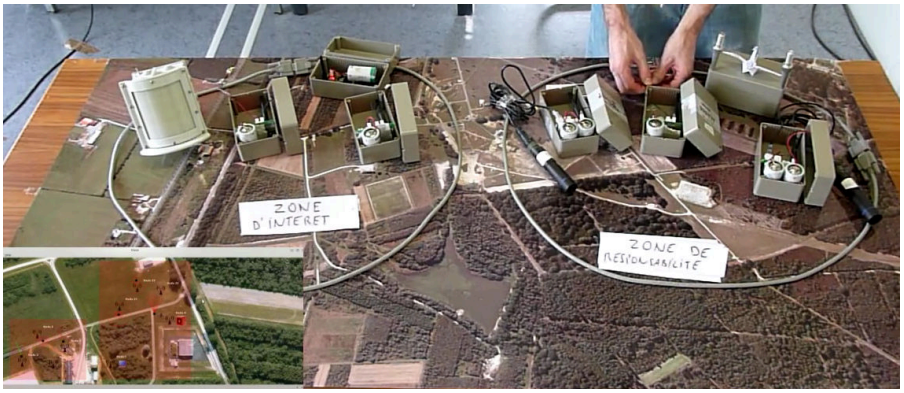


Figure E.5: Testing different area sensitivities

E.2.4 D - In-network network notification

Operational need An operator may be in an area and would like to configure it, read its history or receive alarms when they arrive using a laptop or a tablet. This operator would however not be interested in getting alarms from other areas.

Deployment Two areas are deployed both containing multiple sensors. The main operator can access alarms and monitor nodes from both areas using CoAP. The mobile operator only receives alarms from the zone he/she is currently located in.

Result A simulated pedestrian penetrates the first area and trips enough sensors to trigger an alarm. Only the main operator receives the alarm. When the pedestrian enters the second area and trips enough sensors, both operators receive the alarm. The result can be seen in Figure E.6 and on Youtube, at http://youtu.be/OkNqI5PYujc.

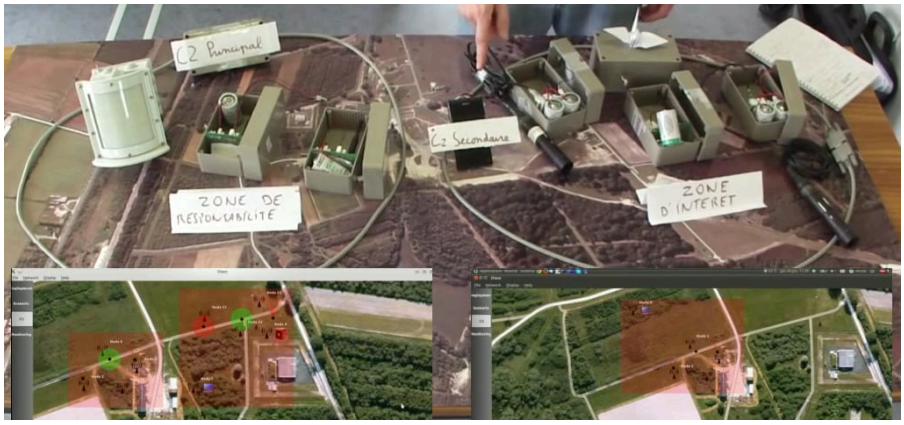


Figure E.6: Testing different area sensitivities

E.2.5 E - Fault tolerance

Operational need Wireless nodes can fail at any time. When they do, the network should also autonomously react to this loss and reconfigure itself to operate properly. This includes finding other communication routes and changing the criticality threshold of the area to react to the loss of the sensors attached to the failing node. Finally, the operator needs to be warned about this failure.

Deployment We only deployed two infrared barriers. One of the barrier (node 3) is connected to the other one (node 2) to reach node 1, the gateway and correlation node. Node 3 cannot reach node 1 and has to go through node 1 or node 7 first.

Result An alarm is generated when both sensors are tripped by a simulated a pedestrian intrusion. Node 2 is then disabled and another intrusion is simulated. An alarm is triggered after tripping node 3's barrier. This means node 3 reconfigured its route to reach the correlating node by going through node 7 instead of node 2. It also means that the correlation node reconfigured itself to take into account the loss of one sensor. The operator is advertised about the loss of node 2 by turning the node's symbol from black to red. The result can be seen in Figure E.7 and on Youtube, at http://youtu.be/QBxLTj57El8.

(a) Before node 2 failed **(b)** After node 2 failed

Figure E.7: Result of an intrusion before and after node 2 failed

E.2.6 F - Protecting a civilian farm

Operational need The deployment of the wireless sensor network should be as easily as possible. The final demonstration scenario aims at protecting a farm to alert the police in case of an intrusion.

Deployment We installed two areas in the vicinity of the farm to be protected. One area containing two infrared barriers is located on the pathway leading to the farm. The second area, in front of the farm, contains one seismic sensor and a SPIRIT.

Result We see an operator installing the network by taking advantage of the terrain. The configuration and flashing of the nodes was done off-camera using the Diase and the build_network firmware-generation script. Then, a pedestrian runs to the farm along the pathway leading to the farm, tripping both barriers. He then is picked up by the SPIRIT of the second area before being detected by the seismic sensor. By the time he reaches the farm, an alarm is sounded, deterring the intruder. The result can be seen in Figure E.8 and on Youtube, at http://youtu.be/Nfa31KqW4F8.

Figure E.8: Final deployment of the DIAFORUS system to protect a farm

Appendix F

Demodulating unknown signals

In Chapter 4, we introduced the idea that it should be possible to demodulate signals in real time without prior-knowledge of their PHY parameters. In this appendix, we show that we started working on such a system and already have promising results.

F.1 Studying an OOK signal

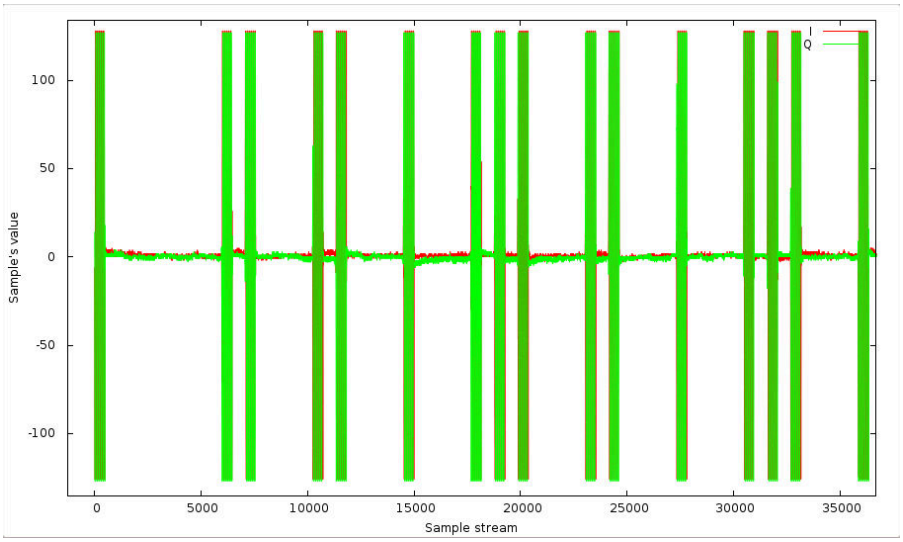


Figure F.1: Exemple of a received OOK signal

In Figure F.1, we can see the OOK modulation that is often used by car keys and remote controls. In this case, the signal got generated by Chacon's DI-O, a remote control to turn on or off up to 3 power plug. The signal has been decoded in real time and the output of our system is the following:

```
5: new communication, time = 1570 µs, len = 143205 samples, len_time = 71602 µs,
  sub-burst = 66, avg_pwr = 218% above threshold (31.432 vs noise mag max 3.602)
OOK: Unknown OFF symbol 5475
```

```
New message: modulation = 'OOK, ON(0 bps) = { 491.9 (p=1.00), 0.0 (p=0.00) },
  OFF(1 bps) = { 612.7 (p=0.49), 2676.8 (p=0.49) },
  freq = nan MHz (chan = 0)', sub messages = 1, process time = 30971 µs
Sub msg 0, OOK, ON(0 bps) = { 246.0 µs, -1.0 µs },
  OFF(1 bps) = { 306.3 µs, 1338.4 µs },
  STOP = { -1.0 µs }, freq = 433.920 MHz (chan = 35): len = 64:
BIN: 0101 1001 0110 0110 0110 1001 1010 1010 1001 0110 0101 0110 1001 0101
     0101 0101
HEX: 59 66 69 aa 96 56 95 55
Manchester code detected:
BIN: 0010 0101 0110 1111 1001 0001 1000 0000
HEX: 25 6f 91 80
```

The communication started 1.5 ms after we started listening and lasted 143205 samples which is equivalent to 71.6 ms. It is composed of 66 bursts of energy while the received power was about 10 times superior than the average noise power and 2 times higher than the power threshold.

A message has been decoded from those 66 bursts. The modulation is indeed a OOK. The ON time is constant (491.9 samples in average) which means that it is not coding any information. The OFF time is however coding 1 bit per symbol as the time can either be 612.7 or 2676.8 samples. The probability of having one time off or the other is however the same which suggests that a Manchester code is being used. A OFF symbol was however unrecognised as it lasted 5475 samples. Based on Figure F.1, we can guess that it is a start bit. No stop bit got detected as the message did not repeat itself. The frequency at which the message has been sent is 433.92 MHz which is equivalent to the channel 35 in this frequency band. The actual len in bit of the received data is 64 bytes and the time it took to demodulate the frame was 31 ms. Even though the program has been run in Debug mode, it still takes less time to demodulate the message than it took to receive it.

The received data is first shown in binary before showing an hexadecimal version. As the received data only contains 01 and 10 sequences, our program detects that the message has been sent with a Manchester code. It then displays the decoded message without the Manchester code.

Using multiple frames, we managed to identify the meaning of every bit of the transmission and managed to understand which button was pressed on the remote control that generated the signal.

F.2 Studying a PSK signal

We then demonstrate the output of a 2-PSK modulation, as received by a bladerf very close to the emitter. The received signal can be seen in Figure F.2 with the decoded information available right after it.

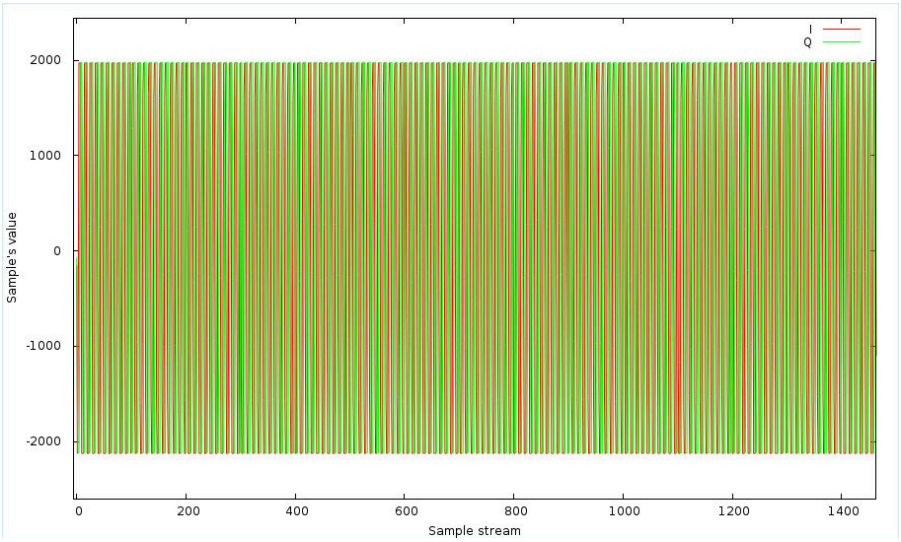


Figure F.2: Exemple of a received 2-PSK signal

```
0: new communication, time = 1015 µs, len = 77876 samples, len_time = 77876 µs,
  sub-burst = 2, avg_pwr = 4116% above threshold (2134 vs noise mag max 12.96)
New message: modulation = '2-PSK: freq offset = 99.6 kHz, symbol time = 100 µs',
sub messages = 1, process time = 26537 µs
Sub msg 0: len = 369:
BIN: 0000 0000 0000 0000 0000 1000 0000 0110 1111 1111 1111 1111 1111 1111 1111
     1111 1111 1111 1111 1111 1010 1110 0010 1010 0100 0100 0001 0101 1111 0010
     1001 1001 0000 1000 0000 0110 0000 0000 0000 0001 0000 1000 0000 0000 0000
     0110 0000 0100 0000 0000 0000 0001 1010 1110 0010 1010 0100 0100 0001 0101
     1111 0010 1001 1001 1100 0000 1010 1000 0000 0010 0000 0001 0000 0000 0000
     0000 0000 0000 0000 0000 0000 0000 0000 0000 1100 0000 1010 1000 0000 0010
     0000 0010 0
HEX: 00 00 08 06 ff ff ff ff ff ff ae 2a 44 15 f2 99 08 06 00 01 08 00 06 04 00
     01 ae 2a 44 15 f2 99 c0 a8 02 01 00 00 00 00 00 00 c0 a8 02 02 (0)
```

Contrarily to the previous signal, the modulation used for this signal is detected as being 2-PSK with a symbol rate of 10 000 symbols per second. The frequency offset between the received transmission and the SW radio's central frequency was 99.6 kHz. This suggests that the clocks of both radios are not perfectly running at the same speed.

The processing time of the frame was 26.5 ms, still much shorter than the 77.9 ms it took for the transmission to finish.

The received data is actually an ARP request from the MAC address AE:2A:44:15:F2:99 to the broadcast destination (FF:FF:FF:FF:FF:FF) to find the IP address 192.168.2.1 with the source IP address is 192.168.2.2.

F.3 Conclusion

We showcased our decoding system using frames sent using two different modulations. The system was able in both cases to detect without any prior knowledge the modulation type, find the symbols, which ones are coding data, the symbol rate and even if the message was using a code or not.

Our system is however not very mature yet but we are working on improving its features to be more efficient and more robust in low-SNR scenarios.

Appendix G

A ring buffer data structure suited for software-defined radios

The following listing defines a ring-buffer data structure that is suited for software-defined radios. It allows its users to annotate the sample stream with the parameters of the radio to enable dynamic switching of the radio's parameters while still keeping the current data available.

It also enables its user to synchronise the time and the frequency domain thanks to the global index given to every sample of the stream which allows an FFT to reference the samples used for its calculation.

More information can be found in the comments of the data structure.

Listing G.1: ringbuffer.h

```
1  /**
2   * \file       ringbuffer.h
3   * \author     MuPuF (Martin Peres <martin.peres@labri.fr>)
4   * \version    1.0
5   * \date       07 Avril 2013
6   */
7
8  #ifndef RINGBUFFER_H
9  #define RINGBUFFER_H
10
11 #include <boost/thread/mutex.hpp>
12
13 #include <stddef.h>
14 #include <atomic>
15 #include <list>
16
17 /**
18  * \class    RingBuffer
19  * \brief    Defines a template-based ring buffer container that never blocks
20  *           and also allows to annotate elements with meta-data.
21  *
22  * \details The first property is important for applications that want a cache
23  *           for the latest n elements of a stream regardless of who is going
24  *           to read them. The point is that the producer should never be
25  *           blocked. The problem with that property is that consumers cannot
26  *           lock the content they are reading so it can be overridden by the
```

```
27 *          producer at any time. This means that the consumer should be
28 *          able to check if the data it read is correct. This results in
29 *          multiple calls for reading.
30 *
31 *          The second property is important when some meta-data need to be
32 *          attached to the samples such as the time at which they have been
33 *          taken in scenarios where the sampling rate isn't fixed. This
34 *          increases the complexity of the ring buffer as both the meta data
35 *          and the data cannot be added at the same time. For this reason, a
36 *          staging area is introduced by this class to keep an atomic commit
37 *          of both the data and its meta-data.
38 *
39 *          **Thread-safety:** There can be only one producer without locking.
40 *          Multiple consumers are ok.
41 *
42 *          ### Examples ###
43 *
44 *          #### Writing data ####
45 *
46 *          Here is a function that would add samples to a #RingBuffer.
47 *
48 *              void writeSamples(RingBuffer<int, bool> &rb, const int *samples,
49 *                                size_t length, bool marker)
50 *              {
51 *                  uint64_t startPos = _head.load();
52 *
53 *                  while (length > 0)
54 *                  {
55 *                      int *rbData;
56 *                      size_t writeLen = requestWrite(length, &r);
57 *
58 *                      memcpy(rbData, samples, writeLen * sizeof(Sample));
59 *
60 *                      // prepare for the next loop
61 *                      samples += writeLen;
62 *                      length -= writeLen;
63 *                  }
64 *                  rb.addMarker(marker, startPos);
65 *                  rb.validateWrite();
66 *              }
67 *
68 *          #### Reading data ####
69 *
70 *          Here is a function that would read samples from a #RingBuffer.
71 *
72 *              bool readSamples(RingBuffer<int, bool> &rb, uint64_t pos,
73 *                               Sample *samples, size_t length)
74 *              {
75 *                  uint64_t startPos = pos;
76 *                  while (length > 0)
77 *                  {
78 *                      int *rbData;
79 *                      size_t wantedLength = length;
80 *
81 *                      size_t rdLen = requestRead(pos, &wantedLength, &rbData);
82 *                      memcpy(samples, wantedLength, rbData);
83 *
84 *                      // prepare for the next loop
85 *                      samples += rdLen;
86 *                      pos += rdLen;
87 *                      length -= rdLen;
88 *                  }
89 *                  return ringBuffer.isPositionValid(startPos);
```

```
 90 *             }
 91 *
 92 */
 93 template <class Sample, class Marker>
 94 class RingBuffer
 95 {
 96 protected:
 97         size_t _ring_length; ///< The length of the ring buffer
 98         std::auto_ptr<Sample> _ring; ///< The ring buffer
 99
100         std::atomic<uint64_t> _newHead; ///< The head of the staging area
101         std::atomic<uint64_t> _head; ///< The head of the public area
102         std::atomic<uint64_t> _tail; ///< The tail of the ring buffer
103
104         /// Structure to associate markers to their positions
105         struct MarkerInternal
106         {
107                 uint64_t pos; ///< Position of the marker
108                 Marker m; ///< The marker
109         };
110         boost::mutex _markersMutex; ///< Mutex to protect the markers' list
111         std::list<MarkerInternal> _markers; ///< The markers' list
112
113         /// Same as #requestRead, but without boundary checks
114         void requestRead_unsafe(uint64_t pos, size_t *length, Sample **samples)
115         {
116                 size_t packetSize = *length;
117
118                 /* check that we won't be wrapping around  */
119                 if ((pos % _ring_length) + packetSize >= _ring_length)
120                         packetSize = (_ring_length - (pos % _ring_length));
121
122                 *samples = (_ring.get() + (pos % _ring_length));
123                 *length = packetSize;
124         }
125
126 public:
127
128         /**
129          * \brief    Construct a new ring buffer
130          *
131          * \param    size The maximum number of elements to be stored in the ring
                buffer.
132          * \return   Nothing.
133          */
134         RingBuffer(size_t size) : _ring_length(size),
135                 _ring(new Sample[size * sizeof(Sample)]), _newHead(0), _head(0),
136                 _tail(0)
137         {
138         }
139
140         /**
141          * \brief    Returns the maximum number of elements that can be stored in
                the ring buffer.
142          * \return   The maximum number of elements that can be stored in the ring
                buffer.
143          */
144         size_t ringLength() const { return _ring_length; }
145
146         /**
147          * \brief    Returns the number of elements currently stored in the ring
                buffer.
148          * \return   The number of elements currently stored in the ring buffer.
```

```
149          */
150         size_t size() const
151         {
152                 return _head.load() - _tail.load();
153         }
154
155         /**
156          * \brief    Checks if a position in the ring buffer is available or not
157          * \return   True if the position is available, false otherwise.
158          */
159         bool isPositionValid(uint64_t pos) const
160         {
161                 return pos >= _tail.load() && pos < _head.load();
162         }
163
164         /**
165          * \brief    Checks if there are n entries available in the ring buffer
                starting from pos.
166          * \return   True if there should be.
167          */
168         bool hasNAvailableFrom(uint64_t pos, size_t n) const
169         {
170                 return (pos + n) < _head.load();
171         }
172
173         /**
174          * \brief    Returns head's position
175          * \return   Returns the position of the head
176          */
177         uint64_t head() const
178         {
179                 return _head.load();
180         }
181
182         /**
183          * \brief    Returns tail's position
184          * \return   Returns the position of the tail
185          */
186         uint64_t tail() const
187         {
188                 return _tail.load();
189         }
190
191         /**
192          * \brief    Empties the ring buffer and markers.
193          *
194          * \return   Nothing.
195          */
196         void clear()
197         {
198                 _markersMutex.lock();
199
200                 _newHead = 0;
201                  _head = 0;
202                 _tail = 0;
203
204                 _markers.clear();
205
206                 _markersMutex.unlock();
207         }
208
209         /**
210          * \brief    Request reading the N last elements of the ring buffer
```

```
211          *
212          * \warning  The data may be changed by the producer while another
213          *           thread is reading data. To ensure that data hasn't been
214          *           overriden, the reader must call #isPositionValid with
215          *           \a pos as a parameter. If the function returns true,
216          *           then the data is safe to use. Otherwise, the data must be
217          *           discarded.
218          *
219          * \param[in]  n           The number of elements you want to read
220          * \param[out] startPos    Stores the position of the first element in
221          *                         the ring buffer.
222          * \param[out] restartPos  Stores the restart position in the case where
223          *                         not all element can be read in a row.
224          * \param[out] samples     Stores the pointer at which data is available.
225          *                         Make sure to only read up to the number
226          *                         returned by this function which can be lower
227          *                         than \a n.
228          *
229          * \return   The samples of elements that can be read. The number can be
230          * lower than what has been requested due to the ring buffer's wrap
231          * around. In this case, one should call #requestRead with \a restartPos
232          * as a starting position.
233          *
234          * \sa #requestRead, #isPositionValid, #addSamples, #requestWrite
235          */
236         size_t requestReadLastN(size_t n, uint64_t *startPos, uint64_t *restartPos
                 , Sample **samples)
237         {
238                 int64_t wantedPos = _head.load() - n;
239                 size_t packetSize = n;
240
241                 /* check there are at least n elements */
242                 if (size() < n) {
243                         packetSize = 0;
244                         if (startPos)
245                                 *startPos = wantedPos;
246                         if (restartPos)
247                                 *restartPos = wantedPos;
248                         if (samples)
249                                 *samples = NULL;
250                 } else {
251                         /* check that we won't be wrapping around */
252                         if ((wantedPos % _ring_length) + n >= _ring_length)
253                                 packetSize = (_ring_length - (wantedPos %
                                         _ring_length));
254
255                         if (startPos)
256                                 *startPos = wantedPos;
257                         if (restartPos)
258                                 *restartPos = wantedPos + packetSize;
259                         if (samples)
260                                 *samples = (_ring.get() + (wantedPos %
                                         _ring_length));
261                 }
262
263                 return packetSize;
264         }
265
266         /**
267          * \brief    Request reading elements from the ring buffer
268          *
269          * \warning  The data may be changed by the producer while another
270          *           thread is reading data. To ensure that data hasn't been
```

```
271          *           overriden, the reader must call #isPositionValid with
272          *           \a pos as a parameter. If the function returns true,
273          *           then the data is safe to use. Otherwise, the data must be
274          *           discarded.
275          *
276          * \param[in]  pos         The starting position in the ring buffer from
277          *                         which data should be read.
278          * \param[in,out] length   Specifies how many elements should be read.
279          *                         It also returns the number of elements that
280          *                         are actually readable.
281          * \param[out] samples     Stores the pointer at which data is available.
282          *                         Make sure to only read up to \a length
283          *                         elements. Warning, \a length can be lower
284          *                         than the requested value.
285          *
286          * \return   True if the position is available and data can be read
287          *           from it. False otherwise.
288          *
289          * \sa #requestReadLastN, #isPositionValid, #addSamples, #requestWrite
290          */
291         bool requestRead(uint64_t pos, size_t *length, Sample **samples)
292         {
293                 if (!isPositionValid(pos)) {
294                         *length = 0;
295                         return false;
296                 }
297
298                 /* check that we aren't trying to read further than head() */
299                 if (pos + *length > _head.load())
300                         *length = _head.load() - pos;
301
302                 requestRead_unsafe(pos, length, samples);
303                 return true;
304         }
305
306         /**
307          * \brief    Request writing new elements to the ring buffer
308          *
309          * \details  This function should be prefered over #addSamples because
310          *           it avoids an uncessary copy.
311          *
312          * \warning  The data added will be staged but not made public until
313          *           #validateWrite is called. This staging area is meant to let
314          *           developers add both data and markers before making
315          *           them public.
316          *
317          * \param[in]  length      The number of elements to be written.
318          * \param[out] samples     Stores the pointer at which data should be
319          *                         written.
320          *                         Make sure to only write up to the number of
321          *                         elements returned by the function.
322          *
323          * \return   The number of elements that can be written at \a samples.
324          *           WARNING: This number may be lower than \a length.
325          *
326          * \sa #addSamples, #validateWrite, #requestRead, #requestReadLastN
327          */
328         size_t requestWrite(size_t length, Sample **samples)
329         {
330                 uint64_t tail = _tail.load();
331                 uint64_t newHead = _newHead.load();
332                 size_t packetSize = length;
333                 requestRead_unsafe(newHead, &packetSize, samples);
```

```
334                 uint64_t wantedNewHead = newHead + packetSize;
335
336                 if ((_newHead.load() - _tail.load()) + packetSize >= _ring_length)
                        {
337                         // reserve the space in the ring buffer
338                         uint64_t wantedTail = wantedNewHead - _ring_length;
339                         if (wantedTail > _head.load())
340                                 _head.store(wantedTail);
341                         _tail.store(wantedTail);
342
343                         /* delete the markers that have been overridden */
344                         typename std::list<MarkerInternal>::iterator it =_markers.
                                begin();
345                         while (it != _markers.end() && (*it).pos >= tail && (*it).
                                pos < wantedTail)
346                                 it = _markers.erase(it);
347                 }
348
349                 _newHead.store(wantedNewHead);
350
351                 return packetSize;
352         }
353
354         /**
355          * \brief    Flush the staging area and make the newly-added data public
356          *
357          * \details  When data is added, it is staged and not made public until
358          *           #validateWrite is called. This staging area is meant to let
359          *           developers add both data and markers before making
360          *           them public.
361          *
362          * \return   Nothing.
363          *
364          * \sa #requestWrite, #addSamples
365          */
366         void validateWrite()
367         {
368                 _head.store(_newHead.load());
369         }
370
371         /**
372          * \brief    Write new elements to the ring buffer
373          *
374          * \warning  This function should be avoided as possible. Please prefer
375          *           avoiding a copy by using #requestWrite.
376          *
377          * \warning  The data added will be staged but not made public until
378          *           #validateWrite is called. This staging area is meant to let
379          *           developers add both data and markers before making
380          *           them public.
381          *
382          * \param[in] samples      The samples to be written to the ring buffer
383          * \param[in] length       The number of elements from samples to be
384          *                         added to the ring buffer.
385          *
386          * \return   The address of the first element added to the ring buffer.
387          *
388          * \sa #requestWrite, #validateWrite, #requestRead, #requestReadLastN
389          */
390         uint64_t addSamples(const Sample *samples, size_t length)
391         {
392                 uint64_t begin = _head.load();
393
```

```
394                 while (length > 0)
395                 {
396                         Sample *r;
397                         size_t packetSize = requestWrite(length, &r);
398
399                         memcpy(r, samples, packetSize * sizeof(Sample));
400
401                         // prepare for the next loop
402                         samples += packetSize;
403                         length -= packetSize;
404                 }
405
406                 return begin;
407         }
408
409         /**
410          * \brief    Add a marker at a specific position
411          *
412          * \param[in] marker      The marker to be added
413          * \param[in] pos         The position at which the marker \a marker
414          *                        should be added. If a marker already exists at
415          *                        this position. It will be replaced by
416          *                        the new marker \a marker. The marker should be
417          *                        stored in added in the staging area otherwise,
418          *                        the function will return false.
419          *
420          * \return   True if the position was valid and the marker has been
421          *           added. False if the position was invalid (position not in
422          *           the staging area).
423          *
424          * \sa #getMarker, #findMarker
425          */
426         bool addMarker(Marker marker, uint64_t pos)
427         {
428                 if (!(pos >= _head.load() && pos < _newHead.load()))
429                         return false;
430
431                 _markersMutex.lock();
432
433                 MarkerInternal m = {pos, marker};
434
435                 typename std::list<MarkerInternal>::reverse_iterator rit;
436                 for (rit =_markers.rbegin(); rit!=_markers.rend(); ++rit) {
437                         if ((*rit).pos == pos) {
438                                 (*rit).m = marker;
439                         } else if ((*rit).pos < pos) {
440                                 _markers.insert(rit.base(), m);
441                                 goto exit;
442                         }
443                 }
444
445                 _markers.push_front(m);
446
447 exit:
448                 _markersMutex.unlock();
449                 return true;
450         }
451
452         /**
453          * \brief    Get a marker at a known position
454          *
455          * \param[in] markerPos   The position at which the marker should be
456          *                        retrieved.
```

```
457          * \param[out] marker     Stores the retrieved marker.
458          *
459          * \return   True if the marker has been retrieved, false otherwise.
460          *
461          * \sa #findMarker, #addMarker
462          */
463         bool getMarker(uint64_t markerPos, Marker *marker)
464         {
465                 typename std::list<MarkerInternal>::reverse_iterator rit;
466                 bool ret = false;
467
468                 _markersMutex.lock();
469                 for (rit =_markers.rbegin(); rit!=_markers.rend(); ++rit) {
470                         if ((*rit).pos == markerPos) {
471                                 *marker = (*rit).m;
472                                 ret = true;
473                                 break;
474                         }
475                 }
476                 _markersMutex.unlock();
477
478                 return ret;
479         }
480
481         /**
482          * \brief    Find the marker right before a position
483          *
484          * \param[in] pos         The position from which the marker should be
485          *                        searched for.
486          * \param[out] marker     Stores the retrieved marker.
487          * \param[out] markerPos  Stores the marker's position
488          *
489          * \return   True if a marker has been found, false otherwise.
490          *
491          * \sa #getMarker, #addMarker
492          */
493         bool findMarker(uint64_t pos, Marker *marker, uint64_t *markerPos)
494         {
495                 bool ret = false;
496
497                 if (!isPositionValid(pos))
498                         return false;
499
500                 _markersMutex.lock();
501                 typename std::list<MarkerInternal>::reverse_iterator rit =
                        _markers.rbegin();
502
503                 while (rit != _markers.rend() && (*rit).pos > pos)
504                         ++rit;
505
506                 if (rit != _markers.rend()) {
507                         *marker = (*rit).m;
508                         *markerPos = (*rit).pos;
509                         ret = true;
510                 }
511
512                 _markersMutex.unlock();
513
514                 return ret;
515         }
516 };
517
518 #endif // RINGBUFFER_H
```

Bibliography

[1] jjgarcia.tsc, "Carrier sense multiple access with collision avoidance," 2011, page Version ID: 628813304. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Carrier_sense_multiple_access_with_collision_avoidance&oldid=628813304 xi, 10

[2] "Network topology," Dec. 2014, page Version ID: 636339599. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Network_topology&oldid=636339599 xi, 14

[3] Z. Lin, H. Liu, X. Chu, and Y.-W. Leung, "Enhanced jump-stay rendezvous algorithm for cognitive radio networks," *IEEE Communications Letters*, vol. 17, no. 9, pp. 1742–1745, Sep. 2013. xii, 47, 67, 68

[4] R. Doost-Mohammady, P. Paweczak, G. Janssen, and H. Segers, "Physical layer bootstrapping protocol for cognitive radio networks," in *2010 7th IEEE Consumer Communications and Networking Conference (CCNC)*, Jan. 2010, pp. 1–5. xii, 48, 67, 68

[5] Juha Pennanen, John Hoversten, and Sasa Radovanonic, "Envelope tracking for cellular RF power amplifiers | PowerGuru - power electronics information portal," Mar. 2014. [Online]. Available: https://www.powerguru.org/envelope-tracking-for-cellular-rf-power-amplifiers/ xii, 78

[6] N. Kim, T. Austin, D. Baauw, T. Mudge, K. Flautner, J. Hu, M. Irwin, M. Kandemir, and V. Narayanan, "Leakage current: Moore's law meets static power," *Computer*, vol. 36, no. 12, pp. 68–75, 2003. xii, 17, 81, 82

[7] H. Cha, R. J. Hasslen III, J. A. Robinson, S. J. Treichler, and A. U. Diril, "Power estimation based on block activity," U.S. Patent 8 060 765, Nov., 2011. [Online]. Available: http://www.freepatentsonline.com/8060765.html xii, 91, 92

[8] E. Rotem, A. Naveh, D. Rajwan, A. Ananthakrishnan, and E. Weissmann, "Power-management architecture of the intel microarchitecture code-named sandy bridge," *IEEE Micro*, vol. 32, no. 2, pp. 20–27, Mar. 2012. xii, xiii, 89, 92, 93, 95, 96

[9] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, "Energy consumption in mobile phones: A measurement study and implications for network applications," in *Proceedings of the 9th ACM SIGCOMM Conference on*

Internet Measurement Conference, ser. IMC '09. New York, NY, USA: ACM, 2009, pp. 280–293. [Online]. Available: http://doi.acm.org/10.1145/1644893.1644927 xiii, 111, 112, 113, 114

[10] ARM, "big.LITTLE technology - ARM," Tech. Rep., 2011. [Online]. Available: http://arm.com/products/processors/technologies/biglittleprocessing.php xiii, 115

[11] F. Broquedis, J. Clet-Ortega, S. Moreaud, N. Furmento, B. Goglin, G. Mercier, S. Thibault, and R. Namyst, "hwloc: A generic framework for managing hardware affinities in HPC applications." IEEE, Feb. 2010, pp. 180–186. [Online]. Available: http://hal.inria.fr/inria-00429889 xiii, 116

[12] A. M. Tamasi, P. B. Johnson, F. R. Diard, and B. M. Kelleher, "Optimizing power and performance for multi-processor graphics processing," U.S. Patent 7 721 118, May, 2010. [Online]. Available: http://www.freepatentsonline.com/7721118.html xiii, 117, 118

[13] SenseFly, "senseFly. swinglet CAM," 2010. [Online]. Available: https://www.sensefly.com/drones/swinglet-cam.html xiv, 142

[14] Google, "Loon for all - project loon - google," 2013. [Online]. Available: http://www.google.com/loon/ xiv, 143

[15] Amazon, "Amazon prime air," 2014. [Online]. Available: http://www.amazon.com/b?node=8037720011 xiv, 143

[16] Quaduro systems, "QuadPad v12 IP54 semi rugged tablet PC," 2011. [Online]. Available: http://www.quaduro.com/en/gb/products/src.php?id=5 xiv, 144

[17] G. Nychis, T. Hottelier, Z. Yang, S. Seshan, and P. Steenkiste, "Enabling MAC protocol implementations on software-defined radios," in *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, ser. NSDI'09. Berkeley, CA, USA: USENIX Association, 2009, pp. 91–105. xvii, 48, 56

[18] D. HISKEY, "This day in history: Martin cooper publicly demonstrates the world's first handheld mobile phone," Apr. 2012. [Online]. Available: http://www.todayifoundout.com/index.php/2012/04/this-day-in-history-martin-cooper-publicly-demonstrates-the-worlds-first-handheld-mobile-phone/ 1

[19] C. Pirlogea and C. Cicea, "Econometric perspective of the energy consumption and economic growth relation in european union," *Renewable and Sustainable Energy Reviews*, vol. 16, no. 8, pp. 5718–5726, Oct. 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1364032112003905 7

[20] B. Plumer, "Can we sever the link between energy and economic growth?" *The Washington Post*, Jan. 2014. [Online]. Available: http://www.washingtonpost.com/blogs/wonkblog/wp/2014/01/17/can-we-sever-the-link-between-energy-and-growth/ 7

[21] EPA, "EPA report to congress on server and data center energy efficiency," Tech. Rep., 2007. [Online]. Available: http://hightech.lbl.gov/documents/data_centers/epa-datacenters.pdf 7, 76

[22] "TIC et développement durable," Mar. 2009, les technologies de l'information et de la communication (TIC) peuvent permettre d'économiser de 1 à 4 fois leurs propres émissions de gaz à effet de serre mais, à l'inverse, ils constituent désormais le premier poste de consommation d'électricité des ménages. Le rapport évalue l'impact des TIC sur le développement durable et propose des politiques pour le futur. [Online]. Available: http://www.ladocumentationfrancaise.fr/rapports-publics/094000118/ 7

[23] F. Krief, *Green Networking*. John Wiley & Sons, Dec. 2012. 7, 8, 11, 18

[24] GeSI, "GeSI SMARTer2020: The role of ICT in driving a sustainable future," Tech. Rep., Dec. 2012. [Online]. Available: http://gesi.org/SMARTer2020 8

[25] A. Bianzino, C. Chaudet, D. Rossi, and J. Rougier, "A survey of green networking research," *IEEE Communications Surveys Tutorials*, vol. 14, no. 1, pp. 3–20, 2012. 8

[26] K.-L. Du and M. N. S. Swamy, *Wireless Communication Systems: From RF Subsystems to 4G Enabling Technologies*. Cambridge University Press, Apr. 2010. 8

[27] J.-l. Yang, W.-j. Yang, and G.-q. Xu, "Blind estimation of carrier frequency and symbol rate based on cyclic spectrum density," *Procedia Engineering*, vol. 29, pp. 514–519, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877705811065908 9

[28] A. Furuskar, S. Mazur, F. Muller, and H. Olofsson, "EDGE: enhanced data rates for GSM and TDMA/136 evolution," *IEEE Personal Communications*, vol. 6, no. 3, pp. 56–66, Jun. 1999. 9

[29] L. E. Frenzel, *Principles of electronic communication systems*. McGraw-Hill, 2008. 9

[30] I. Akbar, "Power from the void?: How steve perlman's "revolutionary" wireless technology works and why it is a bigger deal than anyone realizes." *IEEE Consumer Electronics Magazine*, vol. 3, no. 3, pp. 36–43, Jul. 2014. 9

[31] B. Wang and K. Liu, "Advances in cognitive radio networks: A survey," *IEEE Journal of Selected Topics in Signal Processing*, vol. 5, no. 1, pp. 5–23, Feb. 2011. 10

[32] K. K. Khedo, R. Perseedoss, and A. Mungur, "A wireless sensor network air pollution monitoring system," *CoRR*, vol. abs/1005.1737, 2010. 11

[33] G. Werner-allen, J. Johnson, M. Ruiz, J. Lees, and M. Welsh, "Monitoring volcanic eruptions with a wireless sensor network," in *in Proceedings of the Second European Workshop on Wireless Sensor Networks (EWSN'05*, 2005. 11

[34] M. Hefeeda and M. Bagheri, *Forest Fire Modeling and Early Detection using Wireless Sensor Networks*, 2009. 11

[35] R. Morello, C. De Capua, and A. Meduri, "Remote monitoring of building structural integrity by a smart wireless sensor network," in *Instrumentation and Measurement Technology Conference (I2MTC), 2010 IEEE*, May 2010, pp. 1150 –1154. 11

[36] M. Peres, R. Perier, and F. Krief, "Overcoming the deficiencies of collaborative detection of spatially-correlated events in WSN," in *Advanced Infocomm Technology*, ser. Lecture Notes in Computer Science, V. Guyot, Ed. Springer Berlin Heidelberg, Jul. 2012, no. 7593, pp. 243–257. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-642-38227-7_27 11

[37] Wave2m community, "Wave2m - the platform in-depth," 2008. [Online]. Available: http://www.wave2m.org/the-specification/the-platform-in-depth?view=item 11, 12, 188

[38] Stop Smart Meters Australia, "Privacy issues," 2014. [Online]. Available: http://stopsmartmeters.com.au/privacy-issues/ 11

[39] Q. Wang, M. Hempstead, and W. Yang, "A realistic power consumption model for wireless sensor network devices," in *2006 3rd Annual IEEE Communications Society on Sensor and Ad Hoc Communications and Networks, 2006. SECON '06*, vol. 1, Sep. 2006, pp. 286–295. 12, 22

[40] Marcin Brzozowski and Peter Langendoerfer, "Multi-channel support for preamble sampling MAC protocols in sensor networks," in *The 22nd International Conference on Software, Telecommunications and Computer Networks (SoftCOM 2014)*, Split, Croatia, Sep. 2014. 12

[41] J. Degesys, I. Rose, A. Patel, and R. Nagpal, "DESYNC: Self-organizing desynchronization and TDMA on wireless sensor networks," in *6th International Symposium on Information Processing in Sensor Networks, 2007. IPSN 2007*, Apr. 2007, pp. 11–20. 13

[42] G. Allard, V. Genc, and J. Yelloz, "Fully distributed clock synchronization in wide-range TDMA ad-hoc networks," in *MILITARY COMMUNICATIONS CONFERENCE, 2011 - MILCOM 2011*, Nov. 2011, pp. 920–925. 13

[43] D. Doronin and D. Fakhriev, "Distributed clock synchronization algorithm for wide-range TDMA ad hoc networks," in *Wireless Access Flexibility*, ser. Lecture Notes in Computer Science, G. Bianchi, A. Lyakhov, and E. Khorov, Eds. Springer Berlin Heidelberg, Jan. 2013, no. 8072, pp. 112–124. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-642-39805-6_10 13

[44] T. Herman and S. Tixeuil, "A distributed TDMA slot assignment algorithm for wireless sensor networks," in *Algorithmic Aspects of Wireless Sensor Networks*, ser. Lecture Notes in Computer Science, S. E. Nikoletseas and J. D. P. Rolim, Eds. Springer Berlin Heidelberg, Jan. 2004, no. 3121, pp. 45–58. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-540-27820-7_6 13

[45] T. Clausen, P. Jacquet, C. Adjih, A. Laouiti, P. Minet, P. Muhlethaler, A. Qayyum, L. Viennot, and others, "Optimized link state routing protocol (OLSR)," 2003. 14

[46] A. Neumann, C. Aichele, M. Lindner, and S. Wunderlich, "Better approach to mobile ad-hoc networking (BATMAN)," *IETF draft, October*, 2008. 14

[47] I. D. Chakeres and E. M. Belding-Royer, "AODV routing protocol implementation design," in *Proceedings of the 24th International Conference on Distributed Computing Systems Workshops - W7: EC (ICDCSW'04) - Volume 7*, ser. ICDCSW '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 698–703. [Online]. Available: http://dl.acm.org/citation.cfm?id=977399.977938 14

[48] Y. Yao and J. Gehrke, "The cougar approach to in-network query processing in sensor networks," *SIGMOD Rec.*, vol. 31, no. 3, pp. 9–18, Sep. 2002. [Online]. Available: http://doi.acm.org/10.1145/601858.601861 14

[49] N. Sadagopan, B. Krishnamachari, and A. Helmy, "Active query forwarding in sensor networks," *Journal of Ad Hoc Networks*, vol. 3, pp. 91–113, 2005. 14

[50] C. Chaudet, N. Costagliola, I. Demeure, S. Ktari, and S. Tardieu, "Building an efficient overlay for publish/subscribe in wireless sensor networks," in *2012 9th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, Jun. 2012, pp. 362–370. 15, 26

[51] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-efficient communication protocol for wireless microsensor networks," in *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences, 2000*, Jan. 2000, pp. 10 pp. vol.2–. 15, 24

[52] Tim Winter, "RPL: IPv6 routing protocol for low-power and lossy networks," Mar. 2012. [Online]. Available: https://tools.ietf.org/html/rfc6550 15, 23, 36

[53] L. Zhiyuan, "Geographic routing protocol and simulation," in *Second International Workshop on Computer Science and Engineering, 2009. WCSE '09*, vol. 2, Oct. 2009, pp. 404–407. 16

[54] G. Schaefer, F. Ingelrest, and M. Vetterli, "Potentials of opportunistic routing in energy-constrained wireless sensor networks," Cork, Ireland, Feb. 2009. [Online]. Available: http://rr.epfl.ch/19/ 16

[55] G. S. Aujla and S. S. Kang, "Comprehensive evaluation of AODV, DSR, GRP, OLSR and TORA routing protocols with varying number of nodes and traffic applications over MANETs," Apr. 2013. 16

[56] G. Mulligan, "The 6lowpan architecture," in *Proceedings of the 4th Workshop on Embedded Networked Sensors*, ser. EmNets '07. New York, NY, USA: ACM, 2007, pp. 78–82. [Online]. Available: http://doi.acm.org/10.1145/1278972.1278992 16

[57] B. Frank, Z. Shelby, K. Hartke, and C. Bormann, "Constrained application protocol (CoAP)," Oct. 2010. [Online]. Available: https://tools.ietf.org/html/draft-ietf-core-coap-03 16, 33, 36

[58] N. Schlumpf, M. Declercq, and C. Dehollaini, "A fast modulator for dynamic supply linear RF power amplifier," *IEEE Journal of Solid-State Circuits*, vol. 39, no. 7, pp. 1015–1025, Jul. 2004. 17, 78

[59] D. Talbot, "Beyond CES hype lies a big battery problem. here's how the wireless industry is racing to fix it." Jan. 2014. [Online]. Available: http://www.technologyreview.com/news/523286/the-hottest-technology-not-on-display-at-ces-smart-radio-chips/ 17

[60] S. Chung, P. A. Godoy, T. W. Barton, E. W. Huang, D. J. Perreault, and J. L. Dawson, "Asymmetric multilevel outphasing architecture for multi-standard transmitters," in *Radio Frequency Integrated Circuits Symposium, 2009. RFIC 2009. IEEE*. IEEE, 2009, pp. 237–240. 17, 79

[61] R. R. Schaller, "Moore's law: Past, present, and future," *IEEE Spectr.*, vol. 34, no. 6, pp. 52–59, Jun. 1997. [Online]. Available: http://dx.doi.org/10.1109/6.591665 17

[62] J. Koomey, S. Berard, M. Sanchez, and H. Wong, "Implications of historical trends in the electrical efficiency of computing," *Annals of the History of Computing, IEEE*, vol. 33, no. 3, pp. 46 –54, Mar. 2011. 17, 22, 25

[63] D. R. Suleiman, M. A. Ibrahim, and I. I. Hamarash, *Dynamic Voltage Frequency Scaling (dvfs) for Microprocessors Power and Energy Reduction*, 2005. 17

[64] V. K. Srikantam and A. R. Clark II, "Method and apparatus for clock gating clock trees to reduce power dissipation," U.S. Patent 6 822 481, Nov., 2004. [Online]. Available: http://www.freepatentsonline.com/6822481.html 18

[65] J. Kephart and D. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003. 18, 77, 105

[66] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks*, vol. 38, pp. 393–422, 2002. 22

[67] G. Barrenetxea, F. Ingelrest, G. Schaefer, and M. Vetterli, "Wireless sensor networks for environmental monitoring: The SensorScope experience," in *2008 IEEE International Zurich Seminar on Communications*, Mar. 2008, pp. 98–101. 23

[68] B. Krishnamachari, D. Estrin, and S. B. Wicker, "The impact of data aggregation in wireless sensor networks," in *Proceedings of the 22nd International Conference on Distributed Computing Systems*, ser. ICDCSW '02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 575–578. [Online]. Available: http://dl.acm.org/citation.cfm?id=646854.708078 24

[69] H. Letian and L. Guangjun, "A reconfigurable system for digital signal processing," in *Signal Processing, 2008. ICSP 2008. 9th International Conference on*, Oct. 2008, pp. 439 –442. 25

[70] S. Croce, F. Marcelloni, and M. Vecchio, "Reducing power consumption in wireless sensor networks using a novel approach to data aggregation," *The Computer Journal*, vol. 51, no. 2, pp. 227–239, Mar. 2008. [Online]. Available: http://comjnl.oxfordjournals.org/content/51/2/227 25

[71] B. Lazzerini, F. Marcelloni, M. Vecchio, S. Croce, and E. Monaldi, "A fuzzy approach to data aggregation to reduce power consumption in wireless sensor networks," in *Fuzzy Information Processing Society, 2006. NAFIPS 2006. Annual meeting of the North American*, Jun. 2006, pp. 436 –441. 25

[72] A. Nasipuri, *Collaborative Detection of Spatially Correlated Signals in Sensor Networks*, Dallas, Texas, Nov. 2005, vol. Proceedings of the 2005 International Conference on Telecommunication Systems Modeling and Analysis. 25

[73] A. V. U. Phani Kumar, A. M. Reddy V, and D. Janakiram, "Distributed collaboration for event detection in wireless sensor networks," in *Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing*, ser. MPAC '05. New York, NY, USA: ACM, 2005, pp. 1–8. [Online]. Available: http://doi.acm.org/10.1145/1101480.1101491 25

[74] Y. He, M. Li, and Y. Liu, "Collaborative query processing among heterogeneous sensor networks," in *Proceedings of the 1st ACM international workshop on Heterogeneous sensor and actor networks*, ser. HeterSanet '08. New York, NY, USA: ACM, 2008, pp. 25–30. [Online]. Available: http://doi.acm.org/10.1145/1374699.1374705 25

[75] D. Andersson, M. Fong, and A. Valdes, *Heterogeneous Sensor Correlation: A Case Study of Live Traffic Analysis*, 2002. 25

[76] T. Clouqueur, P. Ramanathan, K. K. Saluja, and K.-c. Wang, "Value-fusion versus decision-fusion for fault-tolerance in collaborative target detection in sensor networks," *in proceedings of fourth international conference on information fusion*, p. pp., 2001. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.116.6587 25, 27

[77] E. Ould-Ahmed-Vall, B. Heck Ferri, and G. Riley, "Distributed fault-tolerance for event detection using heterogeneous wireless sensor networks," *Mobile Computing, IEEE Transactions on*, vol. PP, no. 99, p. 1, 2011. 25

[78] J. Harrison, "A short survey of automated reasoning," in *Proceedings of the 2Nd International Conference on Algebraic Biology*, ser. AB'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 334–349. [Online]. Available: http://dl.acm.org/citation.cfm?id=1769026.1769050 26

[79] F. Portoraro, "Automated reasoning," in *The Stanford Encyclopedia of Philosophy*, summer 2014 ed., E. N. Zalta, Ed., 2010. [Online]. Available: http://plato.stanford.edu/archives/sum2014/entries/reasoning-automated/ 27

[80] A. A. Abbasi and M. Younis, "A survey on clustering algorithms for wireless sensor networks," *Computer Communications*, vol. 30, no. 14–15, pp. 2826–2841, Oct. 2007. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140366407002162 31

[81] LaBRI, "Diaforus," 2010, [Online; accessed 25-May-2012]. [Online]. Available: https://diaforus.labri.fr/doku.php 35

[82] M. Simon, J. Omura, R. Scholtz, and B. Levitt, *Spread Spectrum Communications Handbook, Electronic Edition*. The McGraw-Hill Companies, Inc., 1994. 44

[83] Nuand, "bladeRF x40 | nuand," 2013. [Online]. Available: http://www.nuand.com/blog/product/bladerf-x40/ 47, 55, 62

[84] C.-S. Hsu, Y.-S. Chen, and C.-E. He, "An efficient dynamic adjusting MAC protocol for multichannel cognitive wireless networks," in *2010 IEEE International Conference on Wireless Communications, Networking and Information Security (WCNIS)*, Jun. 2010, pp. 556–560. 47

[85] C. Cordeiro and K. Challapali, "C-MAC: A cognitive MAC protocol for multi-channel wireless networks," in *2nd IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks, 2007. DySPAN 2007*, Apr. 2007, pp. 147–157. 47

[86] I. F. Akyildiz, W.-Y. Lee, M. C. Vuran, and S. Mohanty, "NeXt generation/dynamic spectrum access/cognitive radio wireless networks: A survey," *COMPUTER NETWORKS JOURNAL (ELSEVIER*, vol. 50, pp. 2127–2159, 2006. 47

[87] Ettus Research, "USRP1," 2005. [Online]. Available: https://www.ettus.com/product/details/USRPPKG 48, 56

[88] ——, "USRP hardware driver™," 2006. [Online]. Available: http://code.ettus.com/redmine/ettus/projects/uhd/wiki 48

[89] Martin Peres, "mupuf/hachoir," 2013. [Online]. Available: https://github.com/mupuf/hachoir 52, 62, 168

[90] A. P.S and M. Jayasheela, "Cyclostationary feature detection in cognitive radio using different modulation schemes," *International Journal of Computer Applications*, vol. 47, no. 21, pp. 12–16, Jun. 2012. 53

[91] T. Yucek and H. Arslan, "A survey of spectrum sensing algorithms for cognitive radio applications," *IEEE Communications Surveys Tutorials*, vol. 11, no. 1, pp. 116–130, 2009. 53

[92] Abhilasha Singh and Anita Sharma, "A survey of various defense techniques to detect primary user emulation attacks - inpressco," *International Journal of Current Engineering and Technology International Journal of Current Engineering and Technology Vol4, No.2*, Apr. 2014. [Online]. Available: http://inpressco.com/a-survey-of-various-defense-techniques-to-detect-primary-user-emulation-attacks/ 53

[93] Ettus Research, "USRP x300," 2014. [Online]. Available: https://www.ettus.com/product/details/X300-KIT 56, 57

[94] Nordic Semiconductor, "nRF24l01 - ultra low power 2.4ghz RF transceiver IC," Jul. 2007. [Online]. Available: http://www.nordicsemi.com/eng/Products/2.4GHz-RF/nRF24L01 66

[95] "Aalto talk with linus torvalds [full-length]," Jun. 2012. [Online]. Available: https://www.youtube.com/watch?v=MShbP3OpASA&t=2894s 76

[96] Nouveau Community, "Nouveau, the community-driven open source NVIDIA driver," 2006. [Online]. Available: http://nouveau.freedesktop.org/wiki/ 76

[97] Nouveau, "Envytools - tools for people envious of nvidia's blob driver." 2009. [Online]. Available: https://github.com/envytools/envytools 77, 86, 88, 169

[98] S. Kato, K. Lakshmanan, R. Rajkumar, and Y. Ishikawa, "TimeGraph: GPU scheduling for real-time multi-tasking environments," in *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference*, ser. USENIXATC'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 2–2. [Online]. Available: http://dl.acm.org/citation.cfm?id=2002181.2002183 77

[99] S. Kato, M. McThrow, C. Maltzahn, and S. Brandt, "Gdev: First-class GPU resource management in the operating system," in *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, ser. USENIX ATC'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 37–37. [Online]. Available: http://dl.acm.org/citation.cfm?id=2342821.2342858 77

[100] Y. Fujii, T. Azumi, N. Nishio, S. Kato, and M. Edahiro, "Data transfer matters for GPU computing," in *2013 International Conference on Parallel and Distributed Systems (ICPADS)*, Dec. 2013, pp. 275–282. 77

[101] S. Kato, J. Aumiller, and S. Brandt, "Zero-copy i/o processing for low-latency GPU computing," in *Proceedings of the ACM/IEEE 4th International Conference on Cyber-Physical Systems*, ser. ICCPS '13. New York, NY, USA: ACM, 2013, pp. 170–178. [Online]. Available: http://doi.acm.org/10.1145/2502524.2502548 77

[102] K. Wang, X. Ding, R. Lee, S. Kato, and X. Zhang, "GDM: Device memory management for gpgpu computing," in *The 2014 ACM International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '14. New York, NY, USA: ACM, 2014, pp. 533–545. [Online]. Available: http://doi.acm.org/10.1145/2591971.2592002 77

[103] Y. Suzuki, S. Kato, H. Yamada, and K. Kono, "GPUvm: Why not virtualizing GPUs at the hypervisor?" in *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. Philadelphia, PA: USENIX Association, Jun. 2014, pp. 109–120. [Online]. Available: https://www.usenix.org/conference/atc14/technical-sessions/presentation/suzuki 77

[104] Y. Abe, H. Sasaki, M. Peres, K. Inoue, K. Murakami, and S. Kato, "Power and performance analysis of GPU-accelerated systems," in *Proceedings of the 2012 USENIX conference on Power-Aware Computing and Systems*, ser. HotPower'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 10–10. [Online]. Available: http://dl.acm.org/citation.cfm?id=2387869.2387879 77, 86, 100

[105] ——, "Power and performance characterization and modeling of GPU-accelerated systems," in *IEEE International Parallel & Distributed Processing Symposium 2014*, PHOENIX (Arizona), USA, 2014. [Online]. Available: http://phd.mupuf.org/publication/2014/05/19/power-and-performance-characterization-and-modeling-of-gpu-accelerated-systems/ 77, 86, 101, 103, 120

[106] Martin Peres, "Reverse engineering power management on NVIDIA GPUs - anatomy of an autonomic-ready system," in *ECRTS, Operating Systems Platforms for Embedded Real-Time applications 2013*, Jul. 2013. [Online]. Available: http://phd.mupuf.org/publication/2013/07/09/reverse-engineering-power-management-on-nvidia-gpus/ 77, 94

[107] ——, "Reverse engineering power management on NVIDIA GPUs - a detailed overview," in *X.Org Developer's Conference 2013*, Sep. 2013. [Online]. Available: http://phd.mupuf.org/publication/2013/09/25/reverse-engineering-power-management-on-nvidia-gpus/ 77

[108] C. Wittenbrink, E. Kilgariff, and A. Prabhu, "Fermi GF100 GPU architecture," *IEEE Micro*, vol. 31, no. 2, pp. 50–59, 2011. 79

[109] E. Le Sueur and G. Heiser, "Dynamic voltage and frequency scaling: the laws of diminishing returns," in *Proceedings of the 2010 international conference on Power aware computing and systems*, ser. HotPower'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 1–8. [Online]. Available: http://dl.acm.org/citation.cfm?id=1924920.1924921 80, 101, 102, 114, 120

[110] G. Klimeck, "ECE 606 lecture 25: Modern MOSFETs," 2012. [Online]. Available: https://nanohub.org/resources/16055/watch?resid=16068&tmpl=component 81

[111] T. Vogelsang, "Understanding the energy consumption of dynamic random access memories," in *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec. 2010, pp. 363–374. 82

[112] M.-T. Chang, P. Rosenfeld, S.-L. Lu, and B. Jacob, "Technology comparison for large last-level caches (l3cs): Low-leakage SRAM, low write-energy STT-RAM, and refresh-optimized eDRAM," in *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA2013)*, Feb. 2013, pp. 143–154. 83

[113] R. Evans and P. Franzon, "Energy consumption modeling and optimization for SRAM's," *IEEE Journal of Solid-State Circuits*, vol. 30, no. 5, pp. 571–579, May 1995. 83

[114] Micron Technology, Inc., "Wear leveling in micron NAND flash memory," 2011. [Online]. Available: http://www.micron.com/-/media/documents/products/technicalnote/nandflash/tn2961_wear_leveling_in_nand.pdf 83

[115] Texas Instruments, Inc., "MSP430x2xx family user's guide," 2013. [Online]. Available: http://www.ti.com/lit/ug/slau144j/slau144j.pdf 84

[116] NVIDIA, "NVIDIA SDR (software defined radio) technology," 2013. [Online]. Available: http://www.nvidia.com/docs/IO//116757/NVIDIA_i500_whitepaper_FINALv3.pdf 85, 153

[117] USB Implementers Forum, Inc., "A technical introduction to USB 2.0," 2001. [Online]. Available: http://www.usb.org/developers/whitepapers/usb_20g.pdf 85

[118] PCI-SIG, "PCI-SIG - specifications," 1992. [Online]. Available: https://www.pcisig.com/specifications/ 85

[119] ——, "PCI express," 2013. [Online]. Available: http://www.pcisig.com/specifications/pciexpress/resources/ 85

[120] Intel/Microsoft/Toshiba, "ACPI - advanced configuration & power interface," 1996. [Online]. Available: http://www.acpi.info/ 85

[121] Thomas Pettazzoni, "Device tree for dummies," 2013. [Online]. Available: http://events.linuxfoundation.org/sites/events/files/slides/petazzoni-device-tree-dummies.pdf 86

[122] NVIDIA, "PerfKit," 2006. [Online]. Available: https://developer.nvidia.com/nvidia-perfkit 86

[123] ——, "CUDA profiling tools interface," 2007. [Online]. Available: https://developer.nvidia.com/cuda-profiling-tools-interface 86

[124] H. Nagasaka, N. Maruyama, A. Nukada, T. Endo, and S. Matsuoka, "Statistical power modeling of GPU kernels using performance counters," in *Green Computing Conference, 2010 International*, 2010, pp. 115–122. 86, 92, 101, 120

[125] J. M. Alben, R. J. Hasslen III, and S. J. Treichler, "Clock throttling based on activity-level signals," U.S. Patent 7 958 483, Jun., 2011. [Online]. Available: http://www.freepatentsonline.com/7958483.html 88

[126] Taylor Kidd, "Power management states: P-states, c-states, and package c-states," Apr. 2014. [Online]. Available: https://software.intel.com/en-us/articles/power-management-states-p-states-c-states-and-package-c-states 88, 89, 90

[127] Arjan van de Ven, "Some basics on CPU p states on intel processors there seems to be a lot of...," Jun. 2013. [Online]. Available: https://plus.google.com/+ArjanvandeVen/posts/dLn9T4ehywL 89

[128] Nouveau Community, "Codenames of NVIDIA hardware," 2008. [Online]. Available: http://nouveau.freedesktop.org/wiki/CodeNames/ 89

[129] Intel Corporation, "Intel® 64 and IA-32 architectures software developer manuals," Feb. 2014. [Online]. Available: http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html 94

[130] M. Hähnel, B. Döbel, M. Völp, and H. Härtig, "Measuring energy consumption for short code paths using RAPL," *SIGMETRICS Perform. Eval. Rev.*, vol. 40, no. 3, pp. 13–17, Jan. 2012. [Online]. Available: http://doi.acm.org/10.1145/2425248.2425252 94, 120

[131] D. Kanter, "Intel's sandy bridge microarchitecture," Sep. 2010. [Online]. Available: http://www.realworldtech.com/sandy-bridge/9/ 95

[132] NVIDIA, "NVIDIA GeForce GTX 680," 2012. [Online]. Available: http://www.geforce.com/Active/en_US/en_US/pdf/GeForce-GTX-680-Whitepaper-FINAL.pdf 96

[133] M. Burtscher, I. Zecena, and Z. Zong, "Measuring GPU power with the k20 built-in sensor," in *Proceedings of Workshop on General Purpose Processing Using GPUs*, ser. GPGPU-7. New York, NY, USA: ACM, 2014, pp. 28:28–28:36. [Online]. Available: http://doi.acm.org/10.1145/2576779.2576783 97

[134] Texas Instruments, "INA3221 - triple channel shunt and bus voltage monitor," May 2012. [Online]. Available: http://www.ti.com/product/ina3221 97

[135] NVIDIA, "NVIDIA x config options," May 2014. [Online]. Available: http://us.download.nvidia.com/solaris/337.25/README/xconfigoptions.html 100

[136] Unigine Corp., "Heaven benchmark | unigine: real-time 3d engine (game, simulation, visualization and VR)," Dec. 2013. [Online]. Available: https://unigine.com/products/heaven/ 103

[137] Digital Content Protection, LLC, "High-bandwidth digital content protection (HDCP)," 2001. [Online]. Available: http://www.digital-cp.com/ 106

[138] Alex Bradbury, "Open source ARM userland," Oct. 2012. [Online]. Available: http://www.raspberrypi.org/archives/2221 106

[139] Dave Airlie, "raspberry pi drivers are NOT useful," Oct. 2012. [Online]. Available: http://airlied.livejournal.com/76383.html 106

[140] E. Anholt, "new job!" Jun. 2014. [Online]. Available: http://anholt.livejournal.com/44239.html 107

[141] F. Broaekaert, A. Fristsch, L. San, and S. Tverdyshev, "Towards power-efficient mixed critical systems," in *OSPERT 2013*, 2013, pp. 30–35. [Online]. Available: http://ertl.jp/~shinpei/conf/ospert13/OSPERT13_proceedings_web.pdf 110

[142] "ARM big.LITTLE technology explained," Apr. 2014. [Online]. Available: https://www.youtube.com/watch?v=KClygZtp8mA&feature=youtube_gdata_player 114

[143] P. McKenney, "A big.LITTLE scheduler update," Jun. 2012. [Online]. Available: https://lwn.net/Articles/501501/ 114, 133

[144] Linaro, "big.LITTLE software update," Sep. 2013. [Online]. Available: http://www.linaro.org/blog/hardware-update/big-little-software-update/ 114

[145] M. Kim, K. Kim, J. Geraci, and S. Hong, "Utilization-aware load balancing for the energy efficient operation of the big.LITTLE processor," in *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, Mar. 2014, pp. 1–4. 115

[146] M. Pricopi, T. Muthukaruppan, V. Venkataramani, T. Mitra, and S. Vishin, "Power-performance modeling on asymmetric multi-cores," in *2013 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, Sep. 2013, pp. 1–10. 115

[147] Y. Zhu and V. Reddi, "High-performance and energy-efficient mobile web browsing on big/little systems," in *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA2013)*, Feb. 2013, pp. 13–24. 115

[148] F. Broquedis, N. Furmento, B. Goglin, R. Namyst, and P.-A. Wacrenier, "Dynamic task and data placement over NUMA architectures: An OpenMP runtime perspective," in *Proceedings of the 5th International Workshop on OpenMP: Evolving OpenMP in an Age of Extreme Parallelism*, ser. IWOMP '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 79–92. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-02303-3_7 116

[149] C. Rossignon, H. Pascal, O. Aumage, and S. Thibault, "A NUMA-aware fine grain parallelization framework for multi-core architecture," in *PDSEC - 14th IEEE International Workshop on Parallel and Distributed Scientific and*

Engineering Computing - 2013, Boston, États-Unis, May 2013. [Online]. Available: http://hal.inria.fr/hal-00858350 116

[150] Runtime, INIRIA Bordeaux, "StarPU," 2008. [Online]. Available: http://runtime.bordeaux.inria.fr/StarPU/ 118

[151] C. Augonnet, S. Thibault, and R. Namyst, "Automatic calibration of performance models on heterogeneous multicore architectures," Aug. 2009. [Online]. Available: http://hal.inria.fr/inria-00421333 119

[152] L. Courtès, "C language extensions for hybrid CPU/GPU programming with StarPU," Apr. 2013. [Online]. Available: http://hal.inria.fr/hal-00807033 119

[153] M. Handley, O. Bonaventure, C. Raiciu, and A. Ford, "TCP extensions for multipath operation with multiple addresses," Jan. 2013. [Online]. Available: https://tools.ietf.org/html/rfc6824 120

[154] Olivier Bonaventure, "Apple seems to also believe in multipath TCP," Sep. 2013. [Online]. Available: https://perso.uclouvain.be/olivier.bonaventure/blog/html/2013/09/18/mptcp.html 120

[155] J. Choi, S. Govindan, J. Jeong, B. Urgaonkar, and A. Sivasubramaniam, "Power consumption prediction and power-aware packing in consolidated environments," *IEEE Transactions on Computers*, vol. 59, no. 12, pp. 1640–1654, 2010. 120

[156] J. Ylitalo, T. Jokikyyny, A. J. Tuominen, and J. Laine, "Dynamic network interface selection in multihomed mobile hosts," *HAWAII INTL. CONF. ON SYSTEM SCIENCES (HICSS*, vol. 9, p. 315, 2003. [Online]. Available: http://citeseer.uark.edu:8080/citeseerx/viewdoc/summary?doi=10.1.1.99.9062 120

[157] L. Liu, Y. Wei, Y. Bi, and J. Song, "Cooperative transmission and data scheduling mechanism of multiple interfaces in multi-radio access environment," in *Global Mobile Congress 2009*, 2009, pp. 1–5. 120

[158] B. Hu, S. Chen, W. Ye, and X. Zhan, "An autonomic interface selection mechanism for multi-interface host," in *2012 8th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM)*, 2012, pp. 1–5. 121

[159] M. Chowdhury, Y. M. Jang, C. S. Ji, S. Choi, H. Jeon, J. Jee, and C. Park, "Interface selection for power management in UMTS/WLAN overlaying network," in *11th International Conference on Advanced Communication Technology, 2009. ICACT 2009*, vol. 01, 2009, pp. 795–799. 121

[160] P.-N. Tran and N. Boukhatem, "SiPiA: The shortest distance to positive ideal attribute for interface selection," in *Telecommunication Networks and Applications Conference, 2008. ATNAC 2008. Australasian*, 2008, pp. 175–179. 121, 136

[161] M. Sowmia Devi and P. Agrawal, "Dynamic interface selection in portable multi-interface terminals," in *IEEE International Conference on Portable Information Devices, 2007. PORTABLE07*, 2007, pp. 1–5. 121, 136

[162] G. Castignani, N. Montavont, and A. Arcia-Moret, "Multi-objective optimization model for network selection in multihomed devices," in *2013 10th Annual Conference on Wireless On-demand Network Systems and Services (WONS)*, Mar. 2013, pp. 113–115. 121

[163] K. Deb, *Optimization For Engineering Design: Algorithms And Examples*, 1995. 121

[164] S. Bleuler, M. Laumanns, L. Thiele, and E. Zitzler, "PISA — a platform and programming language independent interface for search algorithms," in *Evolutionary Multi-Criterion Optimization*, ser. Lecture Notes in Computer Science, C. M. Fonseca, P. J. Fleming, E. Zitzler, L. Thiele, and K. Deb, Eds. Springer Berlin Heidelberg, Jan. 2003, no. 2632, pp. 494–508. [Online]. Available: http://link.springer.com/chapter/10.1007/3-540-36970-8_35 121, 137

[165] Martin Peres, "mupuf/RTGDE," 2013. [Online]. Available: https://github.com/mupuf/RTGDE 125, 128, 136, 138, 170

[166] J. Racine, "gnuplot 4.0: a portable interactive plotting utility," *Journal of Applied Econometrics*, vol. 21, no. 1, pp. 133–141, 2006. [Online]. Available: http://ideas.repec.org/a/jae/japmet/v21y2006i1p133-141.html 128

[167] tcpdump, "Tcpdump/libpcap public repository," 1987. [Online]. Available: http://www.tcpdump.org/ 130

[168] Tyler Andrews, "Computation time comparison between matlab and c++ using launch windows," Jun. 2012. [Online]. Available: http://digitalcommons.calpoly.edu/cgi/viewcontent.cgi?article=1080&context=aerosp 137

[169] SenseFly, "SenseFly: 07 - disaster management," 2012. [Online]. Available: https://www.sensefly.com/user-cases/07-disaster-management.html 142

[170] "OpenBTS | open source cellular infrastructure," 2010. [Online]. Available: http://openbts.org/ 142

[171] Intel, "PowerTOP," 2012. [Online]. Available: https://01.org/powertop/ 150

[172] Neil Brown, "Control groups series by neil brown," 2014. [Online]. Available: https://lwn.net/Articles/604609/ 151

[173] K. Nichols, D. L. Black, S. Blake, and F. Baker, "Definition of the differentiated services field (DS field) in the IPv4 and IPv6 headers," Dec. 1998. [Online]. Available: https://tools.ietf.org/html/rfc2474 152

[174] "Inside the linux 2.6 completely fair scheduler," Dec. 2009. [Online]. Available: http://www.ibm.com/developerworks/linux/library/l-completely-fair-scheduler/ 152

[175] Texas Instruments, Inc., "MSP430™ SoC with RF core," 2013. [Online]. Available: http://www.ti.com/lit/ds/symlink/cc430f6137.pdf 160

[176] Martin Peres and Romain Perier, "mupuf/diaforus_labri," 2012. [Online]. Available: https://github.com/mupuf/diaforus_labri 167

[177] Linus Torvalds, "Linux," 1991. [Online]. Available: http://www.kernel.org 169

[178] Ben Skeggs, "Nouveau," 2007. [Online]. Available: http://cgit.freedesktop.org/~darktama/nouveau/ 169

[179] Envytools, *Envytools documentation on readthedocs*, 2014. [Online]. Available: https://envytools.readthedocs.org/en/latest/ 169

[180] Martin Peres, "mupuf/pdaemon_trace," 2014. [Online]. Available: https://github.com/mupuf/pdaemon_trace 170

[181] Cortus, "Cortus - APS3r," 2010. [Online]. Available: http://www.cortus.com/aps3r.html 187