Wireless Process Control using IEEE 802.15.4 Protocol

AITOR HERNÁNDEZ

KTH Electrical Engineering

Masters' Degree Project
Stockholm, Sweden November 4, 2010

XR-EE-RT 2010:020

Wireless Process Control using IEEE 802.15.4 Protocol

AITOR HERNÁNDEZ

Master's Thesis
Supervisors: José Araújo,
Pangun Park and
Prof. Karl .H. Johansson
Examiner: Asst. Prof. Carlo Fischione

XR-EE-RT 2010:020

Abstract

Considering the potential benefits offered by Wireless Sensor Networks (WSNs), they have been becoming an interesting technology for process, manufacturing, and industrial control and Smart Grid applications. These applications motivate many companies, industrial communities and academy to focus and research in this direction.

The IEEE 802.15.4 is the standard proposed to be use in low-power communication of which WSN is part. Even though there are many implementations of the standard for the selected operating system, TinyOS, they are not fully validated or fully implemented. Moreover, in spite of the existence of previous studies using the protocol, there is no sufficient analysis of the performance of this standard.

In this thesis, a comparison between the two main implementations is done through the experiments to validate the feasibility of the implementations. Because of the fact that the selected implementation does not have the Guaranteed Time Slots (GTSs) mechanism developed, in this thesis are provided all the mechanisms necessary to transmit during the Contention-Free Period (CFP): allocation, expiration, reallocation and deallocation. Hence, a IEEE 802.15.4 implementation is provided with a comprehensive evaluation with which the behaviour is proven. The implementation is validated in terms of packet delivery rate and delay for different network configurations and different parameters.

Owing to no practical results for the use of this protocol in control applications, a inverted pendulum process is introduced to show the benefits in wireless process control by using the IEEE 802.15.4 in a real-time control loop process. The extensive experimental results show that packets losses and delays are the essential factors to guarantee the stability of the system. Furthermore, we also demonstrate and analyse the benefits of using this protocol in a Home Smart Grid setup.

Acknowledgements

First of all, I would like to thank my supervisor José Araújo to give me the possibility to work on this thesis. Mention that without his help, the inverted pendulum would not success as it does. Thanks for the discussions, problems, and solutions provided.

Moreover, I thank my supervisor Pangun Park for his support and guidance with the IEEE 802.15.4, without him, the understanding and validation of this protocol it would not be easy.

I want to thank my colleague João Faria for the improvements made on the pendulum and Micke for the board that he made for us.

Thanks for all the people at Automatic Control and special thanks for the people at the eighth floor. Thanks to Aziz, George, Hans, Håkan and João for the pleasant discussions and moments during the breaks and meals.

Thanks to all the friends I met here in Sweden. In special for my corridor mates at DKV with whom I spent my first months in Stockholm.

I take advantage of this opportunity to thank my family and friends in Spain for supporting me along these years. Above all, I would like to thank my parents,Carlos and Encarna, and sister María for the continuous presence and encourage me to struggle on through the years. They support me to come and make this thesis at KTH. An special thank, for all my friends who gave me greats moments, Alex, Alberto, Dani, Felipe, Ismael, Miguel, Ruben and the rest who give me such a greats moments.

Contents

Acronyms

ACK	Acknowledge packet. 16, 17, 21, 23–25, 28, 35, 40, 41, 78, 80, 85, 109, 111
ADC	Analog-Digital converter. 30, 101
AI	Analog input. 100, 101
AO	Analog output. 101
BE	Backoff Exponent. 22, 23
BI	Beacon Interval. 20
BO	Beacon Order. 20, 40, 43, 60, 62, 72, 74, 85
CA	Collision Avoidance. 111
CAP	Contention Access Period. 15, 21, 27, 44, 67, 69, 72, 74, 78, 80, 84–86, 97, 111, 114
CCA	Clear Channel Assessment. 17, 23, 45, 80, 81, 107
CFP	Contention-Free Period. iii, 15, 21, 50, 51, 62, 63, 66, 67, 69, 71, 72, 74, 78, 80, 84–86, 97, 113, 114
CISTER	Research Centre in Real-Time Computing Systems. 38
CRC	cyclic redundancy check. 17
CSMA	Carrier Sense Multiple Access. 96, 109–111, 114
CSMA-CA	Carrier Sense Multiple Access with Collision Avoidance. 15–19, 21–23, 39, 41–44, 78, 80, 81, 84, 96, 97, 105, 111
CW	Contention Window (length). 22, 23
DAC	Digital-Analog converter. 30, 101
DAQ	Data Acquisition. 96, 97, 100, 101, 124
DC	Direct current. 99
DMA	Direct memory access. 30
DSSS	direct-sequence spread spectrum. 17

ED	Energy detection. 17
EOM	Equations of Motion. 97, 99
ETSI	European Telecommunication Standards Institute. 55
FCS	Frame Check Sequence. 124
FFD	Full-Function Device. 12, 14, 110
GTS	Guaranteed Time Slot. iii, 9, 15, 18, 19, 21, 25–28, 42, 49–53, 55, 56, 59, 62, 63, 65–69, 71–74, 77–80, 84–86, 97, 113, 114
IP	Internet Protocol. 34
IPP	Polytechnic Institute of Porto. 38, 39, 113
ISEP	Research Unit based at the School of Engineering. 38, 39
ISM	Industrial Scientific Medical. 18, 86
ISO	International Organization for Standardization. 55
ITU-T	International Telecommunication Union - Telecommunication Standardization Sector. 17
LED	Light-Emitting Diode. 32
LQ	linear-quadratic. 104
LQI	Link Quality Indication. 17
LQR	linear-quadratic regulator. 99, 101, 105, 122, 123
LR-WPAN	low rate WPAN. 12, 14, 17
MAC	Medium Access Control. 9, 12, 16–19, 21, 23–26, 40, 42, 56, 62, 77, 83–86, 114
MCPS	MAC Common Part Sublayer. 19
MCPS-SAP	MAC Common Part Sublayer - Service Access Point. 18, 19
MCU	microcontroller. 31, 43, 44, 114
MHR	MAC Header. 32
MLME	MAC Sublayer Management Entity. 19, 25–28
MLME-SAP	MAC Sublayer Management Entity - Service Access Point. 18, 20
NB	Number of backoff periods. 22, 23
NI	National Instruments. 100, 101

O-QPSK	orthogonal - quadrature phase-shift keying. 18
OPAM	Operational Amplifier. 123
OS	operating system. 9, 29, 32, 35, 36, 39
OSI	open systems interconnection. 12
P2P	peer-to-peer. 13, 14, 16
PAN	Personal Area Network. 12–15, 17, 18, 21, 25–28, 32, 36, 39, 43, 44, 52, 63, 73, 78, 96, 110
PC	personal computer. 13, 97, 100, 110, 122
PD-SAP	PHY Data - Service Access Point. 19
PHY	Physical Layer. 12, 17–19, 21, 23, 24
PIB	PAN Information Database. 64, 68
PLME-SAP	PHY Sublayer Management Entity - Service Access Point. 19
QoS	Quality of Service. 7
RAM	random access memory. 30, 33
RF	Radio Frequency. 12
RFD	Reduced-Function Device. 12–14, 19, 110
ROM	read-only memory. 33, 43
RSSI	Received Signal Strength Indication. 87
SAP	Service Access Point. 18
SD	Superframe Duration. 21
SICS	Swedish Institute of Computer Science. 34, 39
SMA	SubMiniature version A. 31
SO	Superframe Order. 21, 60, 62, 72–74
SPI	Serial Peripheral Interface. 100
SSCS	service-specific convergence sublayer. 19
SSSUP	Scuola Superiore Sant'Anna di Studi Univertari e di Perfezionamento. 39
TCP	Transport Control Protocol. 123
TDMA	Time Division Multiple Access. 110, 111
TEP	TinyOS Extension Proposal. 33, 37
USB	Universal Serial Bus. 30–32, 122, 123
WLAN	Wireless Local Area Network. 6
WPAN	Wireless Personal Area Network. 6
WPC	wireless process control. iii, 9, 96, 108, 114

WSN	Wireless Sensor Network. iii, 5–9, 29, 30, 32, 33, 38, 81

Chapter 1

Wireless sensors networks

1.1 Introduction

In the late 1800's, the first successful wireless radio transmission was accomplished by the Italian Guglielmo Marconi. It was by September 1895 when Marconi had already built the equipment that transmitted electrical signals through the air. He opened the door towards a world that we are still discovering.

For most people the significance of wireless technologies comes from the ability to provide communication services like voice/data without any restriction on movement. The mobility and the no necessity of being wired connected is perceived as the main characteristics of the wireless technologies. However, it gives us the possibility of scaling a network, communicate between two points which could be impossible with wires, and the independence of being always connected.

With the success of wireless technologies in consumer electronics, standard wireless technologies are envisioned for the deployment in industrial environments as well. [45, 46] Wireless technologies have also been identified for industrial and factory automation, distributed control systems, automotive systems and other kinds of networked embedded systems with high mobility, reduced cabling and installation cost, reduce danger of breaking cables, and less hassle with connectors. Nevertheless, these applications must often satisfy tight real-time and reliability requirements otherwise non-productivity, loss of time, money or even physical damage can result. In wired environments, timing and reliability are well attended by bus systems and protocols, which are a mature technology. When wireless links are included, reliability and timing requirements are slightly more difficult to achieve, due to the characteristics of the radio channels.

WSNs have recently received increased attention in the industrial applications, fact that led many companies, industrial communities and academy to focus and

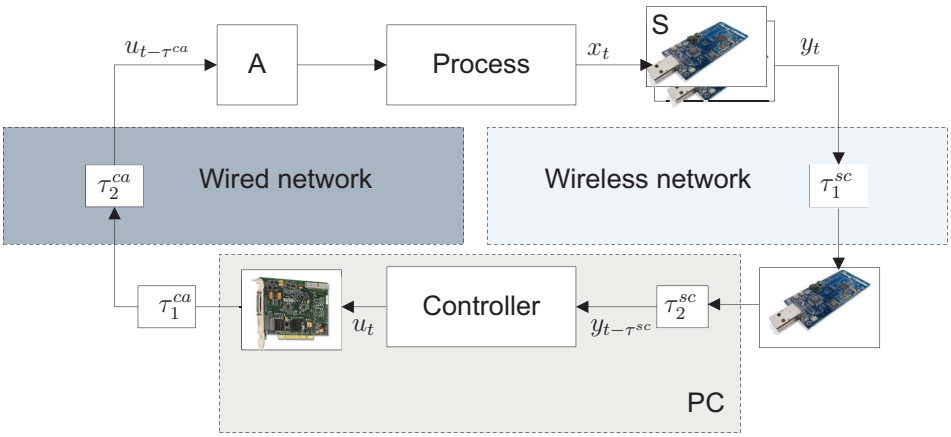


Figure 1.1: Inverted pendulum diagram

research in this direction. WSNs support much lower data rates and much smaller transmit powers than other kind of Wireless Local Area Network (WLAN). This Wireless Personal Area Network (WPAN), due to a limited energy budget sensors should spend most of their time in a sleep state in which they are not able to transmit or receive data. Even though these properties do not favour the adoption of sensor networks in tight control loops, we apply a protocol for WSN in a control loop where a high sample rate is needed in comparison with monitoring tasks.

This implementation will help us understanding the challenges of performing real-time control over wireless which is not well studied.

1.2 Motivating applications

WSN can be applied in two realistic scenarios, for Smart Grid technology and process control, like home smart grid to monitor the consumption of different devices, or for controlling the air conditioning system. We could find other applications of WSN in [1, 27, 34].

1.2.1 Process control

The main purpose of this thesis is to show how wireless technology can become a key in process control using the IEEE 802.15.4. In comparison to traditional wired sensors, wireless sensors provide advantages in the manufacturing environment, such as an increased flexibility for locating and reconfiguring sensors, elimination of wires in potentially hazardous locations, and easier maintenance.

Figure 1.1 shows the block diagram of our inverted pendulum. Our research

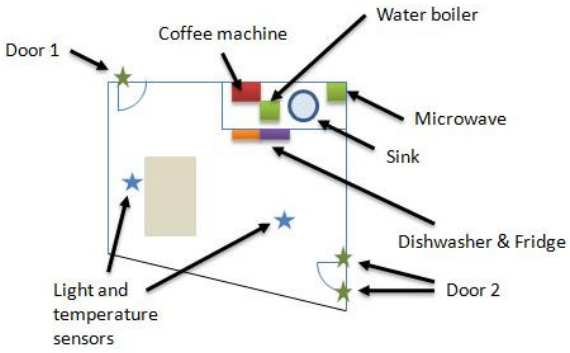


Figure 1.2: Home Smart Grid deployment in a kitchen. Source: [33]

presented on this thesis has been based on the demonstration that it was possible to control a tight timing control process using the IEEE 802.15.4. The full description about the process is shown in Section 6.1.

1.2.2 Smart Grid technology

One interesting application of WSNs is to build a smart-grid infrastructure. WSNs are an integral part of the automated metering infrastructure and smart-grid plans. Smart Grid technology is designed to allow customers and utility companies to collaboratively manage power generation, delivery and storage.

One case of study has been the research made in [33]. The research involves the programming and deployment of several motes in a kitchen. It shows an example of how it could work in a home or wherever we want to apply this smart-grid technology. It shows also how it is possible to analyse and detect patterns and user profiles.

Figure 1.2 shows an example of a deployment in a kitchen. Mention that huge number of motes placed in a reduced space could be difficult and hard to deploy in case of using wires. This example was tested with the standard protocol IEEE 802.15.4 [12]. In Section 6.2, we show the results on the communication analysis for this application.

1.3 Challenges of WSN

The protocol design for industrial control applications with WSNs encounters much more challenges than the protocol design for traditional communications networks, where, mainly, considering a good Quality of Service (QoS) with a good throughput

is enough. [27, 44] However, in our case of study appears more challenges to deal with:

- **Reliability:** Sensor readings must be sent/received with a high probability of success, because missing sensors readings could be critical. However, despite the possibility of maximizing the reliability it is necessary a trade-off between reliability and the network energy consumption.

- **Delay:** Sensor information must reach the sink within some deadline. Time delay is very important since it influences on performance and stability of an industrial control system. Even if an outdated packet is received it will not be generally useful for a control application. This plays an important role in our control applications (see Chapter 6).

- **Energy Efficiency:** The lack of battery replacement, which is essential for affordable WSN deployment, requires energy-efficient operations.

- **Scalability** The protocol/application should be able to adapt to variation in the network size, for example, size variations caused by the addition of new nodes. This characteristic is one of the biggest advantages of low-power wireless nodes.

All of these challenges are based on theoretical aspects to take into account when it is needed to design a protocol or evaluate possible applications. As a consequence, we can see how the design of such networked systems has to take into account a large number of factors that ensure correct behaviour. Starting from these requirements, it is important to design an efficient communication protocol that satisfies the constraints and optimizes the energy consumption.

On the other hand, once we have designed a good protocol, then the challenges for the implementation appear. Below we show some of the special requirements to take into account once we want to work with a real deployment with WSN [38, 14]:

- **Limited resources:** Motes have very limited physical resources, due to the goals of small size, low cost and a low power consumption. The motes that we use (TelosB/TmoteSky [7, 41]) only have 48k ROM and a microprocessor of 8MHz.

- **Adaptation:** The network operation should adapt to application requirement changes, time-varying wireless channels, and variations of the network topology. For instance, the set of application requirements may change dynamically and the communication protocol must adapt its parameters to satisfy the specific requests of the control actions.

1.4 Outline

This thesis shows the steps and progress to someone who wants to use a device for wireless process control (WPC). From an external observer of a system (who does not know anything about sensors network and process control) it is easy to detect two characteristics that the system has to satisfy: a low number of data losses and a high determinism for the arrival packets.

Once the characteristics and limitations of the system are known, it is obvious the research of a platform that offers us durability and the expansion of possibility to wide functionalities. At this point some questions appear: Which platform is the most suitable for our application? Is there any operating system already designed, or it is better to use a dedicated system? Which operating system (OS)? Which dedicated system?

First of all, in Chapter 2 we focus on a summary of the protocol that is going to be used and analysed along the thesis, the IEEE 802.15.4 [12, 26]. We find a brief description of the standard utilities, primitives and configuration. It does not intend to be a reference and wide explication of the protocol, it is just for a quick review and make this thesis more understandable in some points.

In Chapter 3, we get the answer for the previous questions. First we present some devices available to use in WSN, and other hardware necessary to analyse the communication protocol. After that, we show a list of OSs that could run in our sensors and we present our choice. Moreover we discuss which is the best IEEE 802.15.4 implementation for the selected OS. The performance evaluation of the IEEE 802.15.4 depends on how good is the implementation and how close it is to the standard. Furthermore, we analyse the precision and accuracy of the timers with the different implementations.

Chapter 4 intends to be a reference guide for the GTS implementation. As a complement of the TKN15.4 documentation [15], we provide a complete guide with the current state of the implementation. This guide consists, first of all, a brief introduction and overview with the current state of the implementation, then we show a decomposition of the components and comments about the radio arbitration, timing issues, and utilities. At the end of this chapter, we show some discussions fields for future work, in order to give an advice on the next steps towards a full implementation.

Chapter 5 shows a performance evaluation of the selected implementation. We analyse reliability and delay for the selected implementation and for our GTS. We shows these characteristics comparing with different configurations. First of all, we compare the two transmission modes that the IEEE 802.15.4 has. Moreover, we analyse the influence of the Medium Access Control (MAC) parameters in order to characterize which are the critical parameters that influences in the reliability and delay. Once we have the TKN15.4 validated, we focus on the analysis of transmis-

sions using our GTS implementation. To end up, we show a brief validation where we compare some graphs with theoretical model, and some of the future work that would be made in order to improve and get an extensive performance evaluation.

Finally, Chapter 6 shows some applications where this protocol is applied to provide more benefits. An inverted pendulum process where the sensing is done through wireless and [33] a home smart grid. More projects related to this protocol in our lab could be found in [24].

Chapter 2

IEEE 802.15.4 Protocol

In this chapter, we show a brief description of the IEEE 802.15.4 to facilitate the understanding of this thesis.

The main features of this standard are network flexibility, low cost, very low power consumption and low data rate in a ad-hod self-organizing network among inexpensive fixed, portable and moving devices. It has been developed for applications with relaxed throughput requirements which can not handle the power consumption of heavy wireless protocol stacks.

2.1 General description

2.1.1 Components of the IEEE 802.15.4 WPAN

The most basic component in IEEE 802.15.4 is the device. A device can be a Full-Function Device (FFD) or Reduced-Function Device (RFD). A network shall include at least one FFD, operating as a Personal Area Network (PAN) coordinator.

The FFD can operate in three modes: PAN coordinator, coordinator or device. An RFD is intended for applications that are extremely simple and do not need to send large amounts of data. An FFD can talk to RFDs or FFDs while RFDs can only talk to an FFDs.

2.1.2 Architecture

The IEEE 802.15.4 architecture is defined in terms of a number of blocks in order to simplify the standard. These blocks are called layers. Each layer is responsible for one part of the standard and offer services to the higher layers. The layout of the blocks is based on the open systems interconnection (OSI) seven-layer model.

Low rate WPAN (LR-WPAN) device comprises a Physical Layer (PHY), which contains the Radio Frequency (RF) transceiver along with its low-level control mechanism, and a MAC sublayer that provides access to the physical channel for all types of transfer. Figure 2.1 shows a graphical representation of these blocks.

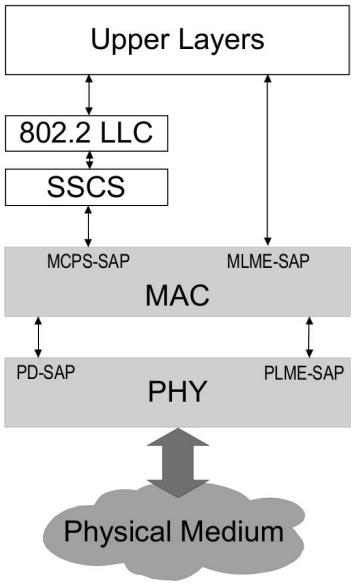


Figure 2.1: LR-WPAN device architecture Source: [26]

2.1.3 Network topologies

A combination of different components could generate 3 types of topologies that this standard supports. Figure 2.2 shows these combinations.

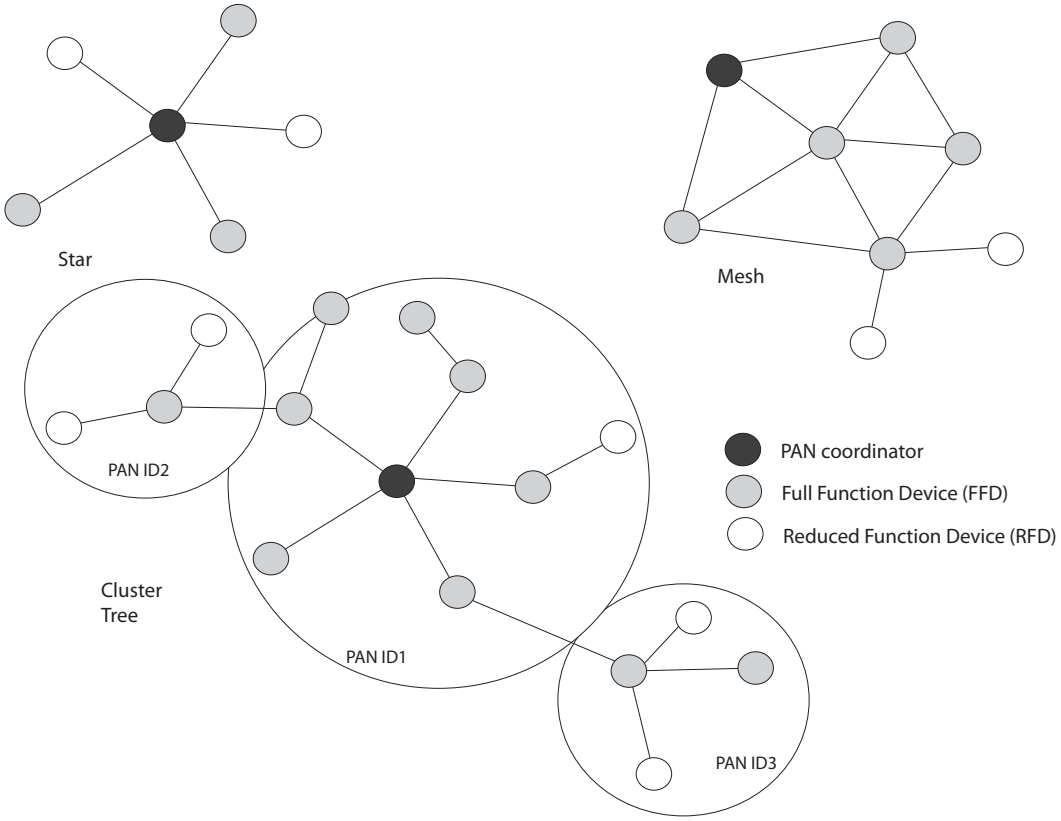


Figure 2.2: Topology models

Star topology The communication is established between devices and a single central PAN coordinator. Applications that benefit from this topology include home automation, personal computer (PC) peripherals, toys and games. All the experiments and simulations in this thesis are based on this topology, with one PAN coordinator and several RFD.

Mesh topology (peer-to-peer (P2P) topology) There is also one PAN coordinator. In contrast to star topology, any device can communicate with any other device as long as they are in range of one another. Applications such as industrial control and environmental monitoring, asset and inventory tracking would benefit from such a topology.

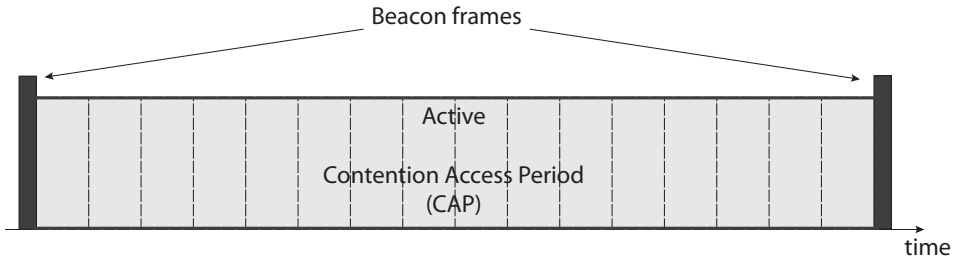
Cluster-tree topology It is a special case of a P2P network in which most devices are FFDs and RFD may connect to a cluster-tree network as a leave node at the end of a branch. Any of the FFD can act as a coordinator and provide synchronization services to other devices and coordinators. However, only one of these coordinators is the PAN coordinator.

2.1.4 Functional overview

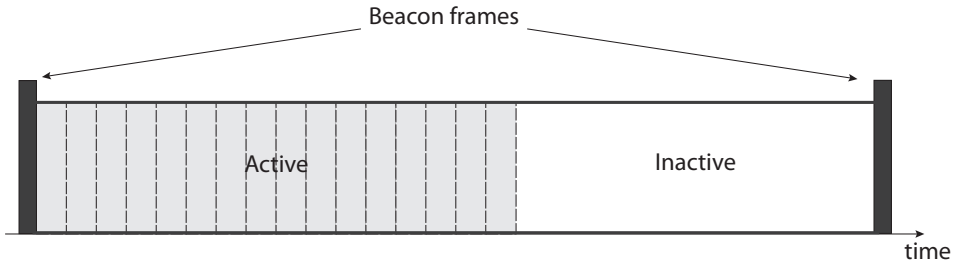
A brief overview of the general functions of a LR-WPAN is given in this section and includes information on the superframe structure, the data transfer model, the frame structure, improving probability of successful delivery and power consumption considerations. We skip the security services because it is out of the scope of this thesis.

2.1.4.1 Superframe structure

This standard allows the optional use of a superframe structure. The format of the superframe is defined by the coordinator, it is divided into 16 equally sized plots. Optionally, the superframe can have an active and an inactive portion, shown in Figure 2.4. During the inactive portion, the coordinator and all the devices may enter a low-power mode.



(a) Superframe without Inactive period



(b) Superframe with Inactive period

Figure 2.3: Superframe Structure

The beacons are used to synchronize the attached devices, to identify the PAN, and to describe the structure of the superframes. Any device wishing to communicate during the Contention Access Period (CAP) competes with other devices using a slotted Carrier Sense Multiple Access with Collision Avoidance (CSMA-CA) mechanism. Moreover, the PAN may dedicate portions of the active superframe for GTSs. The GTSs form the CFP, which always appears at the end of the active superframe starting at a slot boundary immediately following the CAP.

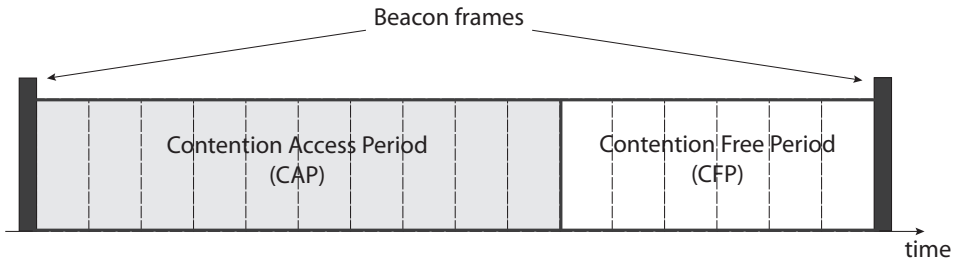
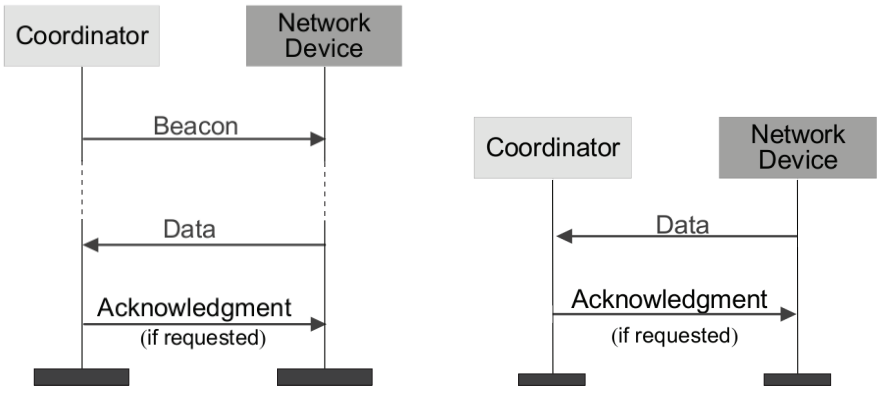


Figure 2.4: Superframe Structure with CFP

2.1.4.2 Data transfer model

There are three types of data transfer. The mechanism for each transfer type depend on whether the network supports the transmission of beacons. A beacon-enabled PAN is used in networks that either require synchronization. If the network does not need synchronization can elect not to use the beacon for normal transfers.

1. **Data transfer to a coordinator**



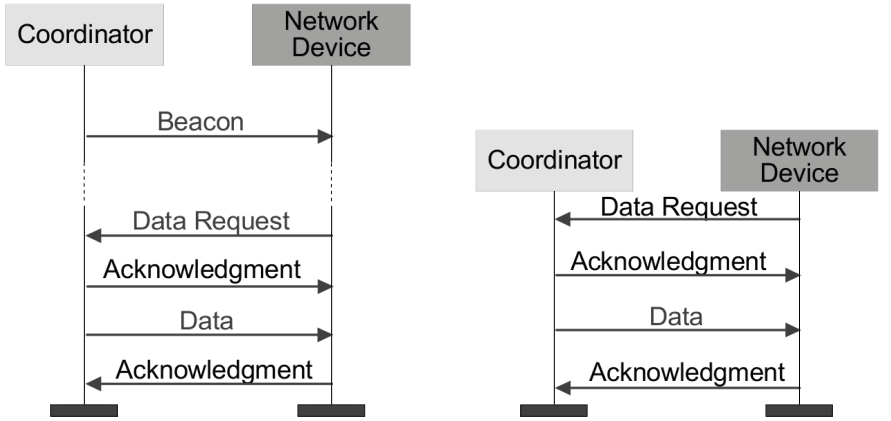
(a) To coordinator in a beacon-enabled PAN

(b) To coordinator in a non beacon-enabled PAN

Figure 2.5: Communication to a coordinator

Figure 2.5 shows the communication from a device to a coordinator for slotted and unslotted CSMA-CA. The transmission can be done when the device has a packet ready to send. It does not need to wait any kind of instruction, like in the following case.

2. **Data transfer from a coordinator**



(a) From coordinator in a beacon-enabled PAN

(b) From coordinator in a non beacon-enabled PAN

Figure 2.6: Communication from a coordinator

Figure 2.6 shows the sequence for the communication from coordinator to device. As far as the device does not have the radio chip enabled for reception all the time, is needed a special mechanism, that allows the device to know when it has to enable the reception.

3. **Peer-to-peer data transfers** In a P2P, every device may communicate with every other device in its radio sphere of influence. The device can simply transmit its data using unslotted CSMA-CA.

2.1.4.3 Frame structure

The frame structure depends on the protocol layer which add to the structure with layer-specific headers and footers. This standard defines four frame structures:

Beacon frame Used by a coordinator to transmit beacons and set the network configuration

Data frame Used for all the transfers data

Acknowledge packet (ACK) frame Used for conforming successful frame reception

MAC command frame Used for handling all MAC peer entity control transfers

2.1.4.4 Improving probability of successful delivery

The IEEE 802.15.4 LR-WPAN employs various mechanisms to improve the probability of successful data transmission. These mechanism are:

CSMA-CA LR-WPAN uses two types of channel access mechanisms: unslotted CSMA-CA for non beacon-enabled PANs and slotted CSMA-CA for beacon-enabled PANs. These mechanisms minimize the probability of packet collisions. A detailed explanation is shown in Section 2.3.2.1

Frame acknowledgement A successful reception and validation of a data or MAC command frame is optionally confirmed with an ACK. If the sender does not receive an ACK after some period, it assumes that the transmission was unsuccessful and retries the frame transmission.

Data verification In order to detect bit errors, an FCS mechanism employing a 16-bit International Telecommunication Union - Telecommunication Standardization Sector (ITU-T) cyclic redundancy check (CRC) is used to detect errors in every frame.

2.1.4.5 Power consumption considerations

The protocol has been developed to favour battery-powered devices. However, in certain applications, some of these devices could potentially be powered.

2.2 Physical sublayer specification

The PHY is responsible for the following task:

- Activation and deactivation of the radio transceiver
- Energy detection (ED) within the current channel
- Link Quality Indication (LQI) for received packets
- Clear Channel Assessment (CCA) for CSMA-CA
- Channel frequency selection
- Data transmission and reception

The standard defines four PHYs, but in our case, we only use one of those, because our radio chip is limited to this one. A 2450 MHz DSSS PHY employing

Figure 2.7: Modulation and spreading functions

O-QPSK modulation. This band is part of the Industrial Scientific Medical (ISM) band, which could be interfered with devices operating in the same frequency.

Our PHY operates in 2450 MHz, actually use a range between 2400-2483.5 MHz, providing a bit rate of 250 kb/s with a symbol rate of 62.5 ksymbols/s (4 bits/symbol). Figure 2.7 shows the functional block diagram.

2.3 MAC sublayer specification

This clause specifies the MAC sublayer of this standard. The MAC sublayer handles all access to the physical radio channel and is responsible for the following tasks:

- Generating network beacons if the device is a coordinator
- Synchronizing to network beacons
- Supporting PAN association and disassociation
- Supporting device security
- Employing the CSMA-CA mechanism for channel access
- Handling and maintaining the GTS mechanism
- Providing a reliable link between two peer MAC entities

2.3.1 MAC sublayer service specification

The MAC sublayer provides two services, accessed through two Service Access Points (SAPs):

- The MAC data service, accessed through the MAC Common Part Sublayer - Service Access Point (MCPS-SAP)
- The MAC management service, accessed through the MAC Sublayer Management Entity - Service Access Point (MLME-SAP)

These two services provide the interface between the service-specific convergence sublayer (SSCS) and the PHY, via the PHY Data - Service Access Point (PD-SAP) and PHY Sublayer Management Entity - Service Access Point (PLME-SAP) interfaces. In addition to these external interfaces, an implicit interface also exists between the MAC Sublayer Management Entity (MLME) and the MAC Common Part Sublayer (MCPS) that allows the MLME to use the MAC data service.

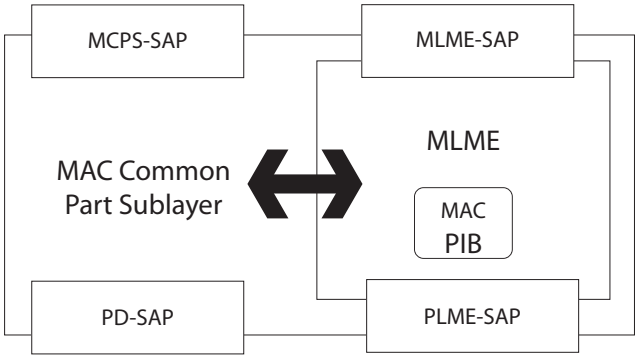


Figure 2.8: MAC sublayer reference model

Below, the main primitives of MLME and MCPS are shown with a brief description.

2.3.1.1 MCPS primitives

Table 2.1 lists the primitives supported by the MCPS-SAP. Note that primitives marked with a diamond (◇) are optional for an RFD.

MCPS-SAP primitive	**request**	**confirm**	**indication**
MCPS-DATA	✓	✓	✓
MCPS-PURGE	✓◇	✓◇	

Table 2.1: MCPS-SAP primitives

2.3.2 MAC functional description

In this subclause, we provide a brief description of the MAC functionality. Section 2.3.2.1 describes the following two mechanism for channel access: contention based and contention free. It describes the superframe structure and the CSMA-CA mechanism. Section 2.3.2.2 shows the transmission scenarios. And Section 2.3.2.3 shows the mechanism for allocating and deallocating a GTS.

MLME-SAP primitive	request	indication	response	confirm
MLME-ASSOCIATE	✓	✓◇	✓◇	✓
MLME-DISASSOCIATE	✓	✓		✓
MLME-BEACON-NOTIFY		✓		
MLME-GET	✓			✓
MLME-GTS	✓*	✓*		✓
MLME-ORPHAN		✓◇	✓◇	
MLME-RESET	✓			✓
MLME-RX-ENABLE	✓*			✓*
MLME-SCAN	✓			✓
MLME-COMM-STATUS		✓		
MLME-SET	✓			✓
MLME-START	✓◇			✓◇
MLME-SYNC	✓*			
MLME-SYNC-LOSS		✓		
MLME-POLL	✓			✓

Table 2.2: MLME-SAP primitives

2.3.2.1 MAC channel access

This subclause describes the mechanism for accessing the physical radio channel.

Superframe structure The structure of the superframe is described by the values of *macBeaconOrder* and *macSuperframeOrder*. *macBeaconOrder* describes the interval at which the coordinator shall transmit its beacon frames. The value *macBeaconOrder*, Beacon Order (BO) and the Beacon Interval (BI), are related as follows,

$$\begin{aligned} BI &= aBaseSuperframeDuration * 2^{BO} \\ &= aBaseSlotDuration * aNumSuperframeSlots * 2^{BO} \end{aligned} \tag{2.1}$$

where $0 \leqslant BO \leqslant 14$. The value *macSuperframeOrder*, Superframe Order (SO) and the Superframe Duration (SD), are related as follows,

$$\begin{aligned} SD &= aBaseSuperframeDuration * 2^{SO} \\ &= aBaseSlotDuration * aNumSuperframeSlots * 2^{SO} \end{aligned} \tag{2.2}$$

where $0 \leqslant SO \leqslant BO \leqslant 14$.

The active portion of each superframe shall be divided into *aNumSuperframeSlots* equally spaced slots of duration $aBaseSlotDuration * 2^{SO}$ and is composed of three parts: a beacon, a CAP and a CFP.

All frames, except ACK and any data frame that quickly follow the ACK of a data request command, transmitted in the CAP shall use a slotted CSMA-CA mechanism to access the channel.

Transmission within the CFP shall not use a CSMA-CA mechanism to access the channel.

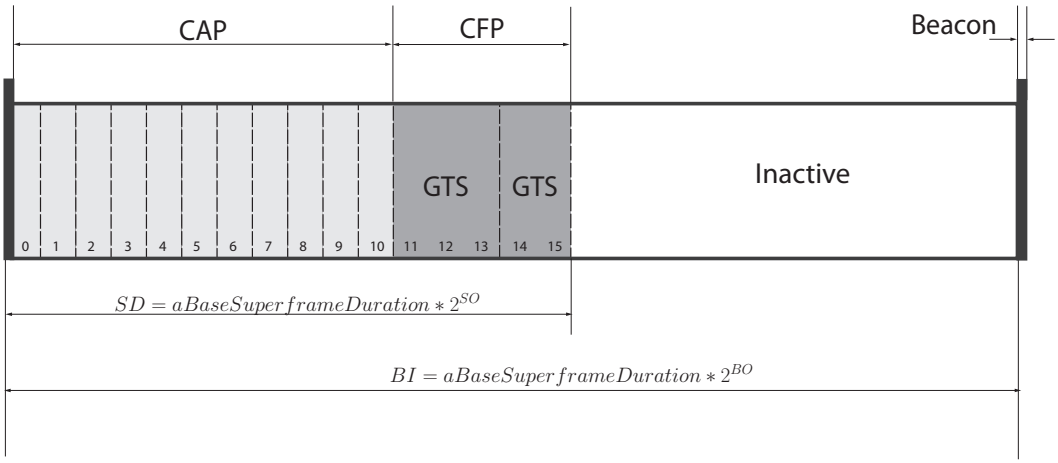


Figure 2.9: An example of the superframe structure

An example of a superframe structure is shown in Figure 2.9. In this case the $BO > SO$, and we have two GTS allocated.

CSMA-CA In slotted CSMA-CA, the backoff period boundaries of every device in the PAN shall be aligned with the superframe slot boundaries of the PAN coordinator. The MAC sublayer shall ensure that the PHY starts all of its transmissions on the boundary of a backoff period. On the other hand, in unslotted CSMA-CA, the backoff periods of one device are not related in time to the backoff periods of any other device in the PAN.

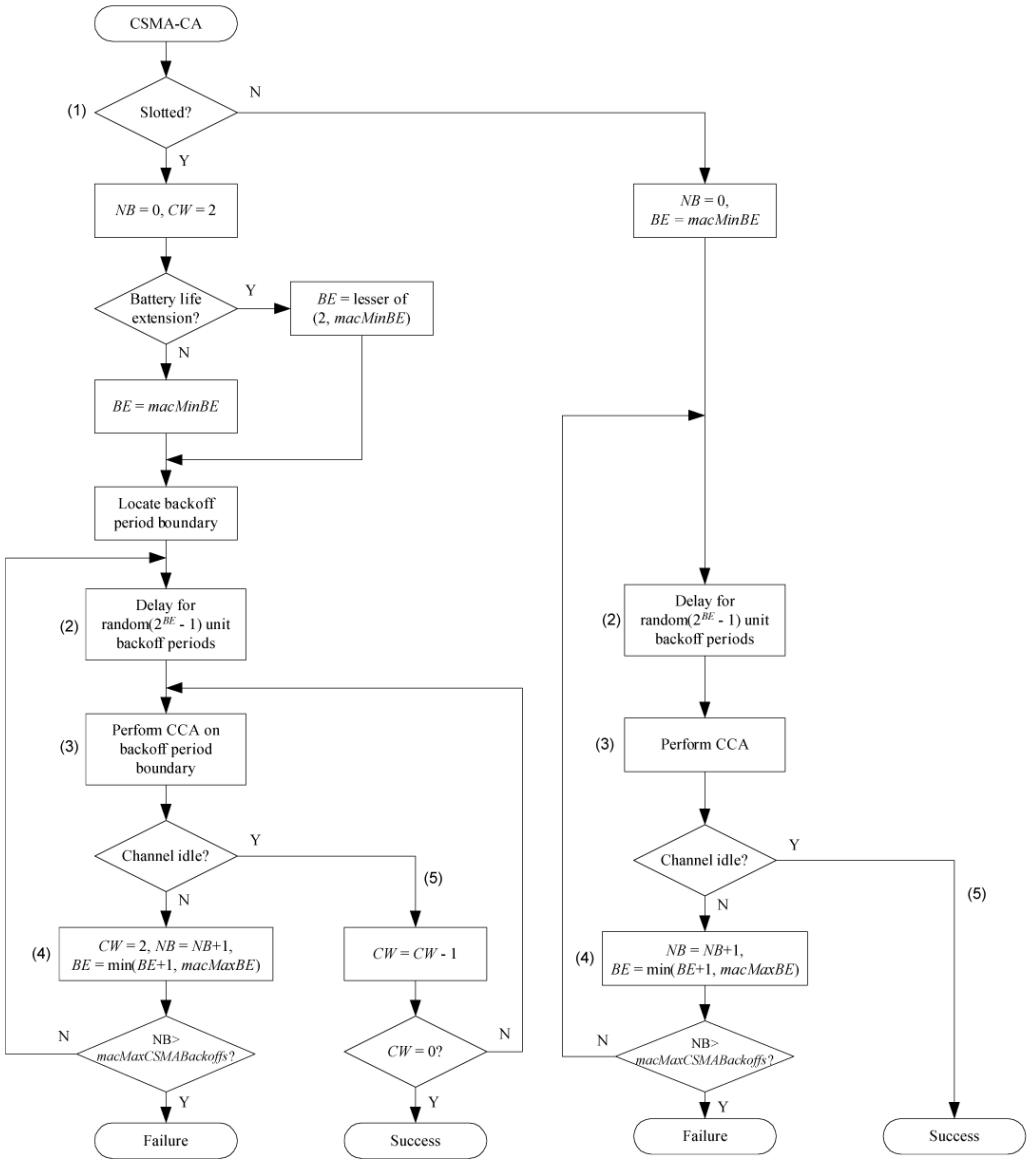


Figure 2.10: CSMA-CA algorithm Source: [26]

Each device shall maintain three variables for each transmission attempt: NB, CW and BE. NB is the number of times the CSMA-CA algorithm was required to backoff while attempting the current transmission. Contention Window (length) (CW) is the contention window length, defining the number of backoff periods that need to be clear of channel activity before the transmission can commence. Note that the CW variable is only used for slotted CSMA-CA. Backoff Exponent (BE) is

the backoff exponent, which is related to how many backoff periods a device shall wait before attempting to assess a channel.

Figure 2.10 illustrates the step of the CSMA-CA algorithm. When using slotted CSMA-CA, the MAC sublayer shall first initialize Number of backoff periods (NB), CW, and BE and then locate the boundary of the next backoff period, step (1). For unslotted CSMA-CA, the MAC sublayer shall initialize NB and BE and the proceed directly to step (2).

The MAC sublayer shall delay for a random number of complete backoff period in the range 0 to $2^{BE} - 1$, step (2), and then request that the PHY perform a CCA, step (3). In a slotted CSMA-CA system, the CCA shall start on a backoff period boundary. In an unslotted CSMA-CA system, the CCA shall start immediately.

If channel is assessed to be busy, step (4), the MAC sublayer shall increment both NB and BE by one, ensuring that BE shall be no more than *macMaxBE*. The MAC sublayer in a slotted CSMA-CA system shall also reset CW to two. If the value of NB is less than or equal to *macMaxCSMABackoffs*, the CSMA-CA algorithm shall return to step (2). If the value of NB is greater than *macMaxCSMABackoffs*, the CSMA-CA algorithm shall terminate with a channel access failure status.

If the channel is assessed to be idle, step (5), the MAC sublayer in a slotted CSMA-CA system shall ensure that the CW has expired before commencing transmission. To do this, the MAC sublayer shall first decrement CW by one and then determine whether it is equal to zero. If it is not equal to zero, the CSMA-CA algorithmic shall return to step (3). If it is equal to zero, the MAC sublayer shall begin transmission of the frame on the boundary of the next backoff period. If the channel is assessed to be idle in an unslotted CSMA-CA system, the MAC sublayer shall begin transmission of the frame immediately.

2.3.2.2 Transmission scenarios

Due to imperfect nature of the radio medium, a transmitted frame does not always reach its intended destination.

Successful data transmission (1) The originator MAC sublayer transmits the data frame to the recipient via the PHY data service. In waiting for an ACK, the originator MAC sublayer starts a timer that will expire after *macAckWaitDuration* symbols. (2) The recipient MAC sublayer receives the data frame, send an ACK back to the originator , and passes the data frame to the next higher layer. (3) The originator MAC sublayer receives the ACK from the recipient before its timer expires and the disables and reset the timers. The originator MAC sublayer issues a success confirmation to the next higher layer.

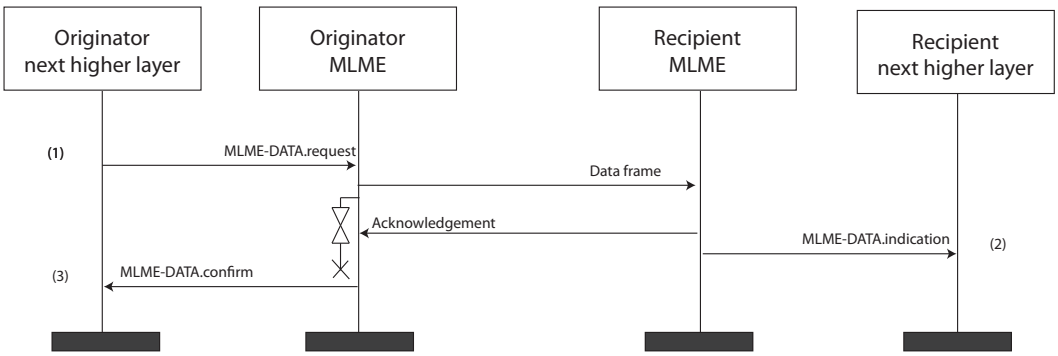


Figure 2.11: Successful data transmission

Lost data frame (1) The originator MAC sublayer transmits the data frame to the recipient via the PHY data service. The reception MAC sublayer does not receive the data frame and so does not respond with an ACK. The timer of the originator expires before an ACK is received; therefore, the data transfer has failed. If the transmission was direct, the originator retransmit the data, and this entire sequence may be repeated up to a maximum of *macMaxFrameRetries* times. If the transmission was indirect, the data frame will remain in the transaction queue until either another request for the data is received and correctly ACK or until *macTransactionPersistenceTime* is reached. (2) The `MLME_DATA.confirm` is signalled if the transmission has failed or success.

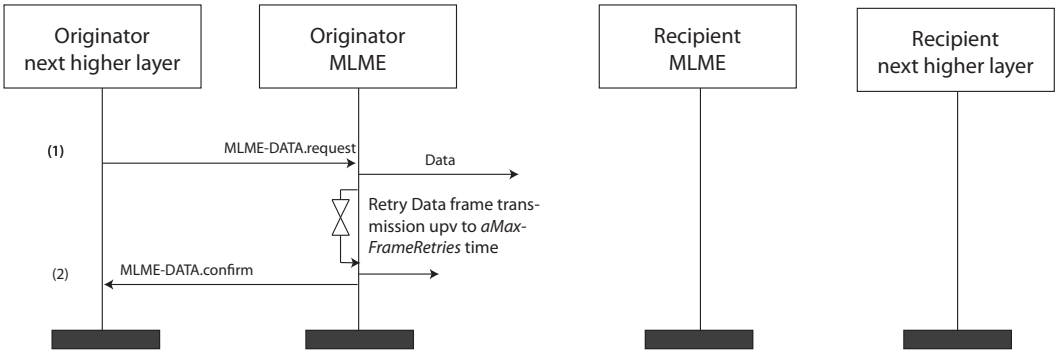


Figure 2.12: Lost data frame

Lost acknowledgement frame (1) The originator MAC sublayer transmits the data frame to the recipient via the PHY data service. (2) The recipient MAC sublayer receives the data frame, sends an ACK back to the originator, and passes the data frame to the next higher layer. (3) The originator MAC does not receive

the ACK frame, and its timer expires. Therefore the data transfer has failed. If the transmission was direct, the originator retransmit the data frame.

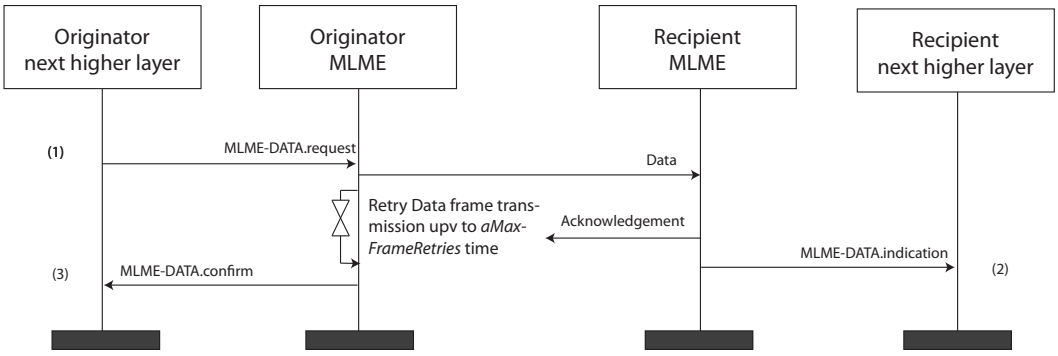


Figure 2.13: Lost acknowledgement frame

2.3.2.3 GTS allocation and management

A GTS allows a device to operate on the channel within a portion of the superframe that is dedicated exclusively to that device and it shall be used only for communications between the PAN coordinator and a device associated with the PAN. A single GTS may extend over one or more superframe slots. The PAN coordinator may allocate up to seven GTSs at the same time, if there is sufficient capacity in the superframe.

GTS allocation Figure 2.14 show the mechanism to allocate a GTS.

(1) A device is instructed to request the allocation of a new GTS through the `MLME-GTS.request` primitive with the GTS characteristics set according to the requirements of the intended application. (2) On reception of a GTS request command indicating a GTS allocation request, the PAN coordinator shall first check if there is available capacity in the current superframe. (3) On reception of the ACK to the GTS request command, the device shall continue to track beacons and wait for at most *aGTSDesPersistenceTime* superframes. (4) If the GTS descriptor is received, the MLME shall notify the next layer of the success with `MLME-GTS.indication` with *SUCCESS* status, if not, it shall indicates the failure with `MLME-GTS.indication`, indicating the *NO_DATA* status.

GTS usage When the MAC sublayer of a device that is not the PAN coordinator receives an `MCPS-DATA.request` primitive with TxOptions parameter indicating a GTS transmission, it shall determine whether it has a valid transmit GTS. If a valid GTS is found, the MAC sublayer shall transmit the data during the GTS.

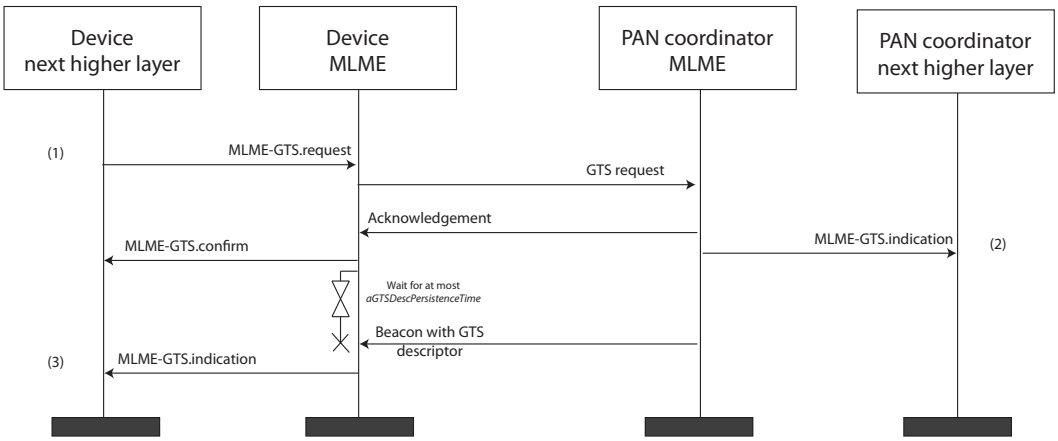


Figure 2.14: MLME-GTS allocation mechanism

It the device has any receive GTSs, the MAC sublayer of the device shall ensure that the receiver is enabled at a time prior to the start of the GTS and for the duration of the GTS.

When the MAC sublayer of the PAN coordinator receives an `MCPS.DATA.request` primitive with TxOptions parameter indicating a GTS transmission, it shall determine whether it has a valid received GTS corresponding to the device with the requested destination address. If a valid GTS is found, the PAN coordinator shall defer the transmission until the start of the receive GTS.

For all allocated transmit GTSs (relative to the device), the MAC sublayer of the PAN coordinator shall ensure that its receiver is enabled at a time prior to the start and for the duration of each GTS.

GTS deallocation The GTS deallocation could be initiated either by the coordinator or the device.

Figure 2.15 show the mechanism to deallocate a GTS initiated by the device. (1) A device is instructed to request the deallocation of an existing GTS through the `MLME-GTS.request` primitive using the characteristics of the GTS it wishes to deallocate. From this point onward the GTS to be deallocated shall not be used by the device, and its stored characteristics shall be reset. (2) On the reception of a GTS request command with the Characteristics Type subfield of the GTS Characteristics field set to zero (GTS deallocation), the PAN coordinator shall attempt to deallocate the GTS. If the GTS characteristics contained in the GTS request command match the characteristics of a know GTS, the MLME of the PAN coordinator shall deallocate the specified GTS and notify the higher layer.

GTS deallocation may be initiated by PAN coordinator due to a deallocation

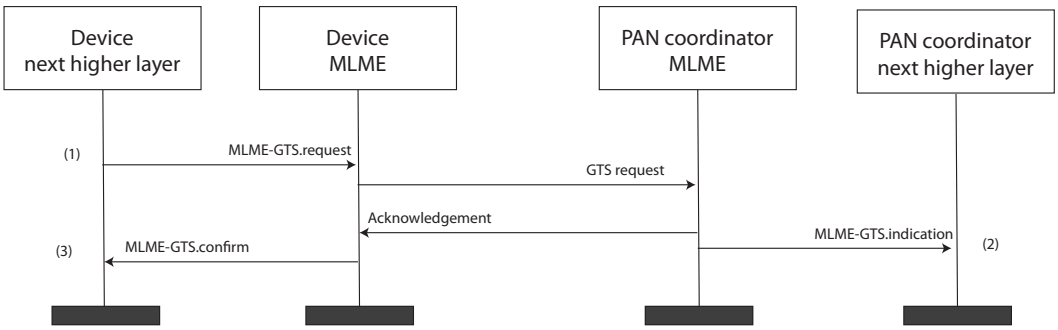


Figure 2.15: MLME-GTS deallocation mechanism

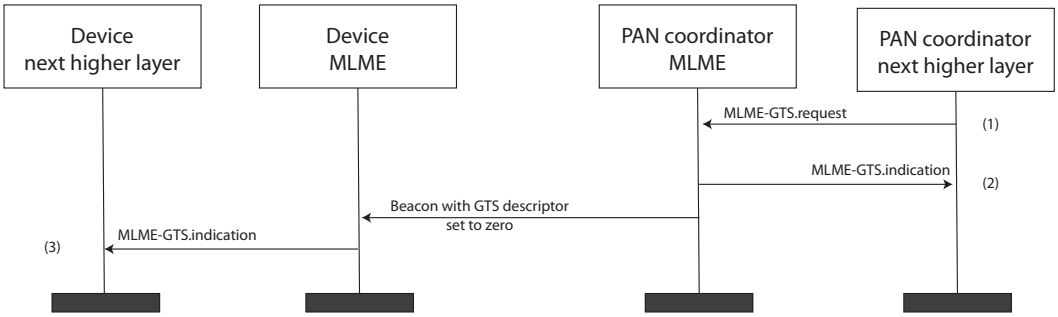


Figure 2.16: MLME-GTS deallocation mechanism initiated by PAN coordinator

request from the next higher layer, the expiration of the GTSs, or maintenance required to maintain the minimum CAP length.

Figure 2.16 shows the GTS deallocation mechanism initiated by the next higher layer of the PAN coordinator. (1) The MLME shall receive the `MLME-GTS.request` primitive with the GTS Characteristics set to zero and the length and direction subfields set according to the characteristics of the GTS to deallocate. (2) The notification is achieved when the MLME issues the `MLME-GTS.indication` primitive.

(3) In case of any deallocation initiated by PAN coordinator, it shall deallocate the GTS and add a GTS descriptor into its beacon frame corresponding to the deallocated GTS, but with its starting slot set to zero.

GTS reallocation The deallocation of a GTS may result in the superframe becoming fragmented. For example, Figure 2.17 shows three stages of a superframe with allocated GTSs. In Figure 2.17a, five GTSs are allocated starting at slots, 15, 13, 12, 11 and 9, respectively. If GTS 2 is now deallocated (Figure 2.17b, there will be a gap in the superframe during which nothing can happen. To solve this, GTS 3 to 5 will have to be shifted to fill the gap, thus increasing the size of the CAP,

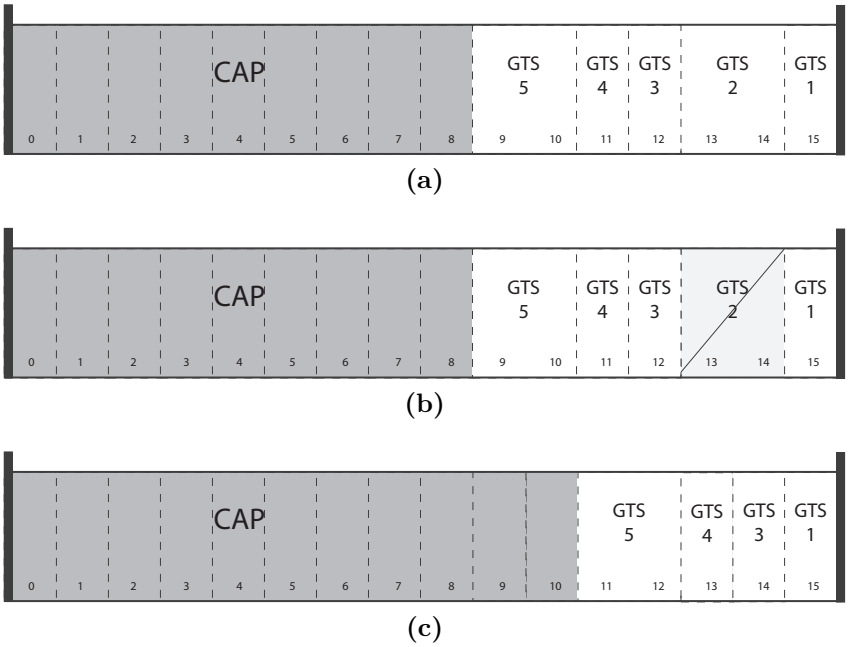


Figure 2.17: CFP defragmentation on GTS reallocations

Figure 2.17c.

GTS expiration The MLME of the PAN coordinator shall attempt to detect when a device has stopped using a GTS using the following rules:

- For a transmit GTS, it shall assume that a device is no longer using its GTS if a data frame is not received from the device in the GTS at least every $2 * n$ superframes, where n is defined below.
- For receive GTSs, it shall assume that a device is no longer using its GTS if an ACK is not received from the device at least every $2 * n$ superframes, where n is defined below. If the data frames sent in the GTS do not require ACK, the MLME of the PAN coordinator will not able to detect whether a device is using its receive GTS.

  The value of n is defined as follows:

$$n = \begin{cases} 2^{8-macBeaconOrder} & 0 \leq macBeaconOrder \leq 8 \\ 1 & 9 \leq macBeaconOrder \leq 14 \end{cases}$$

Chapter 3

Platforms and tools

In this chapter we show the platforms and tools used along this thesis. It intends to answer the questions someone who wants to start with a real WSN implementation need to know. Which platform is the most suitable for our application? Is there any operating system already designed? Which OS is the most suitable? Which is the best implementation for the IEEE 802.15.4?

After these steps, we focus in the platforms we use. For the hardware we select the motes: (a) Tmote Sky (b) Crossbow Telosb and protocol analyser, IEEE 802.15.4/Zigbee protocol analyser. For the software platform we choose the TinyOS and the TKN15.4 implementation.

3.1 Hardware platforms

3.1.1 Motes

Tiny, low-cost and low-power nodes, colloquially referred to as "motes", are the devices deployed in the environment to communicate wirelessly to gather and report information about physical phenomena. There are different types and platforms for being used in WSN. The most common devices are TmoteSky, TelosB, MicaZ [8], iMote2 [6], etc.

3.1.1.1 Tmote Sky

Figure 3.1: Tmote Sky mote without battery extension. Source: [7]

Tmote Sky [7] is an ultra low power wireless module for use in sensor networks, for monitoring applications, and rapid application prototyping. Tmote Sky leverages industry standards like USB and IEEE 802.15.4 to interoperate seamlessly with other devices.

Features

- 250kbps 2.4GHz IEEE 802.15.4 Chipcon Wireless Transceiver [5]
- Interoperability with other IEEE 802.15.4 devices
- 8MHz Texas Instruments MSP430 microcontroller (10k RAM, 48k Flash)
- Integrated ADC, DAC, Supply Voltage Supervisor, and DMA Controller
- Integrated onboard antenna with 50m range indoors / 125m range outdoors
- Integrated Humidity, Temperature, and Light sensors
- Ultra low current consumption
- Fast wakeup from sleep (<6 μs)
- Hardware link-layer encryption and authentication

- Programming and data collection via USB
- 16-pin expansion support and optional SMA antenna connector
- TinyOS support

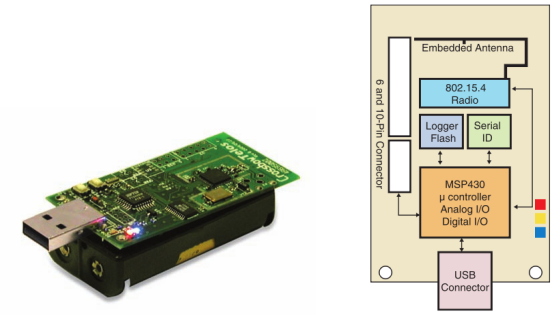
It is important to remark the interoperability they have with the IEEE 802.15.4 devices is coming from the use of the CC2420 radio chip with an external crystal of 16 MHz, which assure that the transmitted frames are standard compliant. But in Section 3.3 we see that it is not true for the MAC implementation sublayer.

3.1.1.2 Crossbow Telosb

Crossbow's TelosB mote is an open source platform designed to enable cutting-edge experimentation for the research community. The telosb bundles all the essentials for lab studies into a single platform including: USB programming capability, an IEEE 802.15.4 radio with integrated antenna, a low-power microcontroller (MCU) with extended memory and an optional sensor suite.

The TelosB platform was developed and published to the research community by UC Berkeley. This platform delivers low power consumption allowing for long battery life as well as fast wakeup from sleep state. Though the telosb is an uncertified radio platform, it is fully compatible with the open-source TinyOS distribution.

Features The Telosb motes have the same hardware than Tmote Sky.



(a) Telosb mote with battery extension

(b) Functional block diagram

Figure 3.2: Telosb mote and functional block diagram Source: [41]

3.1.2 IEEE 802.15.4/Zigbee protocol analyser

To be able to analyse the characteristics of our applications, it is needed to have a device acting in a promiscuous mode and sniffing all the packets which are transmitted through the air.

The selected board is the Development Kit which includes the CC2420B Evaluation Board and a CC2420EM Evaluation Modules [18]. The Evaluation Module contains the CC2420 chip and required external components. The Evaluation Board serves as motherboard for the Evaluation Modules. The Evaluation Board provides a USB port, a serial port, buttons, LEDs, voltage regulator, configuration jumpers and connectors to make it easy to interface the CC2420 with the SmartRF™Studio software.

3.1.2.1 Application tools

With the CC2420DK, it is provided a Windows applications in order to see graphically, and in real-time, the packets. The program is called *SmartRF™Packet Sniffer* [18]. The data coming from this program is saved in a file with PSD extension. The file contains all the packet information, but it is a binary file. In order be able to analyse the information with another program, i.e. Matlab, we use a PHP script which converts the binary file in a ASCII file where the columns are the fields of the packet, length, frame control, source address, destination address, etc.

Figure 3.3 shows the packet format in the PSD file. Note that in our case, the packet format explained in the manual [18] is not the packet format that we have with our packets.

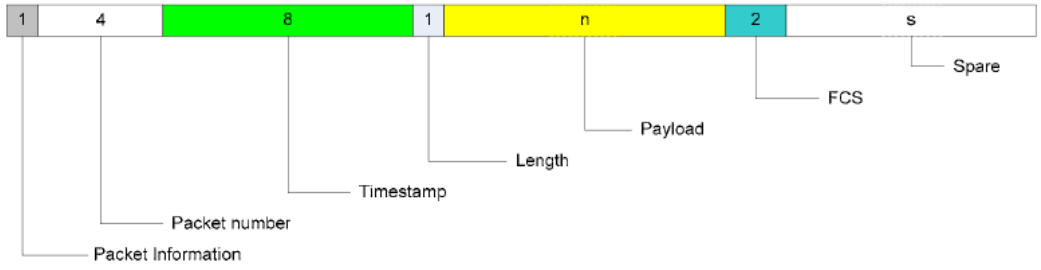
The packet format can vary depending on the addressing mode we use in the packets. For example the TKN15.4 implementation, by default, uses IntraPAN which reduces the MAC Header (MHR) to 11 bytes, by assuming that PAN identification is the same for source and destination. In the openZB Hurray implementation, the IntraPAN is not implemented and they have a MHR of 13 bytes.

Figure 3.4 shows a packet sniffer screenshot from the IEEE 8022.15.4 ZigBee protocol, when we were running a performance test for the GTS implementation.

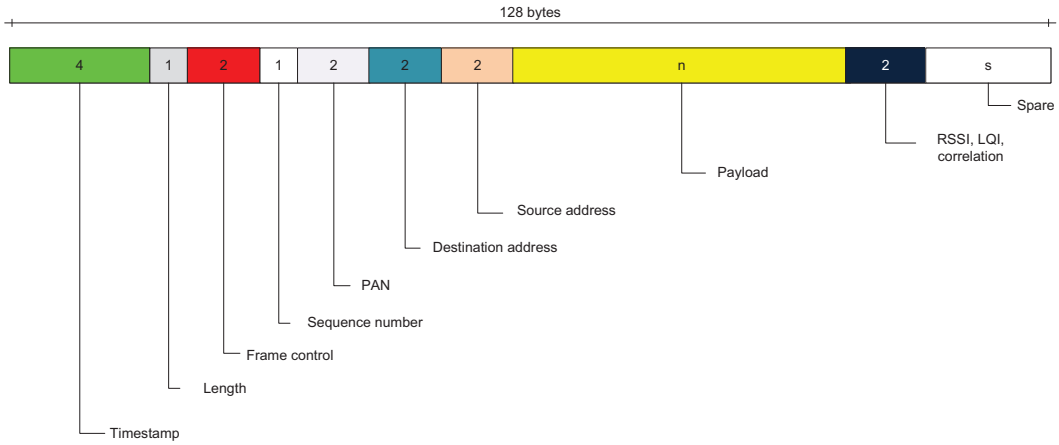
Table 3.1 shows the equivalent parsed PSD file for the packets in the Figure 3.4. Note that all the fields are only valid for data packets.

3.2 Operating systems

The purpose of this section is to show different available OS designed for WSN. Table 3.2 summarizes the present OSs available for the selected motes. It also



(a) Packet format in PSD file Source: [18]



(b) Packet format in PSD file in our case

Figure 3.3: Packet format in PSD file. User manual vs Received packet

shows the last updated date. The two active development projects are TinyOS and Contiki.

We select TinyOS due to the difference in the learning curve in both cases. TinyOS provides a huge documentation and has a large active community, ready to solve any problem. It has applications, tutorials, TEP that make it easy to start working with TinyOS even though it uses a not common programming language (*nesC*). In case of Contiki, it is hard to find tutorials and it lacks documentation and implementation of hardware drivers for the TmoteSky/Telosb.

3.2.1 Contiki

Contiki [11] is an open source, highly portable, multi-tasking operating system for memory-efficient networked embedded systems and WSN. It is designed for microcontrollers with small amounts of memory. A typical Contiki configuration is 2 kB of random access memory (RAM) and 40 kB of read-only memory (ROM).

Contiki is developed by a group of developers from industry and academia lead

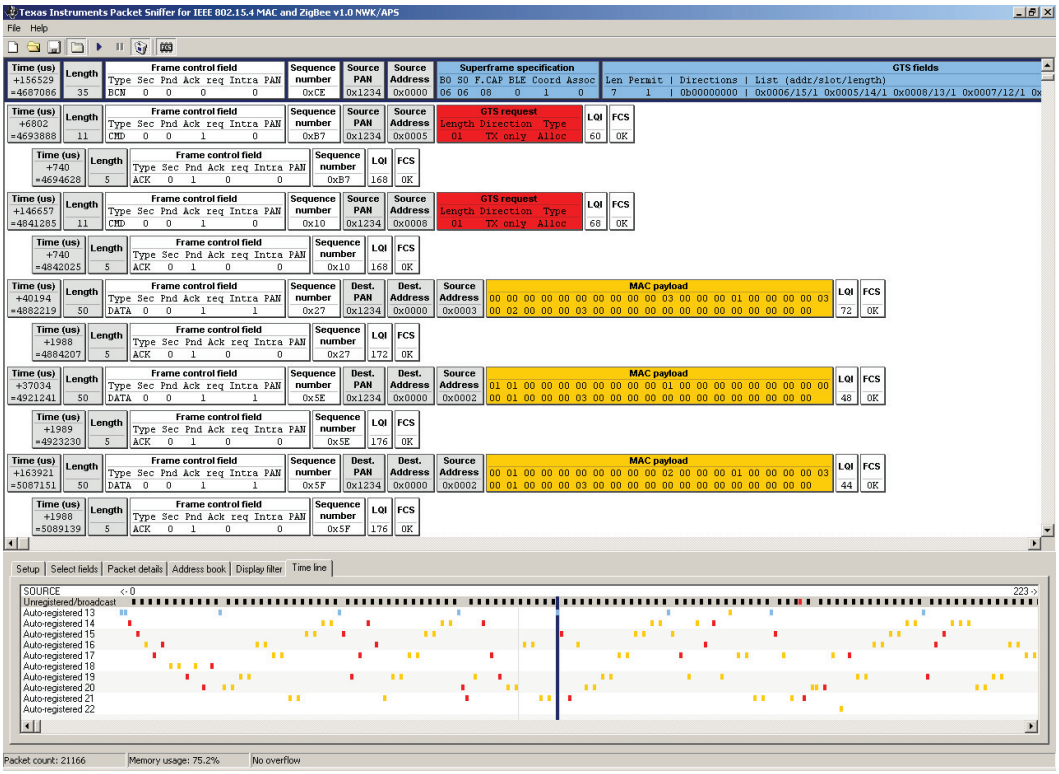


Figure 3.4: Packet sniffer screenshot from the IEEE 802.15.4ZigBee protocols.

by Adam Dunkels from the Swedish Institute of Computer Science (SICS). The Contiki team currently has sixteen members from SICS, SAP AG, Cisco, Atmel, NewAE and TU Münich.

3.2.1.1 Features [11]

Low-power Radio Communication Contiki provides both full IP networking and low-power radio communication mechanisms. For communication within wireless sensor network, Contiki uses the Rime low-power radio networking stack.

Network Interaction Interaction with a network of Contiki sensors can be achieved with a Web browser, a text-based shell interface, or dedicated software that stores and displays collected sensor data. The text-based shell interface is inspired by the Unix command shell but provides special commands for sensor network interaction and sensing.

Power-efficiency To provide a long sensor network lifetime, it is crucial to control and reduce the power consumption of each sensor node. Contiki provides

		Frame format							
Packet type	**Timestamps** [μs]	**Length** [bytes]	**FC**	**Seq. num.**	**PAN ID**	**Dest. ID**	**Src ID**	**RSSI***	**LQI***
Beacon[1]	7659395	35	32768	206	4660	0	18534	237	235
CMD[2]	7666197	11	32803	183	4660	5	8457	237	235
ACK[3]	7666937	5	18	183	59652	5	8457	237	235
CMD[2]	7813594	11	32803	16	4660	8	8457	237	235
ACK[3]	7814334	5	18	16	60164	8	8457	237	235
Data	7854528	50	34913	39	4660	0	3	236	236
ACK[3]	7856516	5	18	39	60165	0	3	236	236
Data	7893550	50	34913	94	4660	0	2	230	235
ACK[3]	7895539	5	18	94	60166	0	2	230	235
Data	8059460	50	34913	95	4660	0	2	229	236
ACK[3]	8061448	5	18	95	60166	0	2	229	236

Table 3.1: Value after parse the PSD file.(1) For beacons, the fields Dest.Id correspond to Src.Id and the Src.Id is the Superframe specification. ; (2) For CMD, the field Dest.Id and Src Id. don't correspond to the real values ; (3) For ACK, as far as ACKs packets don't have any addressing mode, the sniffer copy the values from the previous packet.

OS	**By**	**License**	**Updated**	**Language**	**802.15.4**
Contiki [11]	SICS (Sweden)	BSD	September 2010	C	Partially
TinyOS [43]	UCB, Intel (USA)	BSD	September 2010	nesC	TKN15.4, Hurray, OpenWSN
SOS [23]	UCLA (USA)	Modified BSD	November 2008	C	No
Mantis [4]	CU Boulder (USA)	BSD	2008	C	No

Table 3.2: Main available OS for tmote/telosb

a software-based power profiling mechanism that keeps track of the energy expenditure of each sensor node.

On-node Storage: the Coffee File System Contiki provides a flash-based file system, Coffee, for storing data inside the sensor network.

Simulators To ease software development and debugging, Contiki provides three simulation environments: the MSPsim emulator, the Cooja cross-layer network simulator, and the netsim process-level simulator. The development process for software for Contiki typically goes through all three simulation stages before the software runs on the target hardware.

Programming Model Contiki is written in the C programming language and consists of an event-driven kernel, on top of which application programs can be dynamically loaded and unloaded at run time. Contiki processes use lightweight protothreads that provide a linear, threadlike programming style on top of the event-driven kernel. In addition to protothreads, Contiki also supports per-process optional multithreading and interprocess communication using message passing. Contiki provides three types of memory management: regular malloc(), memory block allocation, and a managed memory allocator.

3.2.2 TinyOS

TinyOS [43] is an open source, BSD-licensed operating system designed for low-power wireless devices, such as those used in sensor networks, ubiquitous computing, PAN, smart buildings, and smart meters. A worldwide community from academia and industry use, develop, and support the operating system as well as its associated tools.

TinyOS has a programming model tailored for event-driven applications as well as a very small footprint. TinyOS is developed in *nesC*, a language for programming structured component-based applications.

3.2.2.1 Features [14, 21]

Component-based architecture Provides a set of reusable system components. A application connects components using a wiring specification, each application customizes the set of components it uses. Decomposing different OS services into separate components allows unused services to be excluded from the application so it reduces the memory requirements.

Task and event-based concurrency Task and events are the two sources of concurrency in TinyOS. Tasks are a deferred computation mechanism, they run to completion and do not preempt each other. [42, TEP 106] [21, Sec 4.4] Events also run to completion, but may preempt the execution of a task or another event.. Events signify either completion of a split-phase operation or an event from the environment.

Split-phase operations Because tasks execute non-preemtively, TinyOS has no blocking operation. All long-latency operation are split-phase: operation request and completion are separate functions. [21, Sec 4.1] Commands are typically requested to execute an operation. A typical example of a split-phase operation is a packet send: a component may invoke the `send` command to initiate the transmission of a radio message, and the communication component signals the `sendDone` event when transmission has completed.

3.2.2.2 Directory structure

Table 3.3 shows the default packages in a TinyOS distribution. [42, TEP 3]:

Folder	Description
`apps/`	Contain applications with some division by purpose. Applications may contain subdirectories. It is not necessary that packages other than the core break up their components and their interfaces.
`tos/system/`	Core TinyOS components. This directory's components are the ones necessary for TinyOS to actually run.
`tos/interfaces/`	Core TinyOS interfaces, including hardware-independent abstractions. Expected to be heavily used not just by tos/system but throughout all other code. tos/interfaces should only contain interfaces named in TEPs.
`tos/platforms/`	Contains code specific to mote platforms, but chip-independent.
`tos/chips/`	Contains code specific to particular chips and to chips on particular platforms.
`tos/libs/`	Contains interfaces and components which extend the usefulness of TinyOS but which are not viewed as essential to its operation. Libraries will likely contain subdirectories.

Table 3.3: TinyOS-2 tree

3.2.2.3 nesC

Programming language *nesC* [14] is an extension to C designed to embody the structuring concepts and execution model of TinyOS. Two of the motivations in designing nesC were to support and evolve TinyOS's programming model and to reimplement TinyOS in the new language. TinyOS has several important features that influenced nesC's design: a component-based architecture, a simple event-based concurrency model, and split-phase operations, shown in sec. 3.2.2.1.

The basics concepts behind *nesC* are [13]:

Construction vs Composition The construction and the composition are separated. Programs are built out of components, which are assembled ("wired") to form whole programs. Components define two scopes, one for their specification (containing the names of their interface instances) and one for their implementation.

Components specification Interfaces may be provided or used by the component. The provided interfaces are intended to represent the functionality that the component provides to its user, the used interfaces represent the functionality the component needs to perform its job.

Interfaces Interfaces are bidirectional, they specify a set of functions to be implemented by the interface's provider (commands) and a set to be implemented by the interface's user (events). This allows a single interface to represent a complex interaction between components.

Components linked Components are statically linked to each other via their interfaces. This increases runtime efficiency, encourages robust design, and allows for better static analysis of program's.

Compiler nesC is designed under the expectation that code will be generated by whole-program compilers. This allows for better code generation and analysis. An example of this is nesC's compile-time data race detector, which detects when a variable could be read/written from two different instruction at the same time.

Concurrency The concurrency model of nesC is based on run-to-completion tasks, and interrupt handlers which may interrupt tasks and each other. The nesC compiler signals the potential data races caused by the interrupt handlers.

3.3 IEEE 802.15.4 software implementation

This section shows the different IEEE 802.15.4 implementations for TelosB/TmoteSky. To select one of the existence we analyse which one accomplish better the standard requirements. Mainly we focused on delay introduced by the code, precision and accuracy. The protocol design process for WSN in industrial application encounters more challenges than the three showed before, like reliability. Hence, in Chapter 5 the performance evaluation of these characteristics is done for the selected implementation, the TKN15.4.

3.3.1 OpenZB - IPP Hurray

This implementation is done and supported by Research Centre in Real-Time Computing Systems (CISTER), a top-ranked Research Unit based at the School of Engineering (ISEP) of the Polytechnic Institute of Porto (IPP), Portugal.

Firstly, the openZB implementation was selected as the main candidate to apply to our project. There were reasons to use it. Mainly they had a good documentation,

a continuous update at the web page [19] and the forum and some recent publications like [10] which re-conform the good feelings with this implementation. Moreover the SICS is working in conjunction with ISEP in the migration to the Contiki OS, and the Scuola Superiore Sant'Anna di Studi Univertari e di Perfezionamento (SSSUP) in the migration to the Erika real-time OS.

In the following section, we can see the missing functionalities of this implementation, because although it was our first selection, it is not completely implemented, and it has some known bugs.

3.3.1.1 Missing functionalities

The next list is the one we could find in the IPP Hurray technical report [9].

- **Unslotted version CSMA-CA** Implemented but not fully tested;
- **Extended Address Fields of the Frames**
- **IntraPAN Address Fields of the Frames**
- **Active and Orphan channel Scan**
- **Orphan Devices**
- **Frame Reception Conditions** Verify Conditions
- **Security**

3.3.1.2 Known Issues

Below, the list shows the found bugs in the openZB implementation of IPP Hurray. This is a summary of the known bugs. It could be the case that more issues are involved, but as soon as we discarded this implementation, we stopped analysing it.

- **Stop working** The main bug detected is that the motes after some time stop working. As an example, a coordinator without any traffic load, stops working after 2100 beacons sent (36 min). Obviously, it is completely impermissible that stops working after 30 min or after days, if it is powered by the power line it has to work forever.
- **Beacon interval** The time between beacons is shifted. We cannot be sure about the periodicity and the interoperability of that protocol implementation and other devices. As we know the beacon is the signal needed to align all the network, and the devices use this to reduce the collisions which mean that if they are not align, the CSMA-CA will not work as it is designed: no interoperability, high delay and low reliability.

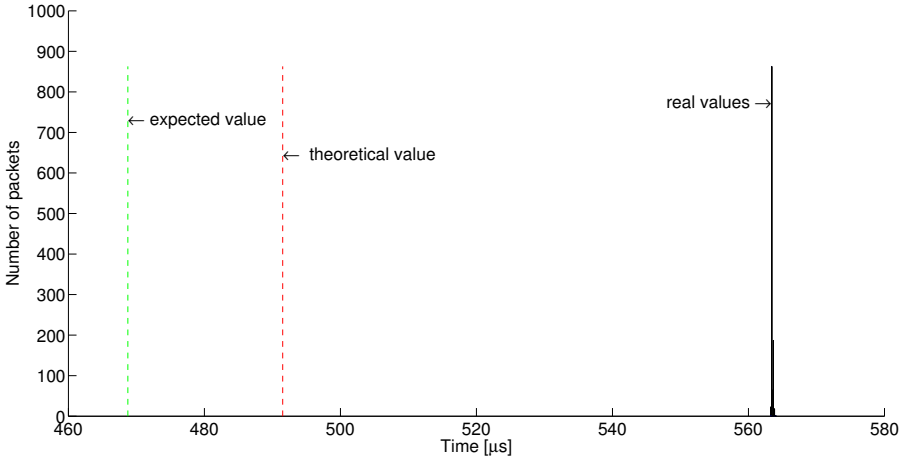


Figure 3.5: Histogram of the beacon time interval for BO=5

Figure 3.5 shows the histogram of time interval between two consecutive beacons for BO=5. The green dashed line on the left represents the expected value taking into account that the T_{symbol} will never be equal to 16 μs because our oscillator runs at 32.768 kHz and the minimum time that we can achieve is $T_{symbol} = \frac{1}{2*32768} = 15.259\mu s$ using a virtualized timer. The red dashed line in the middle represents the theoretical value of the beacon interval. And comparing these values with the histogram, we can see how far we are from the real value (low accuracy). Moreover we can see different bars close, which shows the low precision of the timer.

Table 3.4 shows the beacon interval with the hurray implementation. As we can see these times are not close to the values from $T_{symbol} = 16\mu s$. Moreover we can see how the error *error* column , representing the difference between the theoretical BO and the most occurred value. is increasing with the BO, but the relative error keeps approximately constant. The precision (standard deviation) increases with BO. The conclusion looking at those values is that the beacon interval is not standard compliant and moreover it has not a good accuracy or precision.

- **Acknowledge delay** A device that sends a data or MAC command frame with its ACK Request subfield set to one shall wait for at most *macAckWaitDuration* symbols for the corresponding ACK to be received. It is dependent on a combination of constants and PHY attributes [26, Sec. 7.4.2]

  The corresponding time for our configuration is:

  $macAckWaitDuration = aUnitBackoffPeriod + aTurnaroundTime + phySHRDuration + \lceil 6 * phySymbolsPerOctet \rceil = 20 + 12 + 5*2 + \lceil 6*2 \rceil = 54\ symb =$

BO	**BI** $T_{symbol} = 16\mu s$ [ms]	**BI** openZB impl. [ms]	**Error** e [ms]	e_r [%]	**Precision** Standard deviation [μs]
1	30.7	36.1	5.4	17.6	94.0
2	61.4	71.5	10.0	16.3	107.8
3	122.9	141.6	18.7	15.2	100.9
4	245.8	282.4	36.6	14.9	95.9
5	491.5	563.4	71.9	14.6	104.3
6	983.0	1055.7	72.6	7.4	127.2
7	1966.1	2250.8	284.8	14.5	101.8
8	3932.2	4500.9	568.7	14.5	124.8
9	7864.3	9000.5	1136.1	14.4	168.8
10	15728.6	17999.9	2271.3	14.4	180.2
11	31457.3	35998.9	4541.6	14.4	235.8
12	62914.6	71996.8	9082.2	14.4	341.9
13	125829.1	143992.6	18163.5	14.4	389.0
14	251658.2	287984.7	36326.5	14.4	996.5
			Total	14.4 %	226.3 μs

Table 3.4: Beacon interval on openZB implementation

864 μs.

For this implementation we get the acknowledge within approximately $5ms$ value far away of the timing requirements of *macAckWaitDuration*.

- **ACK rejection** Some ACK are rejecting because the mote was in the middle of the CSMA-CA algorithm of the next transmitted packet. This issue, with the previous one, causes that the transmitted mote re-transmit the previous packet although the receiver has been received it properly. Thus, it increases unnecessarily the traffic load on the network. It could be reasonable in a network with lot of nodes, where the interferences are higher but it also happens when just one node is transmitting to the coordinator.

Although this implementation could be considered complete, involving the main issues of the IEEE 802.15.4 parts. We could not consider it as a useful implementation due to these imperfections and bugs found.

3.3.2 TKN15.4

This implementation is done by the Technical University Berlin - Telecommunication Networks Group, more specifically by Jan-Hinrich Hauer, one of the main

contributor of the TinyOS development. The first release has been done in March 2009 [15].

It is a platform-independent IEEE 802.15.4-2006 MAC implementation. The code is under active development and most of the functionality described in the standard is implemented and tested. The MAC itself is platform-independent, but it requires (1) a suitable radio driver, (2) Alarms/Timers with symbol precision and (3) some "platform glue" code (defining guard times, etc.). Currently the only supported platforms are Tmote/TelosB and micaZ.

Another point to remark about this implementation is the wide knowledge of the platform, possibilities and limitations of the motes. One important thing is discussed in the technical report are the timing issues: precision/accuracy requirements and the clock drift. Because the fact that our motes do not have a clock that satisfies the precision/accuracy requirements of the IEEE 802.15.4 standard – 62.500 kHz, +-40 ppm in the 2.4 GHz band, the timing in beacon-enabled mode is not standard compliant. Instead we have a virtualized clock of $2 * 32.768kHz = 65.536kHz$ that provides a $T_{symbol} = 15.259\mu s$.

Following, we present the missing functionalities and known issues extracted from the technical report [15] or the README in the implementation code.

3.3.2.1 Missing functionalities

- **GTS** The implementation of the GTS has been done in this thesis. A detailed explanation can be found in Section 4.

- **Security services** As far as at the moment the Wireless Process Control applications are in an early development, the security services have not been implemented. The "Home Smart Grids" application shown in Section 6.2 could be interesting to apply a security mechanism in order to protect the data coming from the sensors.

- **Indirect transmissions** frames are not kept in transaction queue in case CSMA-CA algorithm fails

3.3.2.2 Known Issues

The list below corresponds to the known issues that it is necessary to check/solve to have a proper IEEE 802.15.4 implementation. This list correspond to the README file from the TKN15.4 code.

- **Invalid timestamps** If initial beacon transmission timestamps is invalid, the coordinator stops.

- **Frame pending** Frame pending flags are (need to be) always set in the ACK headers

- **Complex networks** Using an incoming and outgoing superframe at the same time has not been tested. A simple star topology has been used.

- **CSMA-CA** During an ongoing CSMA-CA transmission incoming frames are ignored

- **Transmissions modes** On a beacon-enabled PAN: if the device cannot find the beacon the DATA frame is not transmitted (but it should be transmitted using unslotted CSMA-CA [26, Sec. 7.5.6.1]. At the moment they are different components, so at compiling time we differentiate which mode we want to use. As soon as, the motes would have more memory resources (ROM), it would be possible.

The next list corresponds to the results of our test and analysis done to decide if this implementation is good enough to continue the development with it.

- **Missed beacons** When we are running a network with one coordinator and one device, both are completely synchronized and working fine. If we add another device to the network, it starts the application scanning, but once it synchronizes with the coordinator and receives the first beacon, it loses the synchronization and after *aMaxLostBeacons* beacons, it signals the `MLME_SYNC_LOSS`. It means that the packets in the network influences also the reception of the beacon.

- **Beacon interval** The implementation is not completely standard compliant due to hardware limitations. The limitation is coming from our oscillator that runs at 32.768 kHz. We are currently analysing the new MSP430 MCU to use a higher crystal oscillator and overcome this issue.

  Figure 3.6 shows the histogram of time interval between two consecutive beacons for BO=5. The green dashed line on the left represents the expected value taking into account that the $T_{symbol} = 15.259\mu s$. The red dashed line represents the theoretical value of the beacon interval. We see that the histogram is quite tight and close to the expected value.

  Table 3.5 shows the influence of this difference. As we can see on the fifth column (Error). The difference between the value that we get from the mote and the theoretical value is increasing with the BO, but the relative error shown in column six remains constant.

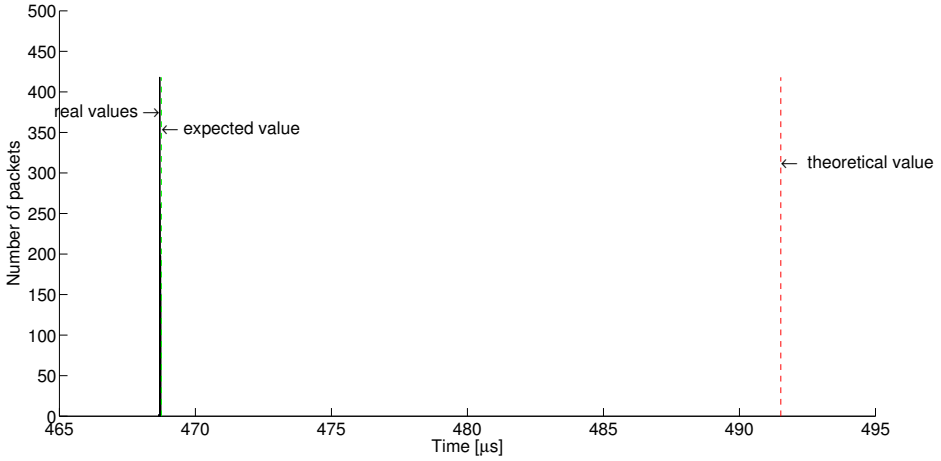


Figure 3.6: Histogram of the beacon time interval for BO=5

3.3.3 TKN15.4 vs OpenZB

We show a comparison between TKN15.4 and openZb IEEE 802.15.4 implementation. Due to the use of the same hardware, we expect similar results. But they are different because of the different ways to use/implement the timers.

For the hurray implementation, a timer based on the 32.768 kHz (30.5 μs) 802.15.4, that provides a $T_{backoff} = 335,5\eta s$, but to launch the different events they used counters of 32 bits, increasing considerably the computational time. [21, Sec. 4.5]. Writing or reading a 32 bits number takes more than one instruction. Then it is possible that an interrupt executes in between two instructions, influencing in the timing related with the MAC layer, as superframe duration fired, beacon interval and backoff fired.

TKN15.4 implements a virtualized timer that runs at 2*32.768 kHz using the external crystal of 32.768 kHz, being able to have a more precision clock. It is possible thanks to the digital clock that is internally is used by the MSP430 MCU.

- **Clock drift:** This limitation exists in both implementations and it is due to a hardware limitation. It is recognized that clock drift can cause contradictions with the timing requirements of the slotted CSMA-CA in beacon-enabled PANs. During the CAP the slotted CSMA-CA algorithm requires frames to be transmitted on backoff slot boundaries. In conjunction with the listen-before-send mechanism, slotted transmissions theoretically guarantee that the time interval during which the arrival of one packet implies a collision with another packet is limited to one backoff slot (320 μs). In practice, however, clock drift can result in collisions between frames even if they are transmitted

BO	BI $T_{symb} = 16\mu s$ [ms]	BI $T_{symb} = 15.26\mu s$ [ms]	BI impl. [ms]	Error |BI(16μs)-BI(impl.)| [ms]	[%]	Precision Standard deviation [μs]
1	30.7	29.3	29.3	1.5	4.7	7.5
2	61.4	58.6	58.6	2.9	4.7	8.4
3	122.9	117.2	117.2	5.7	4.7	9.8
4	245.8	234.4	234.3	11.4	4.6	10.0
5	491.5	468.8	468.7	22.8	4.6	14.4
6	983.0	937.5	937.4	45.6	4.6	13.7
7	1966.1	1875.0	1874.9	91.2	4.6	5.0
8	3932.2	3750.0	3749.8	182.3	4.6	2.6
9	7864.3	7500.0	7499.7	364.6	4.6	7.5
10	15728.6	15000.0	14999.5	729.2	4.6	13.8
11	31457.3	30000.0	29998.2	1459.0	4.6	20.3
12	62914.6	60000.0	59996.4	2918.1	4.6	6.5
13	125829.1	120000.0	119992.9	5836.2	4.6	10.2
14	251658.2	240000.0	239985.8	11672.5	4.6	22.1
				Total	4.6 %	10.8 μs

Table 3.5: Beacon interval on TKN154.4 implementation

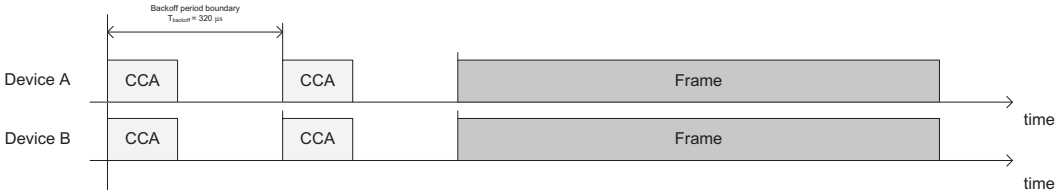
in subsequent backoff slots and thus practical vulnerability period is larger. [15, Sec. 3.2.3.] It is important to take clock drift into consideration for theoretical models because it is a common problem in hardware devices.

Figure 3.7 shows when we detect the collision using devices synchronized. During the CAP of a beacon-enabled PAN, nodes will only collide if they are transmitted in the same backoff slots, because otherwise the CCA mechanism will detect a busy channel. In Figure 3.7a, Device A would collide with the frame from Device B because they are transmitting in the same slot boundaries. However, in Figure 3.7b they would not collide because Device B would detect a busy channel and wait for a random unit of backoff periods.

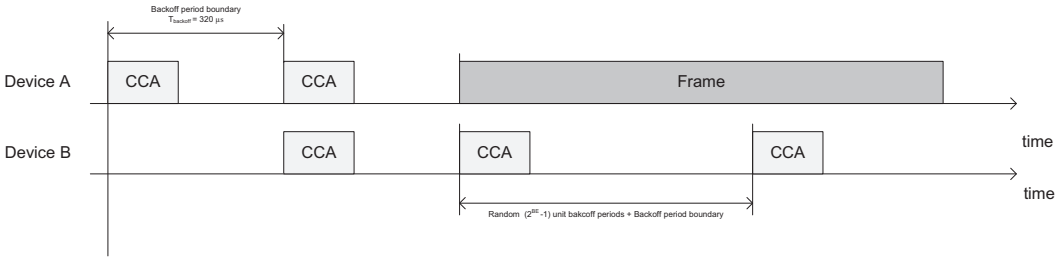
Figure 3.8 shows the case where the devices are not synchronized due to the clock drift. In Figure 3.8a, where they are in the same slot, we do not detect a collision. Moreover, in Figure 3.8b, when they are in different slots and they should detect the collisions, Device A would collide with the frame from Device B because of the clock drift.

- **Beacon interval** As we have shown before, both implementations are far from the expected and theoretical values for the beacon interval. For this reason, we compared both version, to check which one had the best results.

  Table 3.6 shows the real value on the second column, and the values gotten

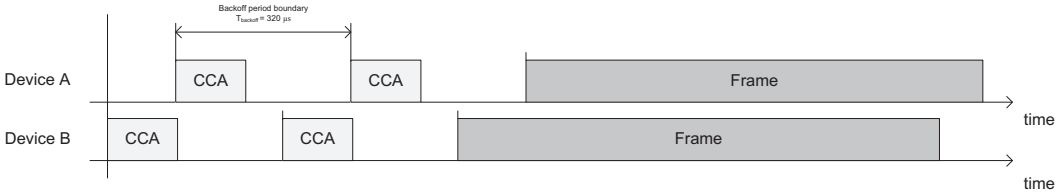


(a) Not collision detection

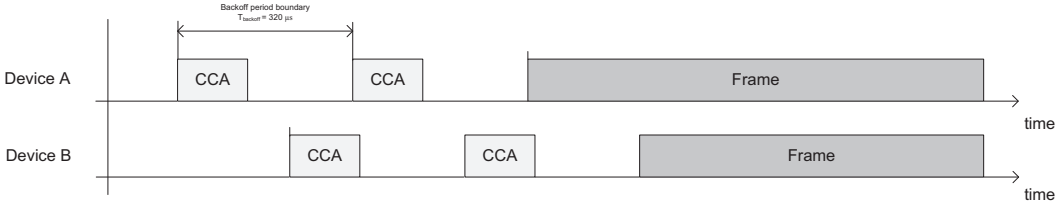


(b) Collision detection

Figure 3.7: Slotted CSMA-CA. Alignment in the slot boundaries with CW=2



(a) Not collision detection



(b) Not collision detection

Figure 3.8: CSMA-CA with misalignment the slot boundaries with CW=2

from the openZB and TKN15.4 implementation as the difference between those values and the expected ones. Even though it is easy to detect which has the best results, on the last row we have the average of the difference for each implementation, which clearly transmit the TKN15.4 has the lowest error.

		openZB impl.		TKN15.4 impl.	
BO	**BI** [ms]	**BI** [ms]	**Difference** [ms]	**BI** [ms]	**Difference** [ms]
1	30.7	36.1	5.4	29.3	1.5
2	61.4	71.5	10.0	58.6	2.9
3	122.9	141.6	18.7	117.2	5.7
4	245.8	282.4	36.6	234.3	11.4
5	491.5	563.4	71.9	468.7	22.8
6	983.0	1055.7	72.6	937.4	45.6
7	1966.1	2250.8	284.8	1874.9	91.2
8	3932.2	4500.9	568.7	3749.8	182.3
9	7864.3	9000.5	1136.1	7499.7	364.6
10	15728.6	17999.9	2271.3	14999.5	729.2
11	31457.3	35998.9	4541.6	29998.2	1459.0
12	62914.6	71996.8	9082.2	59996.4	2918.1
13	125829.1	143992.6	18163.5	119992.9	5836.2
14	251658.2	287984.7	36326.5	239985.8	11672.5
	Relative error:		14.4 %		4.7 %

Table 3.6: Comparison between beacon interval with openZB and TKN15.4 implementation

Chapter 4

GTS implementation

The GTS mechanism provides to the nodes in the network to transmit packets to the coordinator with high reliability and high determinism.

The purpose of this chapter is to provide a reference guide to the GTS implementation described in the IEEE 802.15.4 protocol specification [26] in nesC/TinyOS based on the TKN15.4 implementation [15]. The GTS implementation was developed on and is available for the TelosB [41]/ Tmote Sky [7] platform. This chapter is a complement to [15], providing the overview and the scheme of the actual IEEE 802.15.4 implementation in TinyOS.

4.1 Overview

Before focusing on the more detailed description of the code and the implementation, we present a brief description about the implemented features and their working procedure.

4.1.1 Implementation status

In this section, some screenshots and figures are shown to verify how our implementation is working. It tries to be the equivalence between the theoretical part shown in Section 2.3.2.3 and the practical implementation. Most of the signals and primitives are impossible to show with these screenshots, they only show the packets transmitted and the superframe structure of each case.

The current version of the GTS implementation supports the following IEEE 802.15.4 functionalities: GTS allocation, GTS deallocation, GTS usage, GTS expiration and GTS reallocation.

4.1.1.1 GTS allocation

To allocate a slot in the CFP the device shall send a request to its coordinator and wait for the GTS Descriptor in the beacon payload.

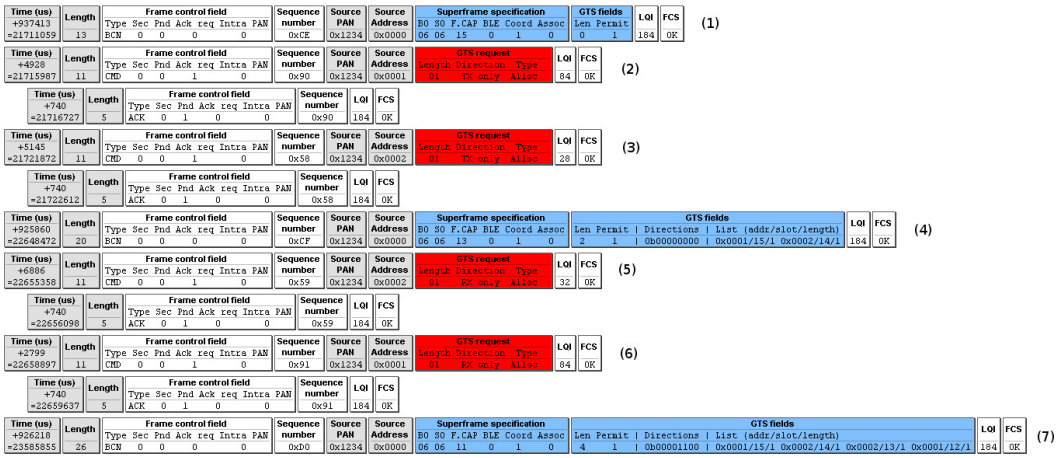


Figure 4.1: GTS allocation mechanism

Figure 4.1 shows the process of GTS allocation and Figure 4.2 shows the state of the superframe. The steps are described here:

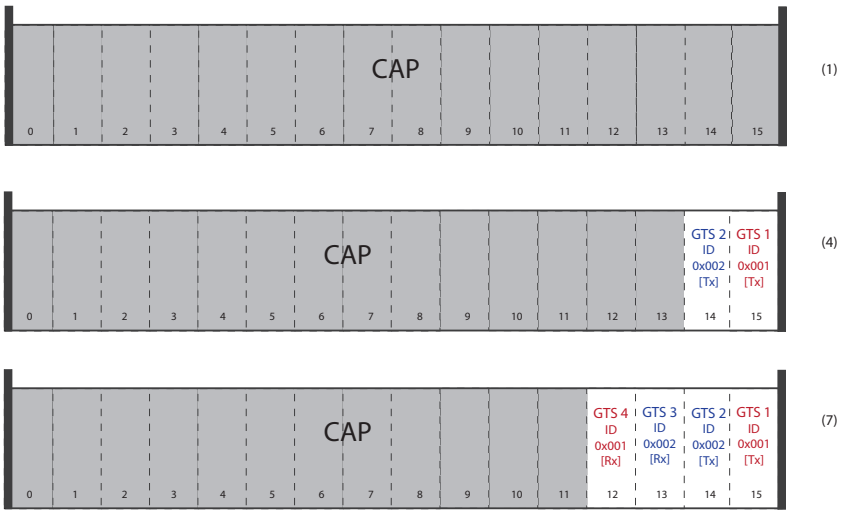


Figure 4.2: Beacon during GTS allocation mechanism

1. In the beacon, the GTS is enabled (Permit = 1) but there is no slot allocated (Len=0)

2. The coordinator, with ID = 0x0000, has received a GTS allocation from the device 0x0001. The request has length = 1, and the direction is for transmission (direction=0). The direction is related to the device.

3. The coordinator has received a GTS allocation from the device 0x0002. The request has length = 1, and the direction is for transmission.

4. In this beacon we have two slots allocated (Len=2). The direction vector are all with 0, indicating slots with transmission direction. In the list we have the description of the slots indicating short address, starting slot and length.

5. The coordinator has received a GTS allocation from the device 0x0002. The request has length = 1, and the direction is for reception (Direction=1).

6. The coordinator has received a GTS allocation from the device 0x0001. The request has length = 1, and the direction is for reception.

7. In this beacon we have four slots allocated (Len=4). The directions vector are two with 0 and two with 1, indicating slots with transmission and reception direction.

4.1.1.2 GTS usage

Figure 4.3 shows a sniffer dump where five motes are running at the same time, and transmitting during the CFP with their own slot. The coordinator is configured to

send packets to the device 0x0001 if the reception slot is requested otherwise it will fail. The devices 0x0001, 0x0003, 0x0004, 0x0005, 0x0006 send allocation request for both directions, but if they do not use it, then it will be deallocated by the expiration mechanism.

All the devices and the coordinator write the slot number in the packet payload in order to assure that they are sending the packet in the correct slot.

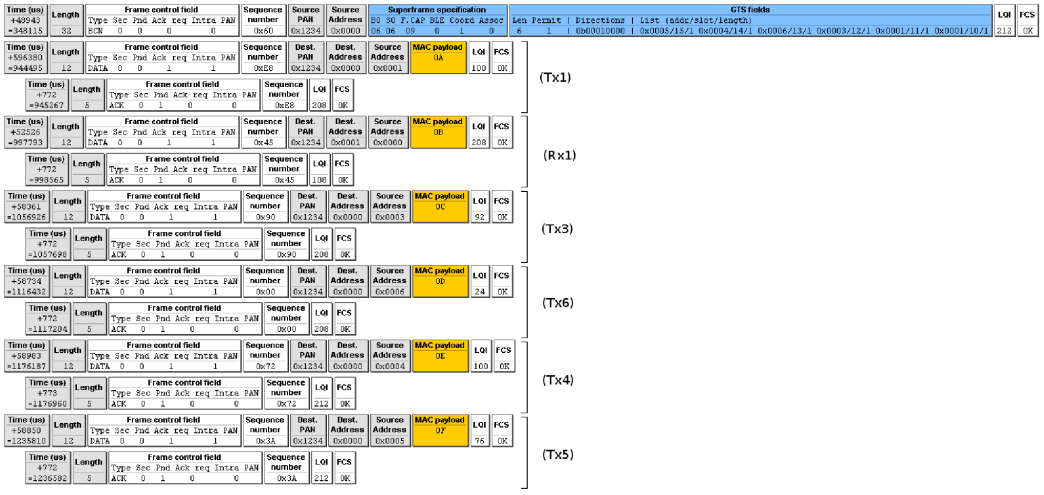


Figure 4.3: GTS data transmission

4.1.1.3 GTS deallocation

To deallocate a slot, there are two ways. If the device starts the deallocation it shall send a request to the coordinator indicating the deallocation. Otherwise the GTS deallocation may be initiated by the PAN coordinator due to a deallocation request from the next higher layer.

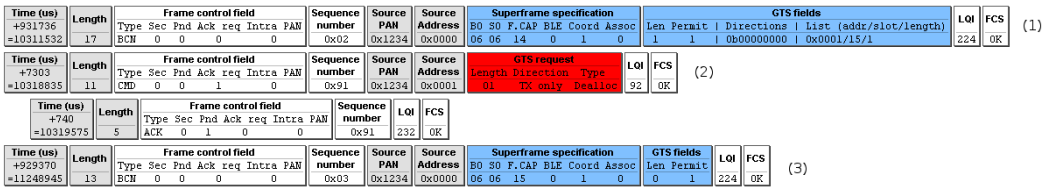


Figure 4.4: GTS deallocation mechanism

Figure 4.4 shows the process of GTS deallocation and Figure 4.5 shows the state of the superframe after receiving the actual packet. The steps are described:

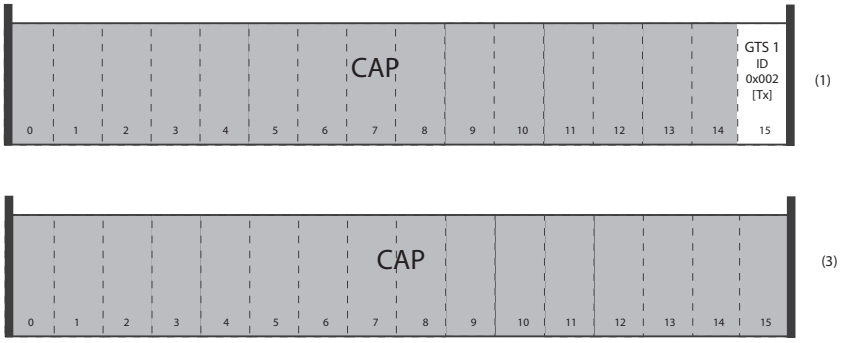


Figure 4.5: Beacon during GTS deallocation mechanism

1. In the beacon we have one slot allocated (Len=1) for the device 0x0001. See state in Figure 4.5.
2. The coordinator has received a GTS deallocation from the device 0x0001.
3. In the beacon, we don't have any slot allocated.

4.1.1.4 GTS reallocation

When a slot has been deallocated, it could do a gap in the superframe as Figure 4.7. To solve this, the slots will realign to avoid the existence of those gaps.

Time (us)	Length	Frame control field	Sequence number	Source PAN	Source Address	Superframe specification	GTS fields	LQI	FCS
		Type Sec Pnd Ack req Intra PAN				BO SO F.CAP BLE Coord Assoc	Len Permit | Directions | List (addr/slot/length)		
+14998599	35	BCN 0 0 0 0 0	0xFB	0x1234	0x0000	06 06 08 0 1 0	7 1 | 0b00101100 | 0x0001/15/1 0x0002/14/1 0x0002/13/1 0x0001/12/1 0x0003/11/1 0x0003/10/1 0x0004/9/1	192	OK
Time (us)	Length	Frame control field	Sequence number	Source PAN	Source Address	Superframe specification	GTS fields	LQI	FCS
		Type Sec Pnd Ack req Intra PAN				BO SO F.CAP BLE Coord Assoc	Len Permit | Directions | List (addr/slot/length)		
+15936012	32	BCN 0 0 0 0 0	0xFC	0x1234	0x0000	06 06 09 0 1 0	6 1 | 0b00010100 | 0x0001/15/1 0x0002/14/1 0x0001/13/1 0x0003/12/1 0x0003/11/1 0x0004/10/1	192	OK
Time (us)	Length	Frame control field	Sequence number	Source PAN	Source Address	Superframe specification	GTS fields	LQI	FCS
		Type Sec Pnd Ack req Intra PAN				BO SO F.CAP BLE Coord Assoc	Len Permit | Directions | List (addr/slot/length)		
+16873427	29	BCN 0 0 0 0 0	0xFD	0x1234	0x0000	06 06 10 0 1 0	5 1 | 0b00000100 | 0x0001/15/1 0x0002/14/1 0x0001/13/1 0x0003/12/1 0x0004/11/1	192	OK

Figure 4.6: GTS reallocation mechanism

To illustrate this mechanism we have deleted the data packets from a sniffer dump because we just want to focus on what happens with the slots and how the coordinator re-organize them in the superframe.

Figure 4.6 shows the beacons from the sniffer, and Figure 4.7 shows the evolution of the superframe structure. The steps are:

1. All the slots are allocated for transmission or reception.
2. The slot for reception of the device with id=0x0002, slot 13, has been reallocated. Slots 9, 10, 11, 12 have been shifted.
3. The slot for reception of the device with id=0x0003, slot 11,has been reallocated. Slot 10 has been shifted.

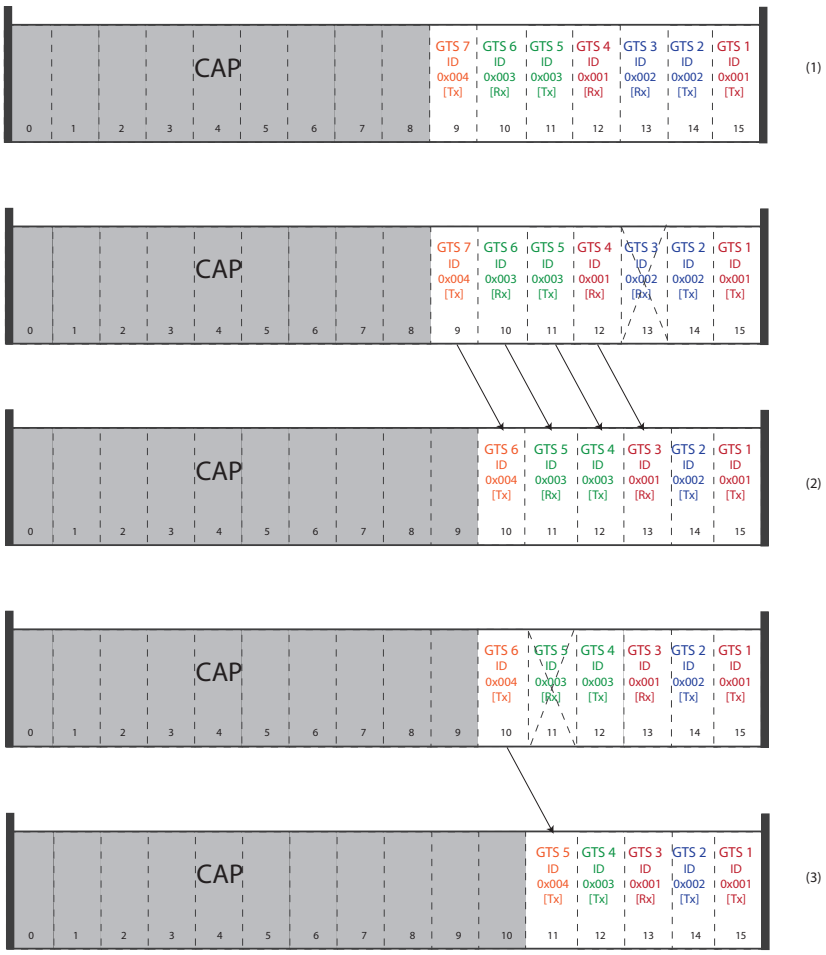


Figure 4.7: Beacon during GTS reallocation mechanism

4.1.1.5 GTS expiration

Figure 4.8 shows a deallocation mechanism due to an expiration. The steps are described:

1. In the beacon, we see that the device 0x0002 has two slots allocated, one for reception and one for transmission.

2. The device 0x0002 is sending packets to the coordinator 0x0000. This pattern has been repeated for $n = 2 * 2^{8-macBeaconOrder} = 8$ superframes (macBeaconOrder = 6). After that, the coordinator detects that the slot allocated for reception has not been used for $n = 8$ superframes.

3. The reception slot has been deallocated by the coordinator.

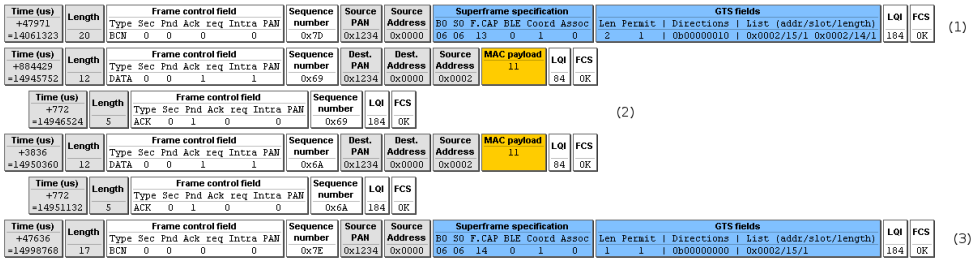


Figure 4.8: GTS expiration mechanism

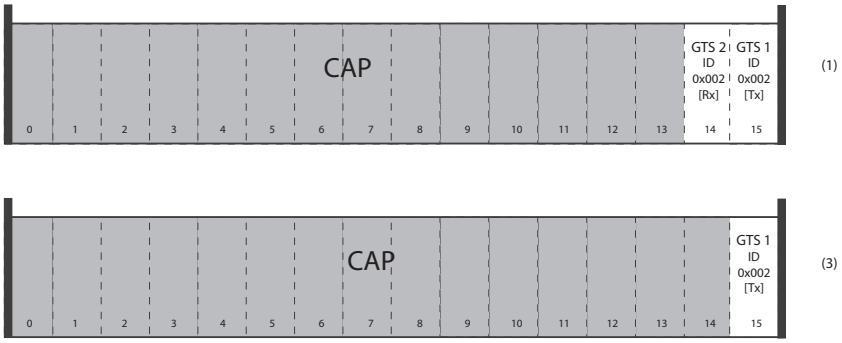


Figure 4.9: Beacon during GTS expiration mechanism

4.1.2 Interoperability

The interoperability between devices, is one of the most important characteristics that a device needs. It has to be standard compliant to be able to use it with different devices everywhere. Since approximately 1950s, associations, companies and organizations, like IEEE, European Telecommunication Standards Institute (ETSI), International Organization for Standardization (ISO),etc. have worked to achieve that.

One of the utilities that we use to analyse and debug our implementation is the IEEE 802.15.4/Zigbee protocol analyser from Texas Instruments as we explained in Section 3.1.2 but although this board only sniffs IEEE 802.15.4 packets, it does not mean the implementation is fully standard compliant.

Another important test is the interoperability between motes with the TKN15.4 implementation and other IEEE 802.15.4 standard compliant devices. They shall be able to communicate each other without problems because the previous TKN15.4 and the GTS addition are done with a very strict and careful standard document tracking.

4.1.3 Directory Structure

Within the TinyOS 2 module the TKN15.4 MAC implementation is located basically in the `tinyos-2.x/tos/lib/mac/tkn154` directory. The directory includes additional subdirectories for the interface definitions and the placeholder components. Moreover, the location of platform specific code is in another directory. Table 4.1 shows an overview of all the directories involved.

Content	Directory in the TinyOS-2 Tree
TKN15.4 MAC Implementation	
components	`tinyos-2.x/tos/lib/mac/tkn154`
interfaces	`tinyos-2.x/tos/lib/mac/tkn154/interfaces`
placeholder components	`tinyos-2.x/tos/lib/mac/tkn154/dummies`
test applications	`tinyos-2.x/apps/test/tkn154`
Platform specific code for TelosB	
configuration files	`tinyos-2.x/tos/platforms/telosb/mac/tkn154`
modified timer subsystem	`tinyos-2.x/tos/platforms/telosb/mac/tkn154`
CC2420 radio driver	`tinyos-2.x/tos/chips/cc2420_tkn154`

Table 4.1: TKN15.4 directories in the TinyOS-2 tree [15]

As we can see, the directory structure is equal to default TKN15.4. The GTS implementation is placed where the rest of the components, interfaces and dummies are.

4.2 GTS Decomposition

In this section the specific components of the GTS implementation are described. First of all an architecture overview of TKN15.4 is shown. After that, a reference model, radio arbitration and timing issues are shown as well. Then, the GTS components are described. Finally the GTS mechanism shown in Section 2.3.2.3 is verified with our implementation.

4.2.1 Reference model

Figure 4.10 shows an architectural overview of TKN15.4, its main components and the interfaces that are used to exchange MAC frames between components. While this figure abstracts from the majority of interfaces and some configuration components, it illustrates one important aspect, namely, how access to the platform specific radio driver (PHY) in the MAC components.

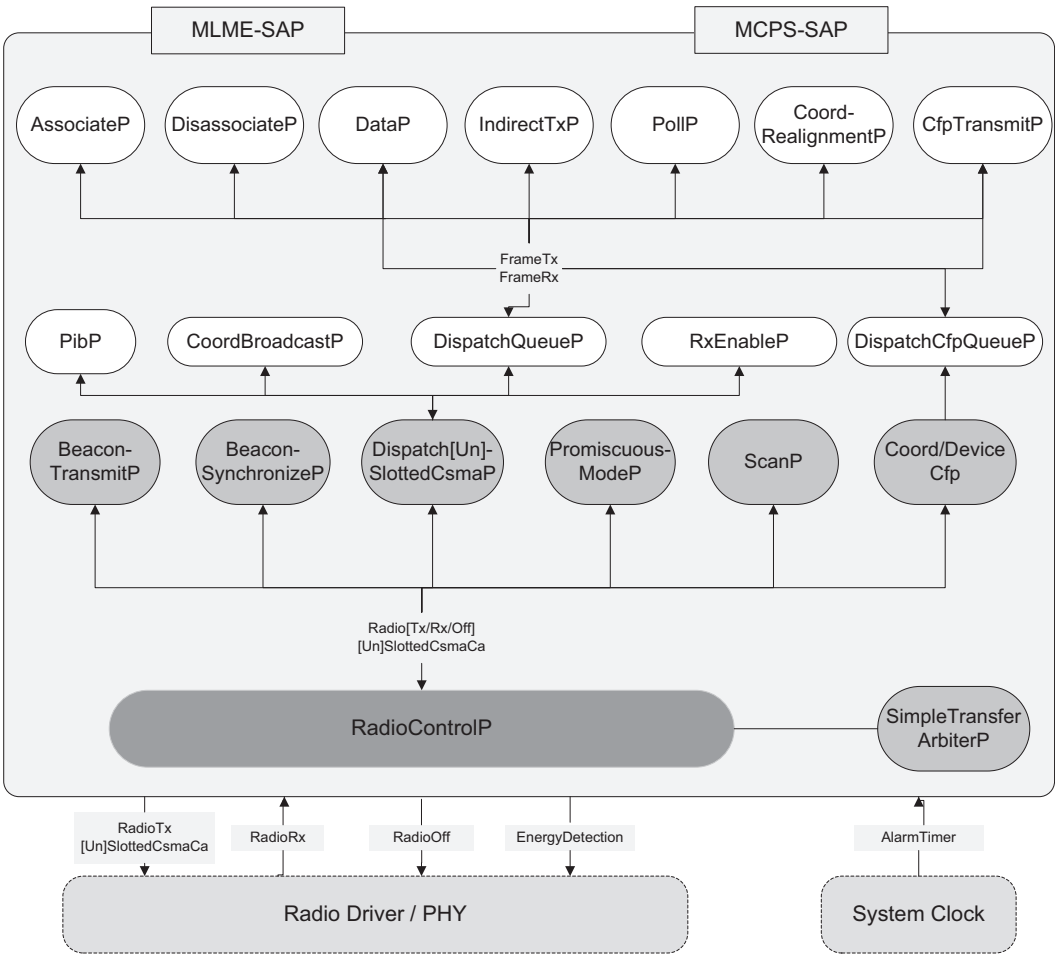


Figure 4.10: TKN15.4 architecture: components are represented by rounded boxes, interfaces by connection lines. The radio driver and symbol clock components are external to TKN15.4

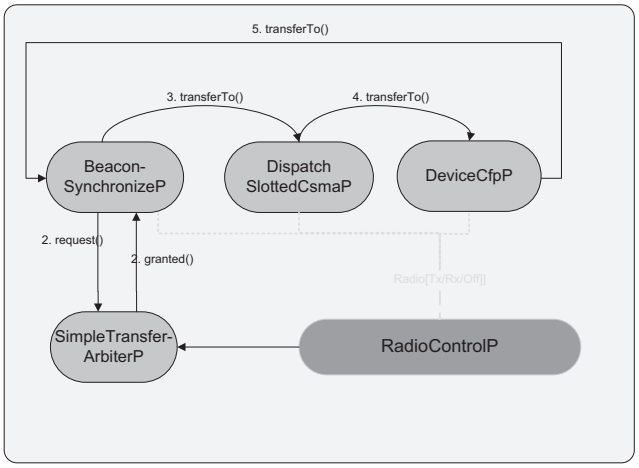
On the lowest level (dark gray boxes), the `RadioControlP` component manages the access to the radio: with the help of an extended TinyOS 2.x arbiter component it controls which of the components on the level above is allowed to access the radio at what point in time.

4.2.2 Radio Arbitration

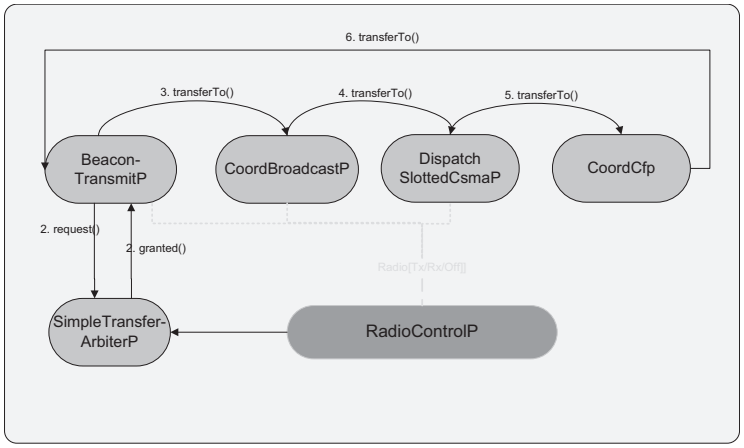
The radio arbitration mechanism is used to coordinate the activities of the components on this level so that they do not overlap in time: typically a component is active only while it has exclusive access to the radio resource. It then performs a certain task and afterwards either releases the resource or passes it on to some other

component. Figure 4.11 explain how the radio resource is transferred and which components can manage it.

The use of a TinyOS resource arbiter avoids inconsistencies in the radio driver state machine and is in line with the standard TinyOS 2.x resource usage model: before a component may access a resource it must first issue a request; once it is signalled the `granted()` event by the arbiter the component can use the resource exclusively; and after usage, the resource must be released. TKN15.4 extends this model by allowing a component that owns the radio resource to dynamically transfer the ownership to a specific other component.



(a) Device



(b) Coordinator

Figure 4.11: Transferring the radio token between the components responsible for an incoming/outgoing superframe. The commands `request()`, `transferTo()` and the `granted()` event are part of the `TransferableResource` interface.

4.2.3 Timing issues

On this section, we analysed some timing issues and problems that the GTS implementation involves. Before focusing on the different issues, we have to keep in mind the environment and timing where we are going to run the GTS.

Considering Table 3.5, we know that the TKN15.4 implementation uses $T_{symbol} = 15.259\mu s$ instead of $T_{symbol} = 16\mu s$, so the time slot duration varies from the standard. Table 4.2 shows the values. From now on, when we talk about the implementation we refer to $T_{symbol} = 15.259\mu s$ and the conversion between symbol units and time is:

$$\begin{aligned} Duration[\mu s] &= Duration[symbols] * T_{symbol} \\ &= Duration[symbols] * 15.259 \frac{\mu s}{symbols} \end{aligned}$$

	Theoretical		Implementation	
BO [symbol]	**BI** [symbol]	**Time Slot** [symbol]	**BI** [ms]	**Time Slot** [symbol]
1	1920.0	120.0	1829.1	114.3
2	3840.0	240.0	3660.1	228.8
3	7680.0	480.0	7322.1	457.6
4	15360.0	960.0	14646.0	915.4
5	30720.0	1920.0	29293.3	1830.8
6	61440.0	3840.0	58589.8	3661.9
7	122880.0	7680.0	117181.6	7323.9
8	245760.0	15360.0	234365.2	14647.8
9	491520.0	30720.0	468732.4	29295.8
10	983040.0	61440.0	937465.9	58591.6
11	1966080.0	122880.0	1874889.8	117180.6
12	3932160.0	245760.0	3749778.1	234361.1
13	7864320.0	491520.0	7499557.0	468722.3
14	15728640.0	983040.0	14999111.0	937444.4

Table 4.2: Beacon interval and Time slot duration on TKN154.4 implementation

4.2.3.1 Timers and Alarms

Figure 4.12 shows the timers and alarm involved in the GTS implementation and needed for the smooth running.

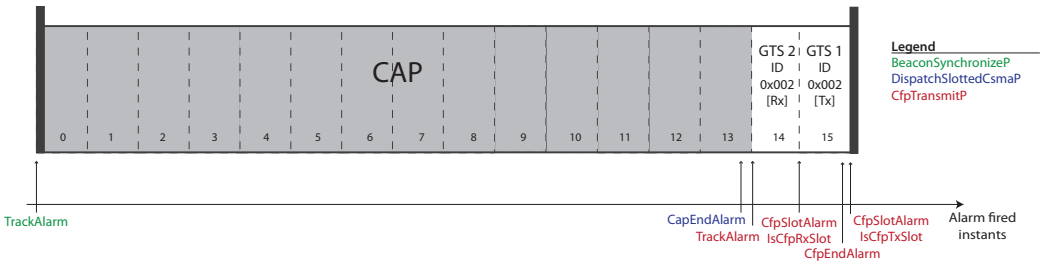


Figure 4.12: Timers and alarm involved in the GTS implementation

Figure 4.13 shows the instant of alarm expiration. The delay influences in the time slot duration, and moreover this delay could not be negligible when we use low SO and BO.

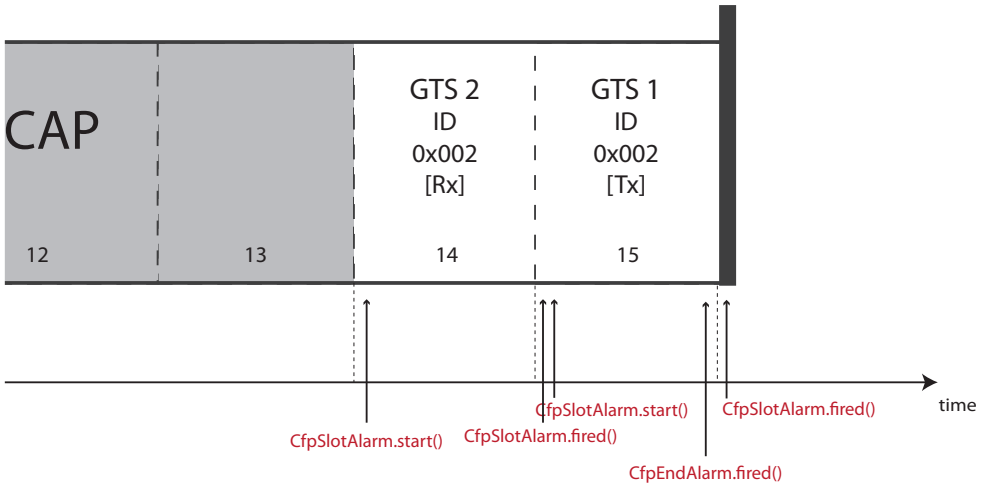


Figure 4.13: Detail of alarms involved in the GTS implementation

4.2.3.2 Time slot duration

It is known that is very difficult to check and debug timing issues without a hardware emulation. For this reason we try to evaluate it, with two methods and obtain conclusions from both.

The first test has been done analysing the sniffer dump. Once enough data is logged with the sniffer, we proceed to convert it to be able to use the data with Matlab. After that, we apply a script to obtain the desired plots and statistics.

The Matlab script computes the difference between data packets and the beacon, preventing to count two packets with the same source id. Some points to be known are:

- The periodicity of the packets is equal to the half of the beacon interval, to be sure that we have packets in every superframe, otherwise if the periodicity is greater than the beacon interval, we have some superframes without any data packet.
- The time between the first packet of each slot is not the same due to the periodicity, so although we obtain some results, we cannot assume that is the time slot duration.

Figure 4.14 shows the histogram of the time slot duration with sniffer method. There are two distinct peaks, one is coming from the slot in the middle of the superframe and the small one, is coming from the last slot, because it does not include the guard time.

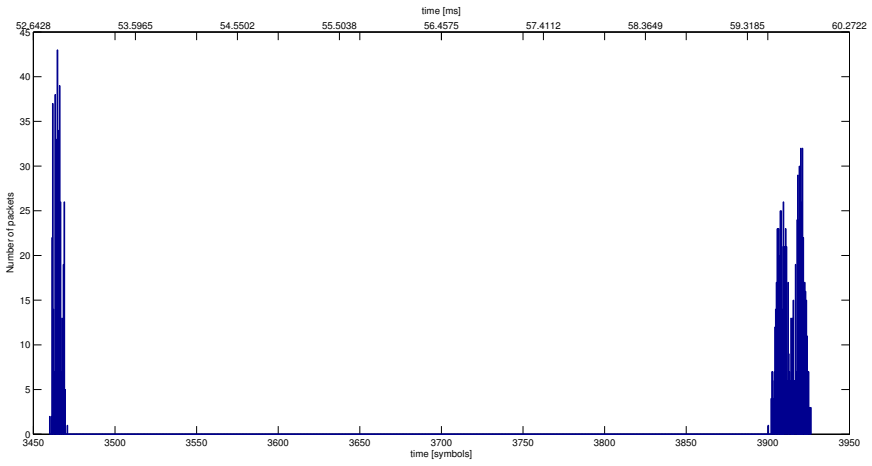


Figure 4.14: Histogram with sniffer method, BO=SO=6 $timeSlotDuration = aBaseSlotDuration * 2^SO = 60 * 2^6 = 3840$

On the other hand, the second test is reached with `printf` functions. To compensate the delay that is introduced by the `printf` instruction. We calculate the difference between the mean using the sniffer method with and without `printf`. The conclusion is that the `printf` introduces a delay around 300 symbols, so, the result should be subtract these number of symbols.

Figure 4.15 shows the histogram of that time slot duration with `printf` method. Here we can see two different peaks, but the difference between them, in that case,

is bigger than in the one with the sniffer method. It is due to the `printf` position in the code. For the slot in the middle the `printf` is placed when a time slot duration is achieved, and for the last one, the shortest one, the `printf` is placed when the token is going to be transferred.

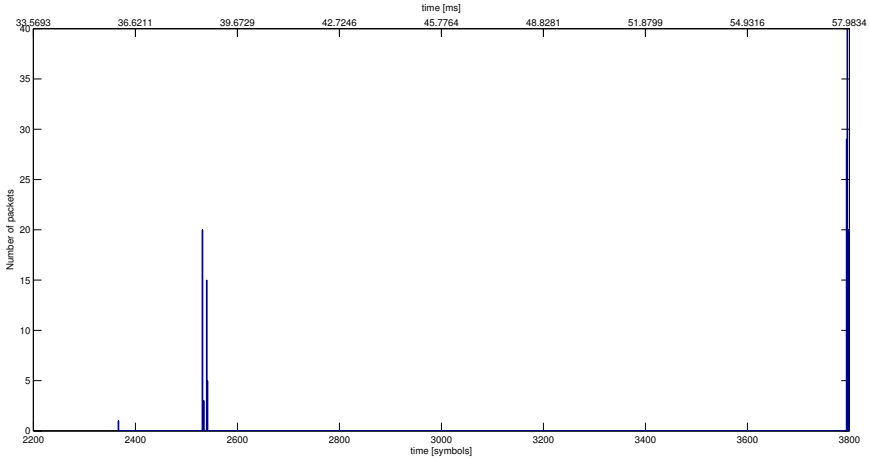


Figure 4.15: Histogram with `printf` method, BO=SO=6 $timeSlotDuration = aBaseSlotDuration * 2^SO = 60 * 2^6 = 3840$

We see that, for our case of study $BO = SO = 6$, the time slot duration from both test correspond with the expected value, considering the limitations. But these results can not be extrapolated to all the range of BO, SO available. Comments and possible improvement are shown in Section 4.3.

4.2.4 Components

The GTS functionality is implemented trying to follow the same structure and the same organization of the TKN15.4 in the MAC layer. Table 4.3 shows the components that provides the GTS functionality.

Component Name	**Function[Provided IEEE 802.15.4 Interface]**
CfpTransmitP	managing GTS transmission [MLME-GTS]
`DeviceCfpP`	functions specific for a device
`DispatchCfpQueueP`	frame transmission during CFP
`CoordCfp`	functions specific for a coordinator

Table 4.3: GTS components

In the `DeviceCfp` and `CoordCfp`, initially it was supposed to contain every function and variable related to the device or coordinator, in order to have device and coordinator in different files. But as soon as the code was growing, most of the code was placed in the `CfpTransmit` due to the necessity of sharing some variables and functions which are in the file.

In the `CfpTransmitP` component, we have some pieces of code with conditional inclusions (*preprocessor directives*) to be able to differentiate device and coordinator without the use of `nesC` conditional sentences, and reduce the size of the program.

The interconnection between components, called *wiring* in `nesC`, are very important to know which components are related to each other. In our case, due to the modularity of the TKN15.4 implementation, it is impossible to show the wiring graph generated by `nesdoc`. For this reason, Figure 4.16 and Figure 4.17 show a graph just with the wiring related to our implementation, even if the components is in the graph it is possible that not all the wiring are shown. [1]

4.2.4.1 DeviceCfp

Uses

`interface MLME_SYNC_LOSS;` Interface that implements the actions in case the device loses synchronization with the PAN coordinator. When it occurs all its GTS allocations shall be lost [26, Sec. 7.5.7 p. 192], and if the device wants to transmit/receive during the CFP again, it has to request a new slot.

`interface GtsUtility;` See Section 4.2.5.1.

Provides

`interface Init;` The basic synchronous initialization interface.

`interface Get<ieee154_GTSentry_t*> as GetGtsDeviceDb;` Interface that is provided to share the device database, with the data of transmit/receive slot. The pointer is shared and the rest of the components that use the interface will be able to modify that variable.

Variables

[1] To generate the complete graph: 1. compile an app with GTS enabled, `make tmote docs`; 2. open `se.kth.tinyos2x.mac.tkn154/docs/nesdoc/index.html`; 3. navigate to the component `TKN154BeaconEnabledP`

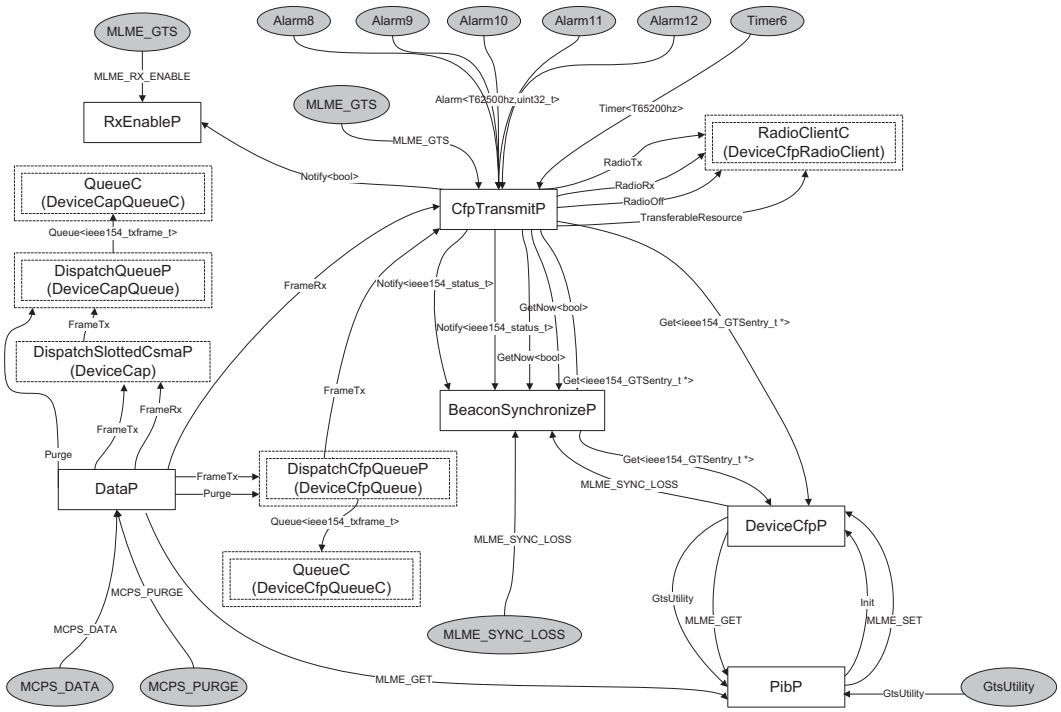


Figure 4.16: Wiring of the components related to the GTS implementation compiling for a device

`ieee154_GTSentry_t GTSentry[2]` Variable that stores the database for the device. There is the configuration for the transmission and reception slots (starting slot, length and direction). As most of the functions are placed in the `CfpTransmitP` components, that variable is shared by the interface `Get<ieee154_GTSentry_t*>` and available for all the components that uses that interface.

4.2.4.2 CoordCfp

Uses

`interface MLME-GET;` Interface that is used to get the configuration variables from the PAN Information Database (PIB). Use: `call MLME_GET . variableName()`.

`interface MLME-SET;` Interface that is used to set the configuration variables to the PIB. Use: `call MLME_SET . variableName()`.

`interface IEEE154Frame as Frame;` Interface that allows access to the content of a IEEE 802.15.4 frame. It is not the same as FrameUtility.

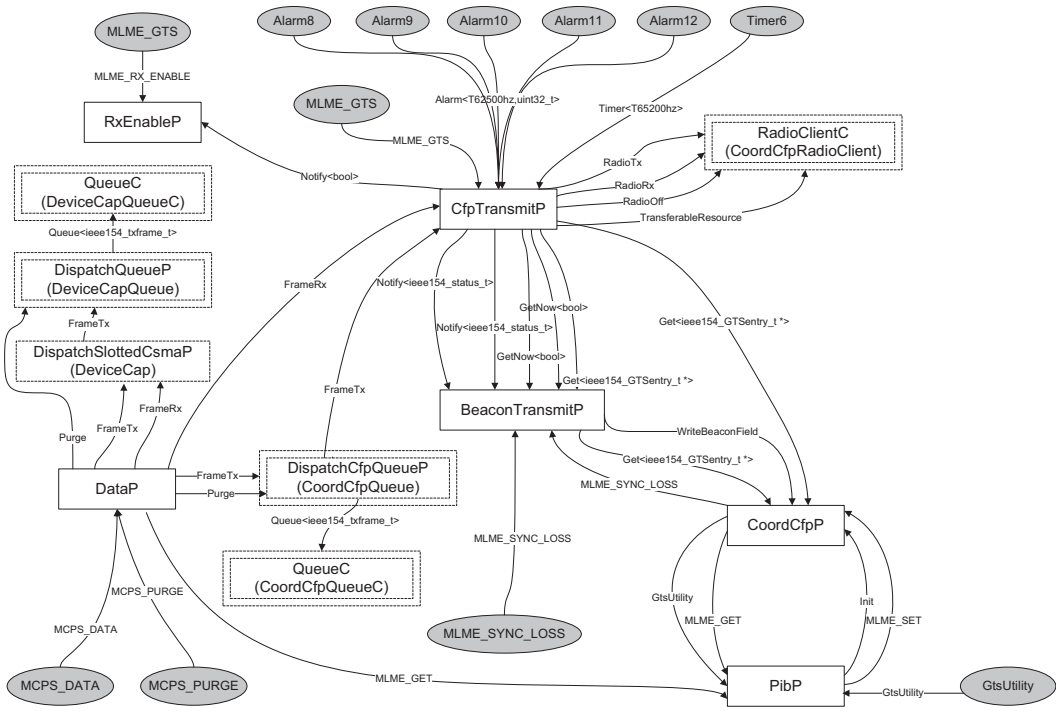


Figure 4.17: Wiring of the components related to the GTS implementation compiling for a coordinator

`interface IEEE154BeaconFrame as BeaconFrame;` Interface that allows to access the content of a beacon frame.

`interface GtsUtility;` See Section 4.2.5.1.

Provides

`interface Init;` The basic synchronous initialization interface.

`interface WriteBeaconField as GtsInfoWrite;` Interface that is used by the `BeaconTransmitP` to write the GTS descriptor in the beacon, in case it was needed. In that interface we analyse all the database that the coordinator has looking for a GTS slot. It is done, just once if the GTS slots don't change.

`interface Get<ieee154_GTSdb_t*> as GetGtsCoordinatorDb;` Interface that provides to share the coordinator database, with the data of all the slots. The pointer is shared and the rest of the components that use that interface will be able to modify that variable.

Variables

`ieee154_GTSentry_t db[CFP_NUMBER_SLOTS];` This is the vector where all the slots informations are store. The number of maximum slots is set to 7, as the standard says [26, Sec. 7.2.2.1.5 p. 145]. The `ieee154_GTSentry_t` contains the following information: starting slot, length, direction, associate device address (short address) and a field indicating the expiration.

`ieee154_GTSdb_t GTSdb;` This is the database shared with the `GetGtsCoordinatorDb` interface. It contains the previous vector of each `ieee154_GTSentry_t` and the number of actual GTS slots that has been used.

`ieee154_macGTSPermit_t m_gtsPermit;` Variable that stores the boolean indicating the possibility to use or not use the GTS.

4.2.4.3 CfpTransmitP

Uses

`interface Leds;` Interface that provides commands for controlling three LEDs.

`interface TransferableResource as RadioToken;` As it has been shown in Section 4.2.2 we need to share the radio resource with all the components to use it properly.

`interface GetNow<token_requested_t> as IsRadioTokenRequested;` Interface that checks if any component has request the `RadioToken`, and immediately it will be released. In a normal operation, the `RadioToken` is not request/release but in some cases like SCAN or RESET is needed.

`interface Alarm<TSymbolIEEE802154,uint32_t> as CfpSlotAlarm;` This is the alarm that triggers every instant one slot has started. When it fires we check in which slot we are at the moment, and if we have to change the radio state to transmission, reception or inactive. Figure 4.12 and Figure 4.13

`interface Alarm<TSymbolIEEE802154,uint32_t> as CfpEndAlarm;` This alarm indicates that the CFP has finished and the token has to be transfer to the `BeaconTransmitP` or `BeaconSynchronizeP` to continue in the inactive period or transmit, receive the beacon. Figure 4.12

`interface Alarm<TSymbolIEEE802154,uint32_t> as IsCfpRxSlot;` When this alarm is running it indicates we are in a reception slot.

`interface Alarm<TSymbolIEEE802154,uint32_t> as IsCfpTxSlot;` When this alarm is running it indicates we are in a transmission slot.

`interface Alarm<TSymbolIEEE802154,uint32_t> as TrackAlarm;` This is the alarm that triggers when the CFP starts. It is needed due to the `guardTime` used at the end of the CAP. Figure 4.12

`interface Timer<TSymbolIEEE802154> as GTSDescPersistenceTimeout;` Once it receives the acknowledgement to the GTS request command, the devices shall continue to track beacons and wait for at most `aGTSDescPersistenceTime` superframe. If the GTS descriptor is received and the `MLME_GTS.confirm()` is signalled this timer stops running. If the timer fired the `MLME-GTS.confirm` primitive is sent with status of NO_DATA.See [26, Sec 7.5.7.2 p. 193].

`interface TimeCalc;` Interface that provides time utilities.

`interface GetNow<bool> as IsRxEnableActive;` Interface that indicates if the reception is active.

`interface Notify<bool> as RxEnableStateChange;` Interface that notifies when the state has changed.

`interface Notify<const void*> as PIBUpdateMacRxOnWhenIdle;` Interface that notifies when the `RxOnWhenIdle` has changed.

`interface SuperframeStructure as SF;` Superframe utilities to get parameters and configuration of the beacon. Superframe start time, slot duration, number of CAP and CFP slots, GTS fields are some of the parameters we can get.

`interface RadioTx;` Interface that provides the functions to be able to transmit using the radio.

`interface RadioRx;` Interface that provides the functions to be able to receive using the radio.

`interface RadioOff;` Interface that provides the functions to turn off the radio.

`interface Notify<ieee154_status_t> as HasRequestedCfpTimeSlot;` Interface that notifies when a new beacon is received in case we are waiting for a GTS descriptor. It checks if our requested CFP slot exists on the beacon.

`interface Notify<ieee154_status_t> as CheckCfpTimeSlot;` Interface that notifies when a GTS slot has been deleted from the beacon.

`interface FrameTx as GtsRequestTx;` Interface that contains the function to send packet through the radio. It sends packets in the CAP.

`interface Pool<ieee154_txframe_t> as TxFramePool;` Interface that provides the pool that has the packets in the same queue and control it.

`interface Pool<ieee154_txcontrol_t> as TxControlPool;` Interface that controls the previous pool, and prevents it to overflow.

`interface MLME-GET;` Interface that is used to get the configuration variables from the PIB.

`interface MLME-SET;` Interface that is used to set the configuration variables to the PIB.

`interface FrameUtility;` Interface that provides utilities to access the content of a frame.

`interface GtsUtility;` See Section 4.2.5.1.

`interface IEEE154Frame as Frame;` Interface that allows to access the content of a IEEE 802.15.4 frame. It is not the same as FrameUtility.

`interface IEEE154BeaconFrame as BeaconFrame;` Interface that allows to access the content of a beacon frame.

```
#ifndef IEEE154_BEACON_TX_DISABLED
```

`interface FrameRx as GtsRequestRx;` Interface that is used to transmit the `MLME_GTS.request` and detect the acknowledge.

`interface Get<ieee154_GTSdb_t*> as GetGtsCoordinatorDb;` Interface that is used just when we compile the program for the coordinator. (see on page 63)

```
#else
```

`interface Get<ieee154_GTSentry_t*> as GetGtsDeviceDb;` Interface that is used just when we compile the program for the coordinator. (see on page 65)

```
#endif
```

Provides

`interface Init;`

`interface MLME_GTS;` Interface that provides all the primitives necessary to use the GTS. The `request()`, `confirm()` and `indication()` are provided.

`interface Notify<bool> as GtsSpecUpdated;` Interface that notifies to other components that the GTS descriptor in the beacon has to be rewritten.

`interface GetNow<bool> as IsGtsRequestOngoing;` Interface that indicates to other components that there is one `MLME_GTS.request` on going. While there is one request on going, it is not possible to process another one.

`interface GetNow<bool> as IsGtsOngoing;` Interface that indicates to other components that the GTS is working and we are receiving/transmitting through the GTS.

`interface Get<ieee154_GTSentry_t*> as GetGtsRequestedDeviceEntry;` Interface that is used to share the information about the requested GTS entry.

`interface FrameTx as CfpTx;` Interface that is used to transmit data in the contention free period instead of contention access period.

`interface FrameRx as CfpRx;` Interface that is used when a packet is received in the contention free period.

`interface Notify<bool> as WasRxEnabled;` Interface that indicates if the radio was enabled to receive packets.

Variables

```
/* ------------ Vars to CFP tx ------------ */
```

`norace ieee154_status_t m_txStatus;` Indicates the transmission status.

`norace uint32_t m_transactionTime;` To store the transaction time needed for the packets.

`norace uint16_t m_slotDuration;` To store the slot duration of the beacon. It is equal to: $timeSlotDuration = aBaseSlotDuration * 2^{SO}$

`norace uint16_t m_guardTime;` That guard time, it is the time that the radio needs to change its state to inactive. (See [39]).

`norace uint32_t m_capDuration;` It is the CAP duration. When the radio resource is transferred to the `CfpTransmitP` component, that time is recalculated because it is possible that the CAP has changed.

$capDuration = numCapSlots * timeSlotDuration = numCapSlots * aBaseSlotDuration * 2^{SO}$

`norace uint32_t m_gtsDuration;` It is the CFP duration. When the radio resource is transfer to the `CfpTransmitP` component, that time is recalculated because it is possible that the CFP has changed.

$gtsDuration = cfpDuration = numCfpSlots * timeSlotDuration = numCfpSlots * aBaseSlotDuration * 2^{SO}$

`norace uint32_t m_cfpInit;` It is the time when the contention free period has to start. It is used by the TrackAlarm to align the device and the coordinator.

$cfpInit = capDuration + superframeStartTime$

`norace ieee154_macMaxFrameRetries_t m_macMaxFrameRetries;` Variable that is used to control the number of retransmission that are available.

`norace ieee154_macMaxFrameTotalWaitTime_t m_macMaxFrameTotalWaitTime;` Variable that stores the maximum time that it waits until the acknowledge is received and we have to retransmit the packet, if the `m_macMaxFrameRetries` is not achieved.

`norace ieee154_macRxOnWhenIdle_t macRxOnWhenIdle;` Variable that indicates in which state of the radio, during the idle state. If it `TRUE` the radio has to be in reception state, to wait for new packets.

`norace bool m_lock;` To prevent non-atomic access to a variable, instead of using atomic block, because although it will assure that the block is accessed with atomic access, it will take some time.[21, Sec. 4.5]

`norace uint8_t m_slotNumber;` Indicates the slot number, where it is at the moment. Variable that is used to load the entry that belongs to the actual slot number.

`norace ieee154_txframe_t *m_currentFrame;` Variable that points to the actual frame, that has been prepared to send.

`norace ieee154_txframe_t *m_lastFrame;` Variable that points to the last frame that has been sent.

`/* ------------ Vars to GTS request ------------ */`

`uint8_t m_payloadGtsRequest[2];` Variable that contains a vector where the payload of the `MLME_GTS.request` is stored. The first byte indicates the command type (*CMD_FRAME_GTS_REQUEST*), and the second byte contains the information of the requested slot: short address, length, direction, and type (allocate or deallocate).

`norace bool m_gtsOngoingRequest = FALSE;` See entry `interface GetNow<bool> as IsGtsRequestOngoing;` on page 68.

`norace bool m_gtsOngoing = FALSE;` See `interface GetNow<bool> as IsGtsOngoing;` on the previous page.

`bool m_hasRequestedTimeSlot = FALSE;` See `interface Notify<ieee154_status_t> as HasRequestedCfpTimeSlot;` on page 67.

`uint8_t m_gtsCharacteristicsType = GTS_DEALLOCATE_REQUEST;` Variable that indicates the characteristics type of the request.

`ieee154_GTSentry_t GTSentryRequested;` Variable that contains the information of the request slot.

`ieee154_GTSentry_t GTSentryRequestedEmpty;` Variable that contains the information of the request slot, to be deleted.

```
/* ------------ Vars to expiration GTS ------------ */
```

`ieee154_GTSentry_t* actualEntry;` Variable that points to the actual slot entry depending on the slot number.

`norace uint16_t m_expiredTimeout;` Variable that stores the number of superframes that the coordinator has to wait if the slot it is not used. Afterwards the coordinator starts the GTS expiration mechanism.[26, Sec. 7.5.7.6]

$$n = 2 * \begin{cases} 2^{8-macBeaconOrder} & 0 \leq macBeaconOrder \leq 8 \\ 1 & 9 \leq macBeaconOrder \leq 14 \end{cases}$$

`norace uint8_t m_expiredReset[CFP_NUMBER_SLOTS];` Variable that is used to count the number of superframes in which a slot is not been used. Once this variable is equal to 0, the coordinator proceed the expiration mechanism. When a new slot is requested or when the slot is being used, that variable is reset to `m_expiredTimeout`.

4.2.5 Utilities

The utility component provides tools to manage and configure the GTSs.

4.2.5.1 GtsUtility

The `TKN154.4` does not provide any functions to access the GTS variables in the beacons. Note that although they are related to the beacon frame or the data frame, it has been considered as best option to do it in a different interface, not in the `FrameUtility` or `IEEE154BeaconFrame`.

Table 4.4 explains the functions and commands implemented. The first group is focused on the frame, as a `MLME_GTS.request` primitive, and the beacon frame group provides the functions to read, parse and get the GTS descriptor. Furthermore it provides utilities to manage the GTS database that needs the coordinator.

4.3 Discussion

In spite of having a working GTS implementation, there are few known issues and problems to be discussed:

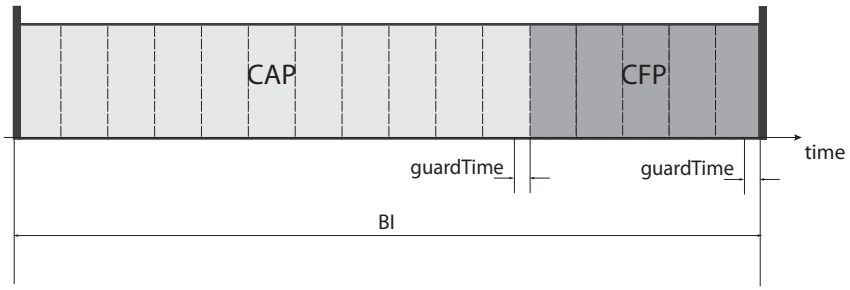
Function Name	Description
Frame functions	
`uint8_t getGtsCharacteristics(...)`	Get the GTS Characteristics from a frame payload
`uint8_t setGtsCharacteristics(...)`	Return the GtsCharacteristics to create a frame
`ieee154_status_t parseGtsCharacteristicsFromFrame(...)`	Parse a frame to get the GtsCharacteristics
`ieee154_status_t parseGtsCharacteristicsFromPayload(...)`	Parse a payload to get the GtsCharacteristics
BeaconFrame functions	
`uint8_t getGtsEntryIndex(...)`	Get the GTS entry index, in case we have it, otherwise return the new one
`error_t addGtsEntry(...)`	Add the new entry to the coordinator database
`error_t purgeGtsEntry(...)`	Purge the entry in the coordinator database. It reallocates the rest of the slots, to keep it in order
`error_t getGtsFields(...)`	Reads the GTS Fields of a beacon frame (except GTS List)
`error_t getGtsList(...)`	Reads the GTS List of a beacon frame
`ieee154_status_t hasRequestedTimeSlot(...)`	Return the requested slot configuration in the ieee154_GTSentry_t*
`ieee154_status_t checkTimeSlot(...)`	Once we have the resource, we just need to check if our slot is still in the beacon
`void setNullGtsEntry(...)`	Empty the slot. Set the length and starting slot to zero; direction to two (invalid) [2] and expiration to `FALSE`

Table 4.4: GtsUtility functions

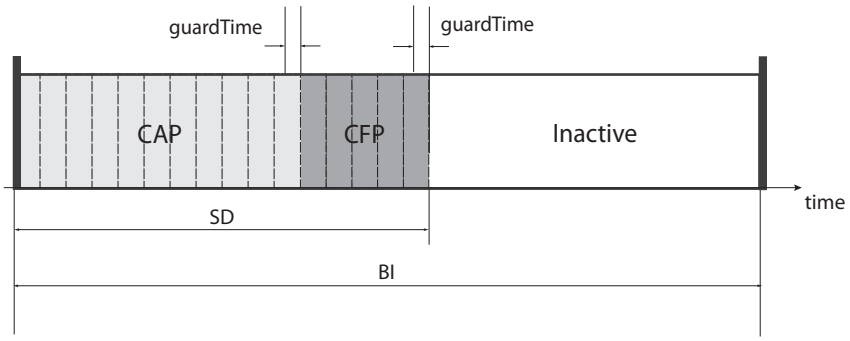
- **Guard time:** According to [39] there is a switching time between sleep and active states for the radio chip (CC2420). The time is called `guard time` $guardTime = 420symbols = 6.720ms$.

  Figure 4.18a shows the guard time in the TKN15.4 when the superframe involves CAP and CFP. In Figure 4.18b with BO≠ SO, we can see that the guard time used in CAP is not needed, and it takes time in the CAP. In Figure 4.18a we do not need any guard time, but the implementation has them because it does not distinguish between $BO = SO$ or $BO \neq SO$.

  Therefore, we see the necessity of deleting this time in the cases where we do not need it because it decreases the useful time of the period. Moreover, as far as the guard time is not negligible for small SO, it would not allow the use of the GTS. Hence, it is really important to analyse this problem deeply, in



(a) $BO = SO$



(b) $BO \neq SO$

Figure 4.18: Influence of guard time

order to be able to provide a GTS for low SO using the CC2420.

- **Superframe Order:** The guard time showed before and the timing limitation with the 32.768 kHz crystal, are the factors that does not allow the implementation to be standard compliant. At the moment it is possible to use $4 < SO \leq 14$.

  For $SO < 4$ we have a time slot duration lower than the guard time, $timeSlot < guardTime$. So the last time slot shall not be possible to use. In Figure 4.18, we can see the influence of the guard time in the last slot on the superframe.

  We are planning to use our implementation in the new Zolertia Z1 [25] motes with the new MSP430 Series 2 which allows to use crystal faster than 32.768 kHz as the Series 1.

- **Missed beacons:** As we have seen in Section 3.3.2.2, the TKN15.4 implementation has a problem of missing beacons when a network is already running. Given a network with several devices, once the device synchronizes with the PAN Coordinator after the `MLME_SCAN`, it is possible to lose some beacons. It also happens when we have a big interference or a device transmitting with high sample rate in the same channel. In the standard specifications, it is defined that once a node lose the synchronization it shall stop using the allo-

cated slots. So in this case, it will be necessary to start a `MLME_GTS.request` mechanism again.

- **Memory limitation:** There is a limit of 48 Kbytes in the ROM memory. With the GTS implementation, the debug utility provided by TKN15.4, `dbg_serial`, is not possible to use. In the default TKN15.4 with the coordinator was impossible to use as well.

  By using the new Zolertia Z1 motes, we shall overcome this problem, because the Z1 has 96 kBytes of ROM, instead of 48kBytes that TmoteSky/Telosb have.

4.4 Future work

To achieve the desired target of a complete implementation, there are few issues that are necessary to implement.

- **Time slot duration:** In Section 4.2.3.2, we saw that the time slot duration is correct, but there is one thing that would improve it. To be sure that the initial time of the slot is correct, it is possible to modify the timer. Instead of using `call AlarmXXX.start(timeSlotDuration)`, use `call AlarmXXX.startAt (initSlotTime, timeSlotDuration)`. Figure 4.19 shows this improvement.

- **Superframe Order:** The IEEE 802.15.4 protocol is designed to be able to use BO and SO between 1 to 14 for beacon-enabled mode. Hence, to achieve a full implementation we need to be able to use all the range available, that at the moment is limited, $4 < SO \leq 14$. As we have seen, our big challenge is for $SO < 4$ where the guard time set this limitation.

- **Configure slot length:** In the `MLME_GTS.request()` the device has to send the requested GTS slot length. At the moment this functionality is not implemented, the slot length is set to one.

- **Invalid GTS slot:** If there was not sufficient capacity to allocate the requested GTS, the start slot shall be set to zero and the length to the largest GTS length that can be currently be supported. [26, Sec. 7.5.7.2]. In this case, it is allow to expand the CFP and use part of the CAP.

- **CAP Maintenance:** As far as we do not have the configuration of the slot length available yet, we assure that the CAP length is bigger than *aMinCAPLength* symbols. [26, Sec. 7.5.7.1] The maximum slot number that the CFP could have is$7 * slotLength = 7$, the the minimum value that we can get

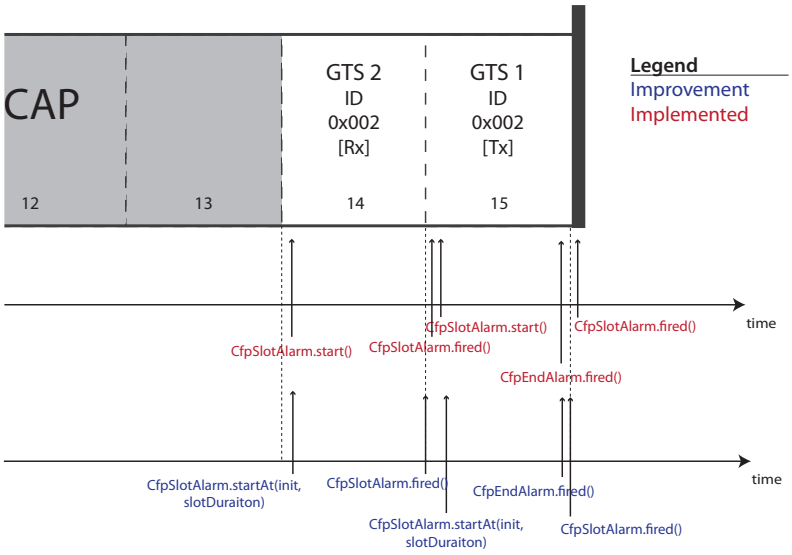


Figure 4.19: Comparison between the time slot duration with the actual implementation and the improvement for the future work. In the actual implementation, we can see the influence of a small delay in the start timer. In the improvement, the timer computes the difference between the starting time and the final, and then the delay does not influence.

is:

$$CAPlength \geq aBaseSlotDuration * (aNumSuperframeSlots - 7) * 2^{SO}|_{min(SO)}$$
$$\geq 1080 symbols$$

and the default $aMinCAPLength = 440 symbols$. So in our implementation there is no problem, but as far as the slots length will be optional, it is necessary to implement it.

Chapter 5

Performance evaluation of IEEE 802.15.4

In the previous chapter, the IEEE 802.15.4 protocol and the GTS implementation were described. There are a lot of factors that could influence the behaviour of the protocol, namely: noise, interferences from other networks and hardware restrictions (delay, precision, accuracy, clock drift). As a consequence, it is important to consider a strong performance evaluation.

In this chapter we analyse the delay and reliability for different traffic load η and MAC parameters.

5.1 System model

For the performance evaluation, the system model defined in [29] is implemented in the motes with some variations. It provides us a good reference of how the performance shall be.

We consider a star network with a PAN coordinator and N nodes. All N nodes contend to send data packet to the coordinator, which is the data sink. In this analysis we consider applications where nodes asynchronously generate packets with probability η_t when a node sends a packet successfully, discards a packet or the GTS request has failed. And with probability η_p when the sampling interval has expired. A node stays for hT_b without generating packets with probability $1 - \eta_t$, where h is an integer and T_b is the time unit *aUnitBackoffPeriod*, corresponding to 20 symbols (320 μs).

In general, we consider two different types of data packets: non-time critical data packets transmitted during CAP and time critical data packets during CFP using GTS allocation mechanism. When a node decides to generate a data packet, it generates a non-time critical data packet with probability η_d and time critical data packet with probability $1 - \eta_d$.

A node that uses beacon-enabled slotted CSMA-CA algorithm, for sending a time critical data packet, first sends a GTS request to coordinator during CAP. Once the `MLME_GTS.confirm` is signalled (due to a GTS descriptor reception or and error), each node may need to send a multiple number of time critical packets τ_n during CFP once the slot is allocated. For the non-time critical data packet, the node sends one packet during CAP. Note that the packet transmission is successful if an ACK packet is received.

Figure 5.1 illustrates the system model implemented in the motes. The grey circles boxes indicate the states of the system. The light blue boxes indicate command to send or a received event.

5.2 Network scenarios

In this chapter we show different modes that IEEE 802.15.4 allows. In the beacon-enabled, we use the slotted CSMA-CA in CAP and GTS in CFP. The non beacon-enabled use the unslotted CSMA-CA.

Figure 5.2 shows the topology considered for this performance evaluation. All N=9 motes are placed surrounding the coordinator between (50cm - 1m). They have a low transmission power to avoid possible interferences between motes.

Following, reliability and delay are shown for different configurations, and as function of some MAC parameters. It does not intend to be a wide comparative

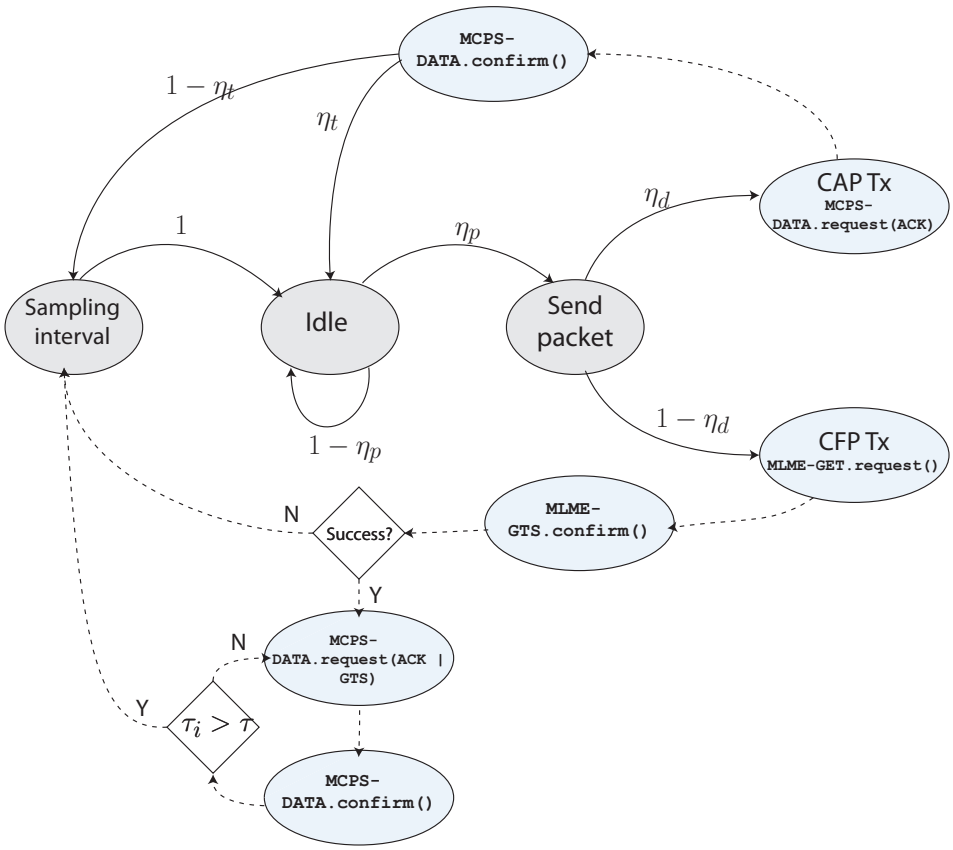


Figure 5.1: System model diagram

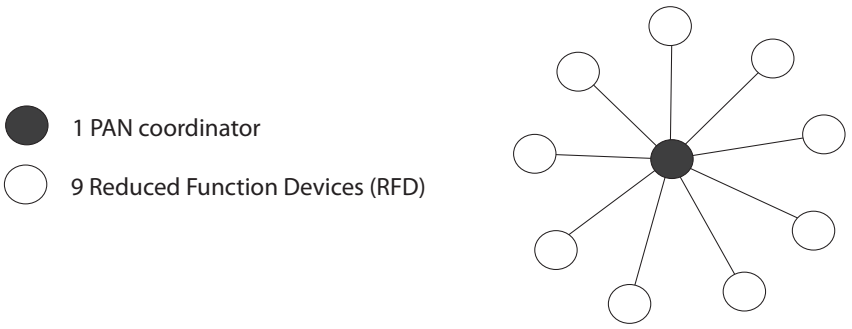


Figure 5.2: Star network topology

between the real implementation and the theoretical model found in [29, 30]. This chapter shows the results from the real implementation using the TKN15.4 and our GTS implementation.

The reliability is the packet reception rate of the network. And the delays show in this chapter are defined as the variation of time between the generated and

received packets. So, it give as the idea of the delay introduced by the CSMA-CA mechanism influences the network. Mention that in papers where the performance evaluation is done, the delay is refereed to the time between a generated packet and when its ACK is received.

Name	Value	Description
SO	6	Superframe Order. This value set the duration of the superframe
BO	6	Beacon Order. This value set the duration of the superframe plus inactive period. As we have the same value $BO = SO = 6$ the inactive period is 0.
T_b	20 symbols	*aUnitBackoffPeriod*
h	500	the number of backoff units. Sampling interval
N	9	Nodes contend to send data packet to the coordinator
n	1	*macMaxFrameRetries*
m	4	*macMaxCSMABackoffs*
m_b	8	*macMaxBE*
m_0	3	*macMinBE*
τ_n	2	The number of time critical data packets for each GTS request.
L_p	50 bytes	Packet length containing the packet header (11 bytes) and payload (39 bytes).
η_d		Probability of non-time critical packet (CAP)
$1 - \eta_d$		Probability of time critical packet (CFP)
η_p		Probability to generate a packet when the sampling interval is expired
η_t		Probability to generate a packet when a node send a packet successfully or discard a packet.
$\eta = \eta_p = \eta_t$		Traffic load

Table 5.1: Default parameters for CSMA-CA

Table 5.1 shows the involved parameters and their default values for our system model.

5.2.1 Beacon-enabled vs non beacon-enabled

In this section we compare the beacon-enabled and the non beacon-enabled mode to see which mode provides a better performance. In the beacon-enabled mode all the motes have their backoff boundaries aligned which reduces the probability of collisions, however, in the non beacon-enabled mode the motes have their own backoff boundaries with no synchronization between motes. In Section 3.3.3, we have seen influence of the clock drift on the CCA mechanism, because even with

the beacon-enabled mode where the motes shall be aligned, they could have some drift.

5.2.1.1 Reliability

In slotted CSMA-CA, packets are discarded due to two reasons: (i) CCA failure ; (ii) retransmission limits. CCA failure happens when a packet fails to obtain idle channel in two consecutive CCAs within $m + 1$ backoffs. Furthermore, a packet is discarded if the transmission fails due to repeated collisions after $n + 1$ attempts.

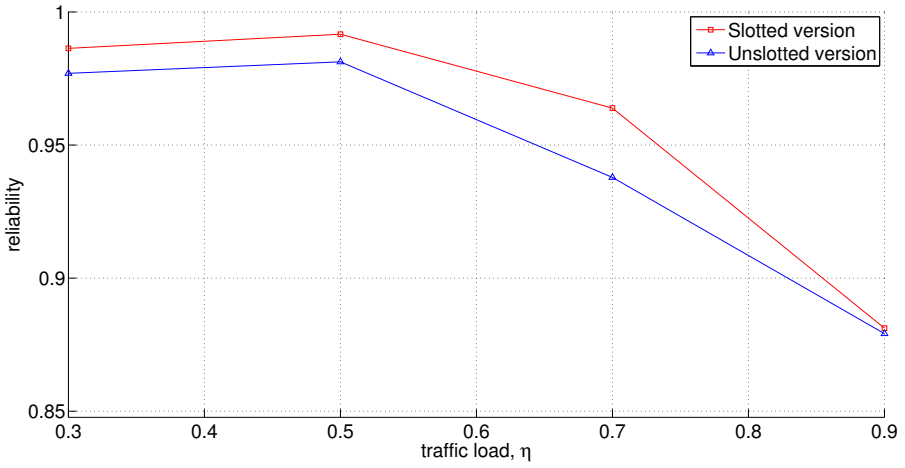
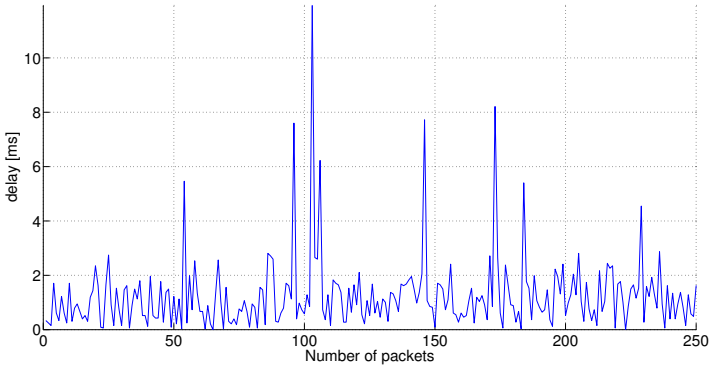


Figure 5.3: Reliability as a function of the traffic conditions $\eta = \eta_p = \eta_t = 0.3, 0.5, 0.7, 0.9$ for slotted and unslotted CSMA-CA, with a given probability for generating non-time critical packet $\eta_d = 1$, superframe and beacon order $BO = SO = 6$ (for slotted version), MAC parameters $n = 1$, $m_b = 8$, $m_0 = 3$, $N = 9$, $h = 500$ and the packet length $L_p = 50bytes$.

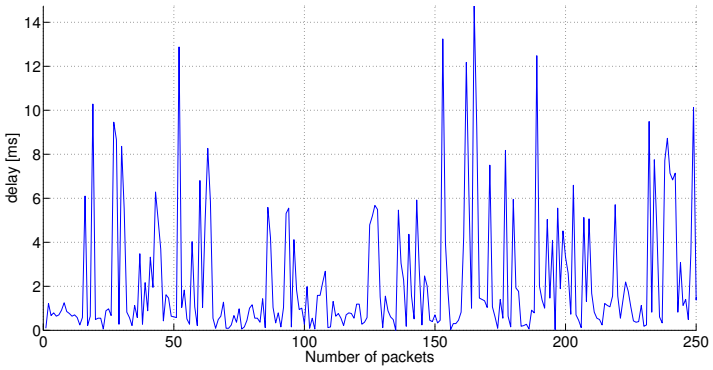
Figure 5.3 compares the reliability of the slotted and unslotted CSMA-CA as a function of the traffic load η with a given probability for generating non-time critical packets η_d. The slotted version has a better reliability than the unslotted version. Reliability decreases as the traffic load increases, but for the lowest traffic load the reliability should be closer to 1. This difference could come from the imperfection of the implementation or external interferences in our work place (radio communications, WiFi, other WSN).

The difference of both probabilities is set around 0.01 for $\eta \leqslant 0.7$. For $\eta = 0.9$ the reliability is similar in both cases. This behaviour is reasonable due to the the big number of collisions present in $\eta = 0.9$. All the nodes wants to transmit almost continuously, it does not influence the fact of transmitting in the backoffs boundaries (slotted) or every time after we detect the channel clear (unslotted).

5.2.1.2 Delay



(a) Slotted CSMA-CA with $SO = BO = 6$



(b) Unslotted CSMA-CA

Figure 5.4: Snapshot of delay for $\eta = \eta_p = \eta_t = 0.5$, with a given probability for generating a non-time critical packet $\eta_d = 1$, superframe and beacon order $BO = SO = 6$ (for slotted version) and MAC parameters $n = 1$, $m_b = 8$, $m_0 = 3$, $N = 9$, $h = 500$ and the packet length $L_p = 50bytes$.

Figure 5.4 shows a snapshot of the delay for 250 packets. The delay for the slotted version is lower than the unslotted, where we see a lot of packets between 4 and 10 ms.

Figure 5.5 shows the delay as a function of the traffic load η for the slotted and unslotted version. In case of unslotted the values for $\eta \leqslant 0.5$ should be lower and increasing when η increases.

As we were expect, with the slotted version we get better performance in terms of reliability and delay. We conform the importance of the alignment in the motes

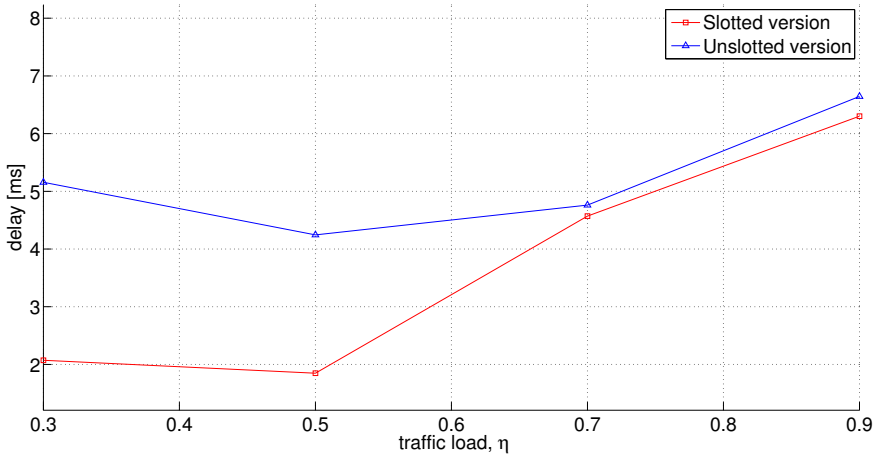


Figure 5.5: Delay as a function of the traffic conditions $\eta = \eta_p = \eta_t = 0.3, 0.5, 0.7, 0.9$ for slotted and unslotted CSMA-CA, with a given probability for generating a non-time critical packet $\eta_d = 1$, superframe and beacon order $BO = SO = 6$ (for slotted version) and MAC parameters $n = 1$, $m_b = 8$, $m_0 = 3$, $m = 4$, $N = 9$, $h = 500$ and the packet length $L_p = 50bytes$.

in order to reduce the collisions. Even though we know that the clock drift of the motes influences the alignment, we see that the influences is not big enough to make the slotted worse than the unslotted. In the delay we see a significant difference for low traffic load between the slotted, unslotted version.

5.2.2 Beacon-enabled and MAC parameters

In this section, we focus in the beacon-enabled mode. We analyse the influence of MAC parameters, *macMinBE*, *macMaxFrameRetries* and *macMaxCSMABackoffs*.

5.2.2.1 Reliability

Figure 5.6 shows the reliability as a function of the traffic load η with a given number of nodes N=9 and different MAC parameters, m_0, m and n. In Figures 5.6a, 5.6b the reliability increases as MAC parameters m_0, m increase, respectively. However, in Figure 5.6c the reliability does not improve as the retry limits n increases for high traffic conditions $\eta \geq 0.7$. Notice that the reliability saturates if $n \geq 3$. Hence, the retransmissions are necessary but not sufficient for high reliability under high traffic load.

5.2.2.2 Delay

Figure 5.7 shows the average delay as a function of the traffic load η with a given number of nodes N=9 and different MAC parameters, m_0, m and n. Observe that the average delay increases as traffic load increases due to high busy channel probability and collision probability. In Figures 5.7a, 5.7b and 5.7c the delay increases as MAC parameters m_0, m and n increases, respectively. However, Figure 5.7a shows that the average delay increases exponentially as m_0 increases. This behaviour is due to the CSMA-CA algorithm. In each backoff period it has a random period of $2^{BE} - 1$ backoffs. Hence, we conclude that m_0 is the key parameter in terms of average delay with respect to m and n.

It is important to remark which are the most important parameters that influences in the performance of our network. For the reliability, increasing the number of retransmissions we improve the reliability but moreover more packets are sent to the network, so the reliability saturates. But if we increase the number of retransmissions then the delay will be influenced, so it is important to use the value of retransmissions that gives the highest reliability with a low delay. In our case, $n = 3$ gives the best performance.

Moreover the delay is highly influenced by the MAC parameter macMinBE ($m0$). The CSMA-CA algorithm generates a random delay where m_0 is the minimal value of the exponent. For this reason, if we increase this parameter the delay will increase exponentially.

5.2.3 Beacon-enabled with hybrid MAC

Section 5.1 shows the system model used in this chapter, but until now, we analyse the performance transmitting during the CAP ($\eta_d = 1$), sending non-critical packets. Now, we analyse the hybrid MAC, where we transmit during the CAP or CFP depending of the probability of critical packets ($1 - \eta_d$). For sending a time critical data packet, first sends a GTS request to coordinator during CAP and send a multiple number of time critical packets τ_n during CFP once the slot is allocated.

This section present the performance analysis done for a hybrid MAC in a real implementation. Even though, the system model implemented is not the one introduced in [29], It gives a good approximation about the expected values.

5.2.3.1 Transient behaviour

In this section, we analyse the transient behaviour of the network, keeping track of the slots allocated, received data packets and received request packets.

Figure 5.8 shows the snapshot of a number of received data packets, received GTSs requests, and allocated GTSs. It shows them as a function of traffic load. The convergence of the values is reached around 30 superframes.

The received data packets are the packets sent by the device with its ACK from the coordinator. The received GTSs request are the GTS commands sent by the device with its ACK, they are the number of GTS request that the coordinator processes. The allocated GTSs are the slots allocated by the coordinator. This information is extracted from the GTS descriptor for each beacon.

In Figures 5.8a and 5.8b, are shown the received packets, request and data, as a function of traffic load. In case of the allocated GTSs, in Figure 5.8c we can see the saturation for $\eta = 0.7, 0.5$ while for $\eta = 0.3, 0.1$ it converges around the 6 and 5 slots allocated, respectively.

In Figures 5.8a and 5.8b, if we focus in one peak when it converges, for example 60, the number of data packets increases when request packets decreases. It is due to the system model implemented, because to send a data packet during the CFP we need to send a request packet and those are not counted as a received data packets.

5.2.3.2 Reliability

Figure 5.9 shows the reliability transmitting in the CAP and CFP as a function of the traffic load η with a given number of nodes N=9 and different MAC parameters, m_0, m and n. The reliability decreases as the traffic load increases. Note the evolution and behaviour of the plots for $\eta_d = 0.5$ and $\eta_d = 0.8$ do not follow a continuous behaviour. For $\eta_d = 0.5$ the reliability should be bigger than $\eta_d = 0.8$.

As we saw in Section 4.4, the time slot alarm could be influenced by delays. With a high traffic load, the motes is trying to send packets more often and these interruptions could occur during the CFP. Moreover, this drift influences the ACK reception. Even if the packet has been sent correctly, the sender do not receive the ACK sent by receiver.

Hence, we can see how important is the improvement proposed in 4.4 for the time slot duration.

5.2.3.3 Throughput

Here we analyse the throughput of the hybrid MAC, namely, the average amount of both non-time critical packets and critical packets that are transmitted in the superframe, during a length of beacon interval T_{BI}. To be able to compare with different BO, we use a normalized throughput,

$$\Theta = \frac{N_{CAP}L_{CAP} + N_{CFP}L_{CFP}}{T_{BI}} \tag{5.1}$$

where N_{CAP} represents the number of successfully received non-time critical packets, L_{CAP} the length of the packets transmitted during the CAP, and N_{CFP} and L_{CFP} are the equivalent values for the time critical packets transmitted during the CFP. Note that the terms $N_{CAP}L_{CAP}$, $N_{CFP}L_{CFP}$ and T_{BI} are normalized by the slot unit.

Figure 5.10 shows the throughput of hybrid MAC as a function of traffic load η, the probabilities of generating a time critical packet $1-\eta_d$ and superframe order $SO = 6$ with a given MAC parameters $m_b = 8$, $m_0 = 3$, $m = 4$, $N = 9$, $h = 500$, the number of time critical data packets for each GTS request $\tau_n = 2$, and the packet length $L_p = 5$. Figure 5.10c presents the hybrid throughput, the sum of throughput of CAP in Figure 5.10a and CFP in Figure 5.10b. Note that it is not a comparison between CAP and CFP, it a a decomposition for the hybrid throughput. We observe that as probability for generating a time critical packets $1-\eta_d$ increases, the throughput of CFP increases, however the throughput of CAP decreases. CAP throughput has a similar trend as the Hybrid throughput. The throughput of CAP is more dominant factor for the hybrid throughput of the network. This is due to the limited number of GTSs slots in the superframe.

5.2.4 Validation with theoretical model

Figure 5.11 shows the reliability for our experimental results and the theoretical model. Reliability as a function of traffic load η using the slotted scheme with a given probability for generating a non-time critical packet $\eta_d = 1$. Increasing the traffic load, with the experimental results, we see a decreasing reliability, but in the theoretical model it does not happen until $\eta \geq 0.6$. In the maximum $\eta = 0.9$ we see that the theoretical model is between the maximum value and the minimum that we get with the real implementation.

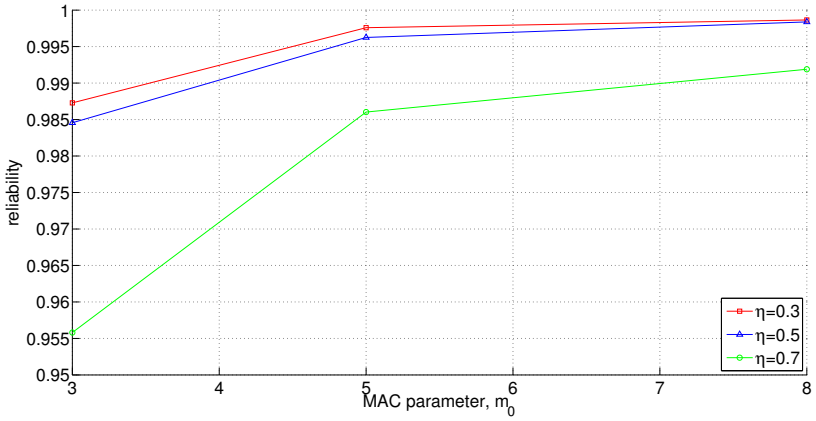
We can see how close the behaviour of our network is in comparison with the theoretical model.

5.3 Future work

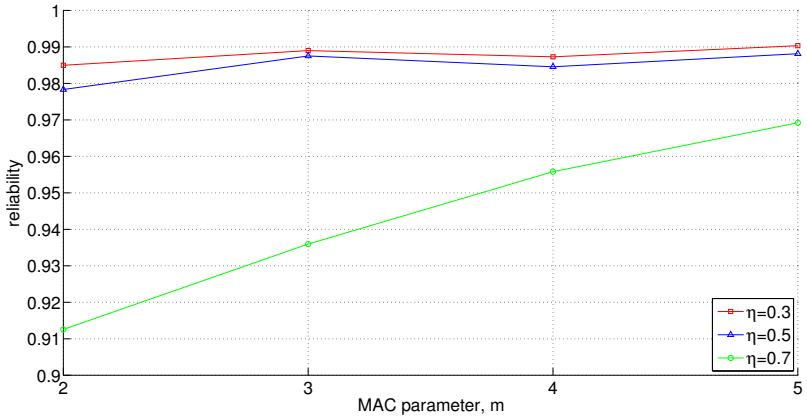
- **Validation with theoretical model:** An extensive performance analysis regarding reliability, delay, throughput, energy consumption, collision probability and probability/cumulative distribution function is needed in order to assure that the behaviour and characteristics of this implementation are close to the model. [29, 30] are two references where an extensive performance evaluation is done for the CAP and for the Hybrid MAC, respectively.

- **Interference analysis:** As far as this standard uses the ISM radio band, other networks and devices surrounding could influence in the reliability,

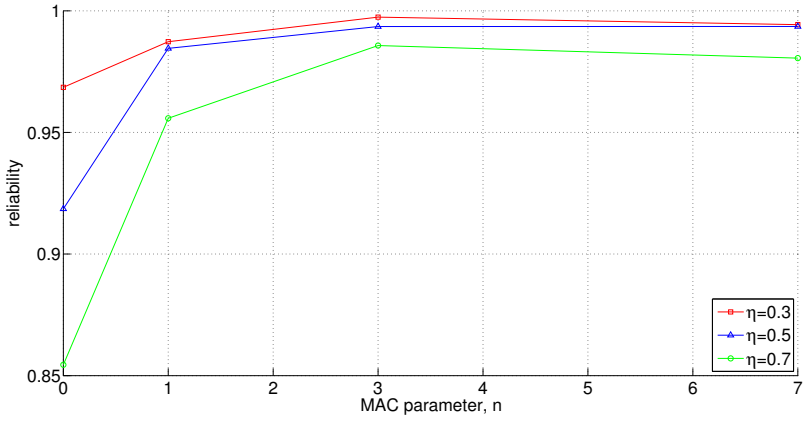
throughput and delay. [2, 16, 32] give some analysis on the impact of the IEEE 802.11 g/n in IEEE 802.15.4 network, or the influence of Received Signal Strength Indication (RSSI) levels. The theoretical model do not take into account these interferences and it will be important to analyse with the real deployment how much the interferences influences in terms of reliability.



(a) $m_0 = 3, 5, 8$

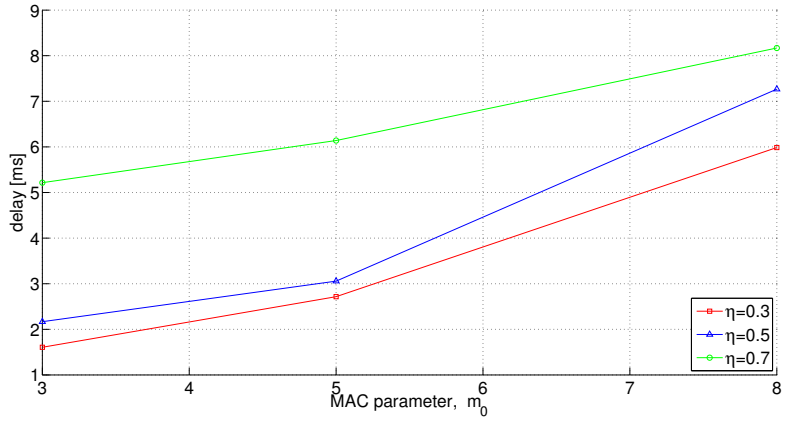


(b) $m = 2, 3, 4, 5$

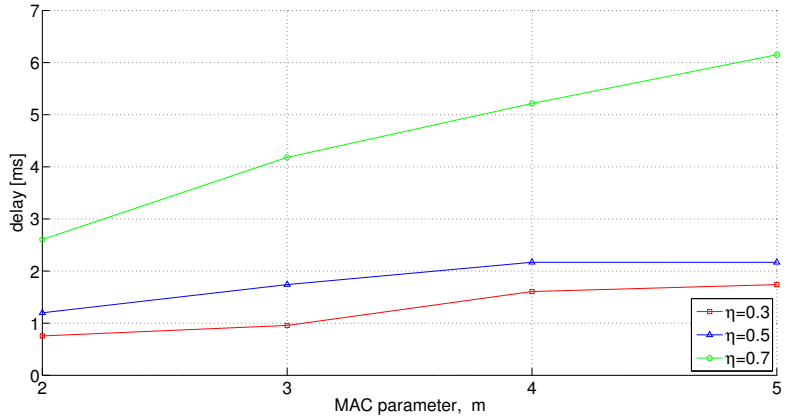


(c) $n = 0, 1, 3, 7$

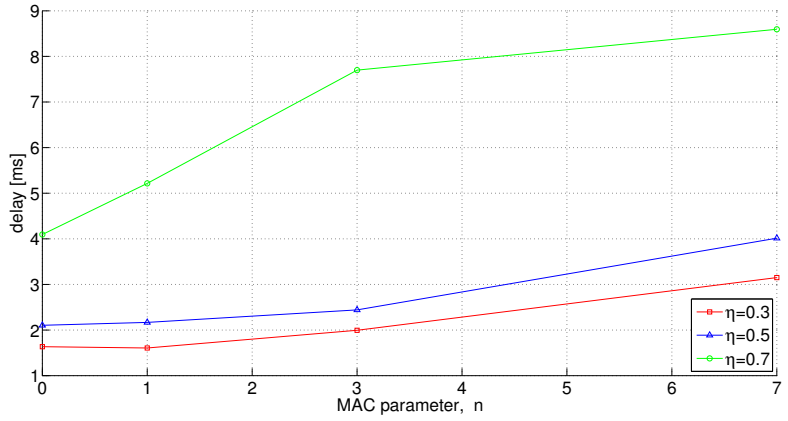
Figure 5.6: Reliability as a function of traffic load $\eta = \eta_p = \eta_t = 0.3, 0.5, 0.7$ and MAC parameters $m_0 = 3, 5, 8$, $m = 2, 3, 4, 5$, $n = 0, 1, 3, 7$ given probability for generating a non-time critical packet $\eta_d = 1$, superframe and beacon order $BO = SO = 6$. $N = 9$, $h = 500$ and the packet length $L_p = 50bytes$.



(a) $m_0 = 3, 5, 8$

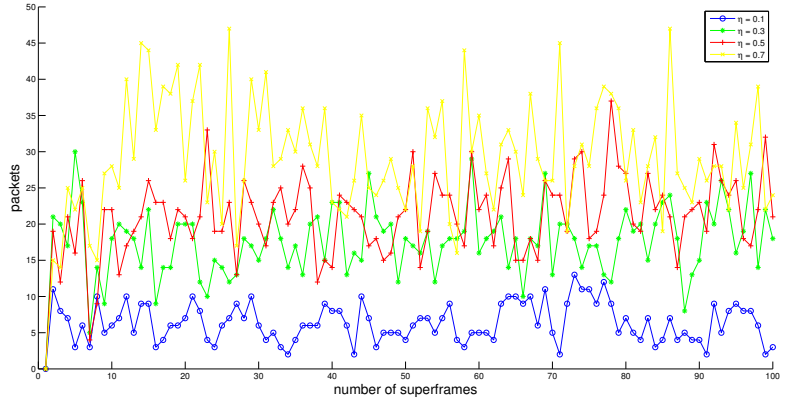


(b) $m = 2, 3, 4, 5$

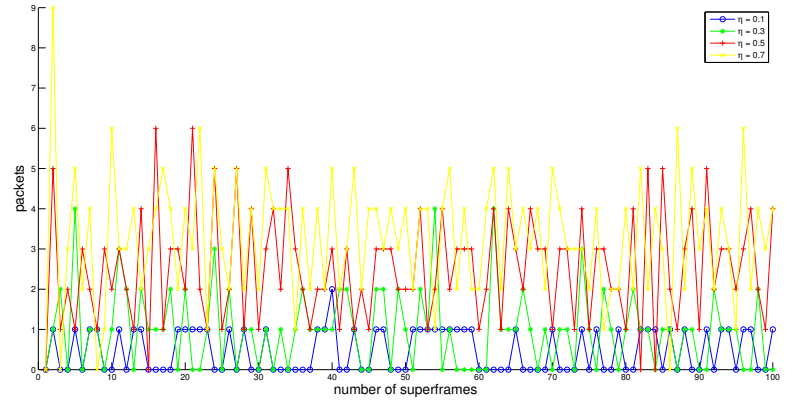


(c) $n = 0, 1, 3, 7$

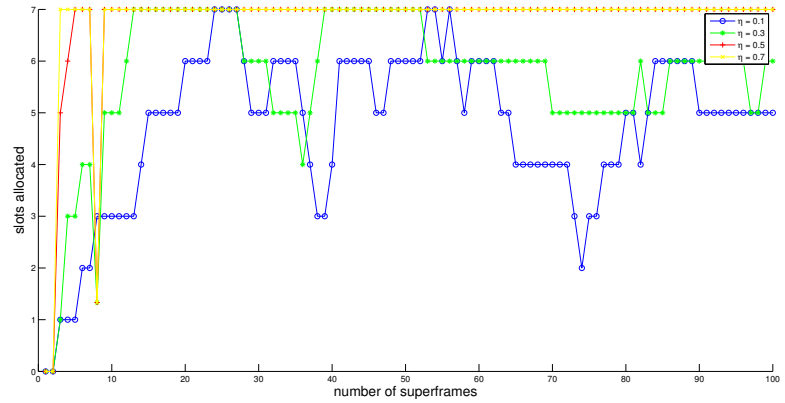
Figure 5.7: Delay as a function of traffic load $\eta = \eta_p = \eta_t = 0.3, 0.5, 0.7$ and MAC parameters $m_0 = 3, 5, 8$, $m = 2, 3, 4, 5$, $n = 0, 1, 3, 7$ given probability for generating a non-time critical packet $\eta_d = 1$, superframe and beacon order $BO = SO = 6$. $N = 9$, $h = 500$ and the packet length $L_p = 50bytes$.



(a) Received data packets



(b) Received requests



(c) Allocated GTSs

Figure 5.8: Snapshot of number of received data packets, received request, and allocated GTSs as a function of traffic load $\eta = \eta_p = \eta_t = 0.1, 0.3, 0.5, 0.7$, with a given probability for generating a non-time critical packet $\eta_d = 0.8$,superframe and beacon order $BO = SO = 6$, MAC parameters $n = 1$, $m_b = 8$, $m_0 = 3$, $N = 9$, $h = 500$, the number of time critical data packets $\tau_n = 2$ for each GTS request and the packet length $L_p = 50bytes$.

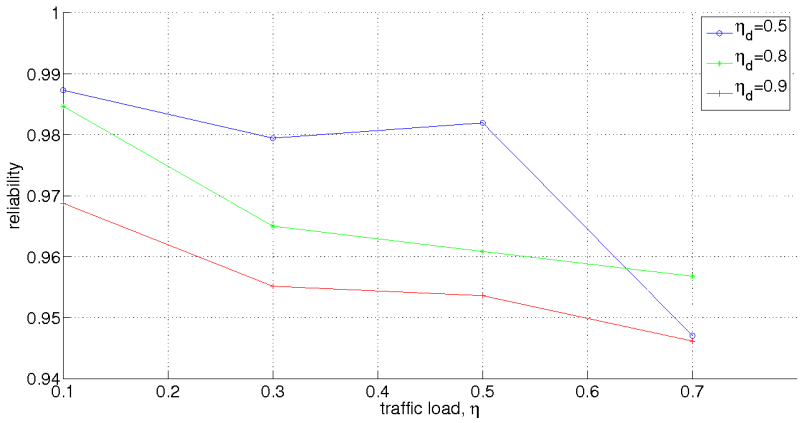
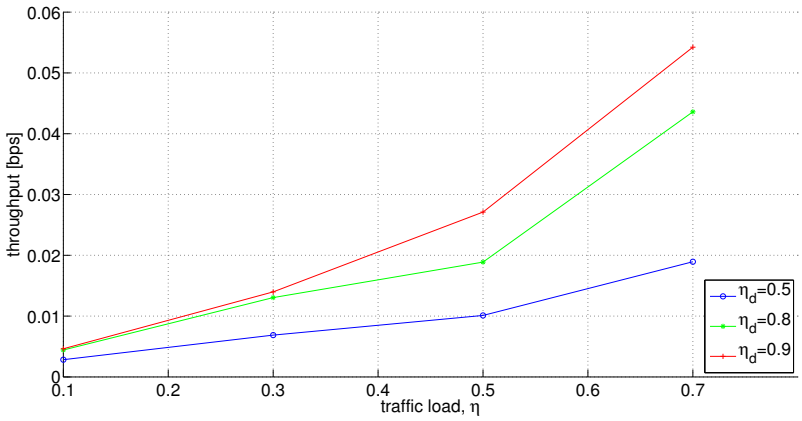
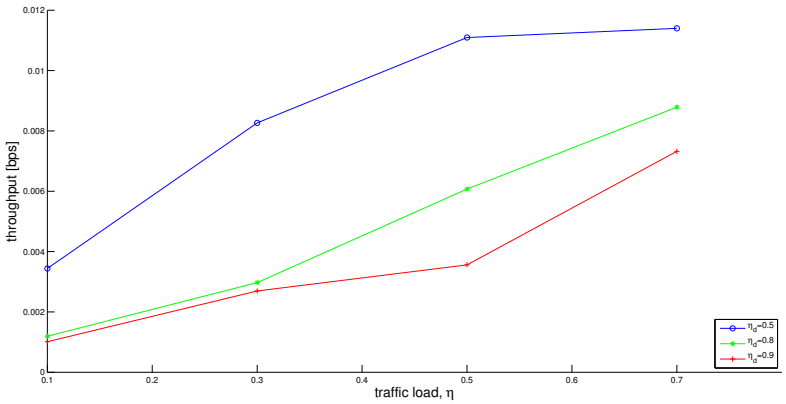


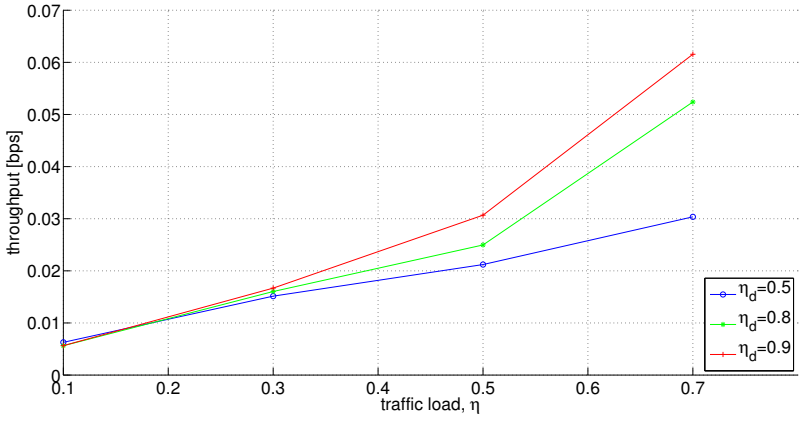
Figure 5.9: Reliability as a function of the traffic load $\eta = \eta_p = \eta_t = 0.1, 0.3, 0.5, 0.7$ with a given probability for generating a non-time critical packet $\eta_d = 0.5, 0.8, 0.9$, superframe and beacon order $BO = SO = 6$, and MAC parameters $m_b = 8$, $m_0 = 3$, $m = 4$, $N = 9$, $h = 500$ and the packet length $L_p = 50bytes$.



(a) CAP throughput



(b) CFP throughput



(c) Hybrid throughput = CAP + CFP throughput

Figure 5.10: Throughput as a function of the traffic load $\eta = \eta_p = \eta_t = 0.1, 0.3, 0.5, 0.7$ with a given probability for generating a non-time critical packet $\eta_d = 0.5, 0.8, 0.9$, superframe and beacon order $BO = SO = 6$, and MAC parameters $m_b = 8$, $m_0 = 3$, $m = 4$, $N = 9$, $h = 500$ and the packet length $L_p = 50bytes$.

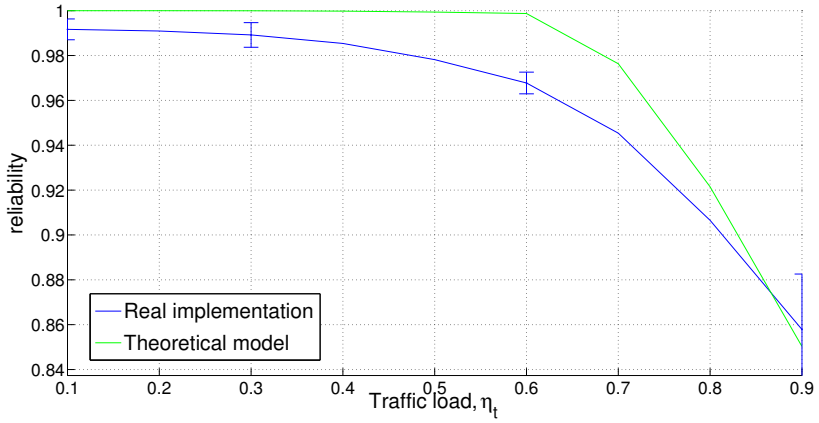


Figure 5.11: Reliability as a function of $\eta = \eta_p = \eta_t = 0.1, 0.3, 0.6, 0.9$ using slotted with a given probability for generating a non-time critical packet $\eta_d = 1$, superframe and beacon order $BO = SO = 6$, and MAC parameters $m_b = 8$, $m_0 = 3$, $m = 4$, $N = 9$, $h = 500$ and the packet length $L_p = 50bytes$.

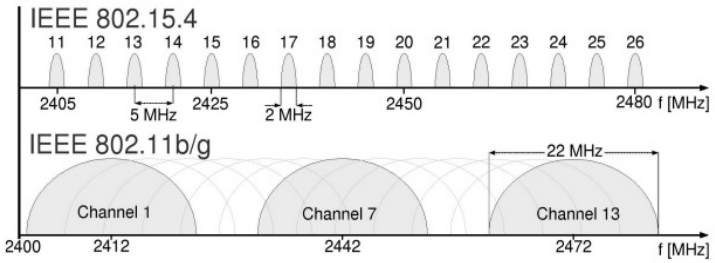


Figure 5.12: IEEE 802.15.4 and 802.11 spectrum usage in the 2.4 GHz ISM band. Source: [16]

Chapter 6

Control applications

In this chapter, we show two practical experiments where the IEEE 802.15.4 applies.

First, we introduce the inverted pendulum process, which we choose as our control application example to run over IEEE 802.15.4. The inverted pendulum involves the control design and the setup of other tools. Furthermore, we also apply our protocol implementation to improve the reliability of a home smart grid [33].

We show how the use of the IEEE 802.15.4 can improve the reliability and gives benefits than a simple radio transmission does not have.

6.1 Inverted pendulum control system

In this chapter, we show the control of single inverted pendulum as an example of tight real-time process. This application example involves communication, a chip software implementation, hardware modifications and a control study.

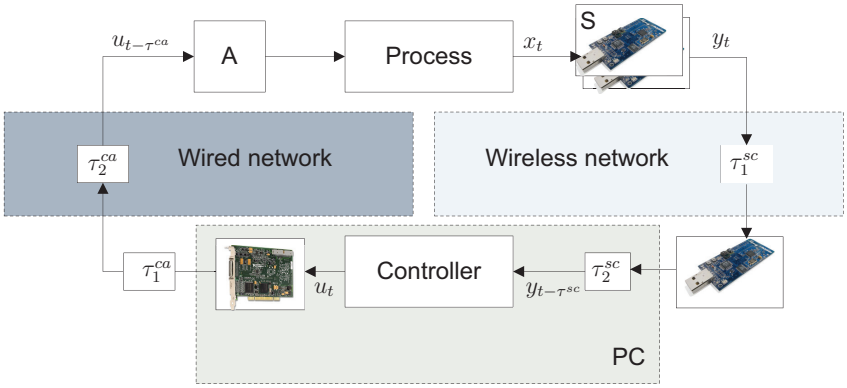


Figure 6.1: Inverted pendulum control system: One computer with NI board, three motes and the inverted pendulum

Figure 6.1 shows the whole wireless inverted pendulum control process. For the sensing, we use two motes transmitting sensor data to the coordinator, which is another mote connected to the PC. For the actuation, we use a Data Acquisition (DAQ) board to send the voltage control output to the motor.

One question that comes up is why we need to use the IEEE 802.15.4 for WPC if it adds more complexity to our motes. We show some important points below:

- **Reliability:** The protocol provides a robustness mechanism to increase the reliability comparing to a simple channel access method like Carrier Sense Multiple Access (CSMA). IEEE 802.15.4 provides a CSMA-CA channel access method with collision avoidance and the possibility of retransmission. In this tight real-time process, it is really important receive all the packets or the pendulum could fall.

- **Adaptation:** With the PAN coordinator role in the network, in case of failure, a device could synchronize again and continue sending packets without any hard-reset.

- **Scalability:** The inclusion of new sensors do not influence in the other, even if the sample rate is really high. In this case, we add two sensors one for the θ angle and one for the cart position. In case of using a simple CSMA, the

use of two sensors affects directly to the reliability, because they would find the channel busy, or would create collisions in the network.

For the pendulum we use the IEEE 802.15.4 with the beacon-enabled mode. Mention that we transmit during the CAP period using slotted CSMA-CA instead of the CFP. In this case, with only two motes transmitting we reach a high reliability and the interferences between each mote, is almost negligible. So it is not needed the use of the GTS mechanism. In case of increasing the number of processes in the same network, I would be necessary the use of the GTS in order to remove the collisions between packets, and assure a high reliability.

To whom is interested in knowing the steps with the problems found in our implementation, Appendix A shows a detailed description. Some videos of the demostration are shows in [24].

6.1.1 Platform and hardware

In this section, we describe the platforms and hardware involved in the wireless inverted pendulum control system. First of all, we see the single inverted pendulum provided be Quanser. Then, we show the counter board that we build to get an accurate resolution from the encoders. Finally the DAQ board and the software to communicate between the PC and the hardware is described.

6.1.1.1 Single Inverted Pendulum

Figure 6.2 shows the inverted pendulum with the modifications, two motes with their counter boards attached to the encoder. The inverted pendulum is provided by Quanser. Quanser offers a guide with the system model. Moreover provides Simulink and Matlab files to run the pendulum using a DAQ board to get the values from the encoders. [35, 36, 37]

The model of the system with the non-linear Equations of Motion (EOM) is characterized with the two equations below:

$$\frac{\partial^2}{\partial t^2} x_c = \left(\frac{\partial^2}{\partial t^2} x_c \right) (x_c, \alpha, F_c) \tag{6.1}$$

$$\frac{\partial^2}{\partial t^2} \alpha = \left(\frac{\partial^2}{\partial t^2} \alpha \right) (x_c, \alpha, F_c) \tag{6.2}$$

$$\tag{6.3}$$

Figure 6.3 show the schematic of the inverted pendulum. [36] explains step by step all the equations of the physical model, in order to solve the partial derivation of the non-linear EOM.

Figure 6.2: Wireless inverted pendulum with the motes and the controller (PC)

$$\begin{aligned}\frac{\partial^2}{\partial t^2}x_c &= \left(\frac{\partial^2}{\partial t^2}x_c\right)(x_c, \alpha, F_c) \\ &= \Big[-\left(I_p + M_p l_p^2\right) B_{eq}\left(\frac{d}{dt}x_c(t)\right) - \left(M_p^2 l_p^3 + I_p M_p l_p\right)\sin(\alpha(t))\left(\frac{d}{dt}x_c(t)\right) - \\ &- M_p l_p \cos(\alpha(t)) B_p \left(\frac{d}{dt}\alpha(t)\right) + \left(I_p + M_p l_p^2\right) F_c + M_p^2 l_p g \cos(\alpha(t))\sin(\alpha(t))\Big] \\ &\Big/ \left[(M_c + M_p)\, I_p + M_c M_p l_p^2 + M_p^2 l_p^2 \sin\left(\alpha(t)\right)^2\right], \end{aligned} \tag{6.4}$$

$$\begin{aligned}\frac{\partial^2}{\partial t^2}\alpha(t) &= \left(\frac{\partial^2}{\partial t^2}\alpha\right)(x_c, \alpha, F_c) \\ &= \Big[(M_c + M_p)\, M_p g l_p \sin(\alpha(t)) - (M_c + M_p)\, B_p \left(\frac{d}{dt}\alpha(t)\right) - \\ &- M_p^2 l_p^2 \sin(\alpha(t))\cos(\alpha(t))\left(\frac{d}{dt}\alpha(t)\right)^2 - M_p l_p \cos(\alpha(t)) B_{eq}\left(\frac{d}{dt}x_c(t)\right) + \\ &+ F_c M_p l_p \cos(\alpha(t))\Big] \Big/ \left[(M_c + M_p)\, I_p + M_c M_p l_p^2 + M_p^2 l_p^2 \sin(\alpha(t))^2\right] \end{aligned} \tag{6.5}$$

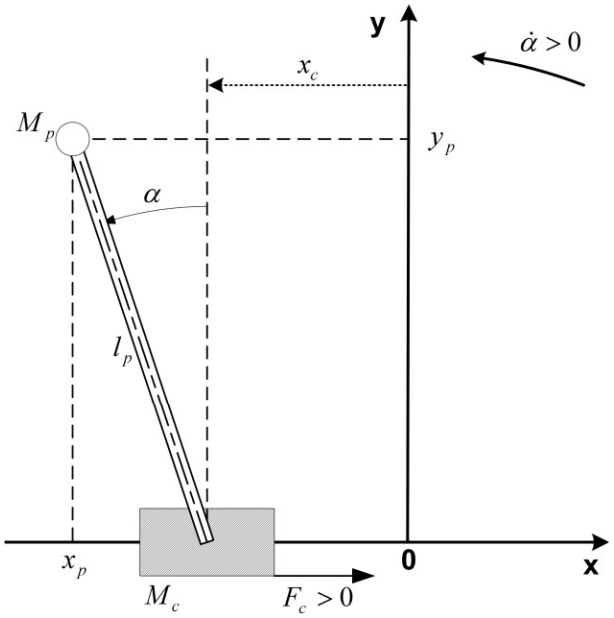


Figure 6.3: Schematic of the single inverted pendulum

In order to design and implement a linear-quadratic regulator (LQR) for our system, a state-space representation of that system needs to be derived. Therefore the EOM should be linearised around a quiescent point of operation. In case of the inverted pendulum, the operating range corresponds to small departure angles, α, from the upright vertical position.

According to the system,

$$\dot{X} = AX + BU, \tag{6.6}$$

where X is the system's state vector, defined as,

$$X^T = \left[x_c(t), \alpha(t), \frac{d}{dt}x_c(t), \frac{d}{dt}\alpha(t)\right], \tag{6.7}$$

and U is linear cart driving force, which is converted to the cart's DC motor voltage.

When we linearise the equations (6.5) and (6.5), the matrix A and B are:

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{gM_p^2 l_p^2}{(M_c+M_p)I_p + M_c M_p l_p^2} & -\frac{B_{eq}(M_p l_p^2 + I_p)}{(M_c+M_p)I_p + M_c M_p l_p^2} & -\frac{M_p l_p B_{eq}}{(M_c+M_p)I_p + M_c M_p l_p^2} \\ 0 & \frac{M_p g l_p (M_c+M_p)}{(M_c+M_p)I_p + M_c M_p l_p^2} & -\frac{M_p l_p B_{eq}}{(M_c+M_p)I_p + M_c M_p l_p^2} & -\frac{(M_c+M_p)B_p}{(M_c+M_p)I_p + M_c M_p l_p^2} \end{bmatrix} \tag{6.8}$$

$$B = \begin{bmatrix} 0 \\ 0 \\ \dfrac{I_p + M_p l_p^2}{(M_c + M_p)I_p + M_p l_p^2 M_c} \\ \dfrac{M_p l_p}{(M_c + M_p)I_p + M_p l_p^2 M_c} \end{bmatrix} \tag{6.9}$$

6.1.1.2 Counter board

LS7366R To be able to use this Serial Peripheral Interface (SPI) external counter and avoid the bias that is present using the mote as a quadrature counter, we needed to implement a complete stack for this chip, as the one existent for CC2420 (radio stack) or STM25P (flash memory). These chips are also using the SPI channel number 0. Hence, it is necessary to build a robust implementation with bus arbitration, otherwise they will try access to the resource at the same time, and the program will crash or act in an unpredictable way.

Content	Directory in the TinyOS-2 Tree for LS7336R
LS7336R chip implementation	
control components	`tos/chips/ls7366r/control`
interfaces	`tos/chips/ls7366r/interfaces`
SPI components	`tos/chips/ls7366r/spi`
test applications	`apps/test/ls7336r`
Platform specific code for TelosB	
ports configuration files	`tos/platforms/telosb/chips/ls7336r`

Table 6.1: TKN15.4 directories in the TinyOS-2 tree for LS7336R chip implementation

Table 6.1 shows the directories where the LS7336R implementation has been placed. Some modifications in `Makerules` and `Makefile` files are necessary to compile the programs successfully.

In the Appendix A, we show the schematics for these boards.

6.1.1.3 Data acquisition

To communicate between the inverted pendulum and the PC, and send the control voltage back to the pendulum motor we use the DAQ board (NI-6221 PCI) provided by National Instruments (NI). [17]

Features

- 16 Analog input (AI). 8 differential or 16 single ended

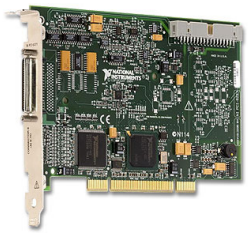


Figure 6.4: NI 6221-PCI board Source: [17]

- Analog-Digital converter (ADC) resolution of 16 bits
- AI with a maximum sampling rate of 250 kS/s and 50 ns of time resolution
- 2 Analog output (AO)
- Digital-Analog converter (DAC) resolution of 16 bits
- Maximum AO rate of 833 kS/S
- AO range $\pm$ 10 V
- 24 digital input/outputs

6.1.1.4 NI LabView

NI LabView has been selected as the software to use the DAQ board to send the voltage to the motor. We implement the control loop using *MathScript RT module* which allows to do the calculations efficiently in LabView. In the Appendix A are screenshots of the VI LabView file.

6.1.2 Control over WSN

Our Networked Control System can be modelled as in Figure 6.5.

The received measurements from the process are subjected with delays and losses. The most important issue in our model is the delay from sensor to the controller τ^{sc}.

For practical experiments, we assume a delay $h > \tau_{cs}$, where $h = 25ms$. In order to have a stable controller even with the delay, it is necessary to estimate our state while compensating for the delay τ^{sc} and then we use a LQR. Note that we follow the notation of [3].

The controller is designed following the methodology in [31]. Figure 6.6 shows the block diagram of our controller.

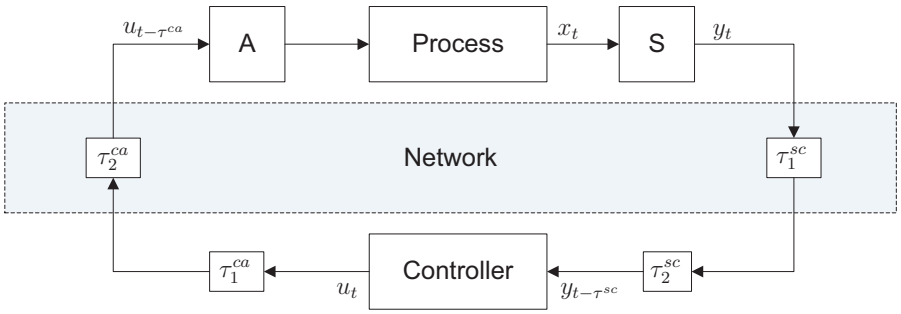


Figure 6.5: Model of our networked control system

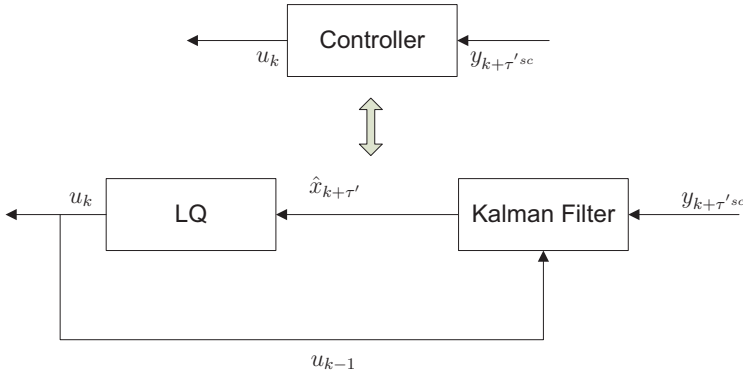


Figure 6.6: Block diagram of the controller

6.1.3 Problem formulation

The process to be controlled is described by the continuous-time model in

$$dx = Axdt + Budt + dv_c, \tag{6.10}$$

where A, B may be time-varying matrices. The process v_c has mean value of zero and uncorrelated increments. $x(t) \in \Re^4$ is the plant state and $u(t) \in \Re$ is the control signal.

A time-varying discrete-time sampled system, with sampling time h can be written as

$$x_{k+1} = \Phi x_k + \Gamma u_k + v_k \tag{6.11}$$

$$y_k = Cx_k + e_k, \tag{6.12}$$

where Φ is the fundamental matrix of (6.10) and v_k and e_k are discrete-time Gaussian white-noise processes with zero-mean value,

$$\Phi = e^{Ah} \tag{6.13}$$

$$\Gamma = \int_0^h e^{As} ds B \tag{6.14}$$

$$R_1 = E[v_k v_k^T] \tag{6.15}$$

$$R_{12} = E[v_k e_k^T] \tag{6.16}$$

$$R_2 = E[e_k e_k^T] \tag{6.17}$$

6.1.3.1 Time-varying Kalman Filters

With a time-varying Kalman filter we estimate the states by computing a new Kalman gain with the received measurements data. The first step is to predict the next value of the states with the knowledge of the previous state and the system model. Optimal Kalman filter equations are given by the equations (6.18) - (6.25).

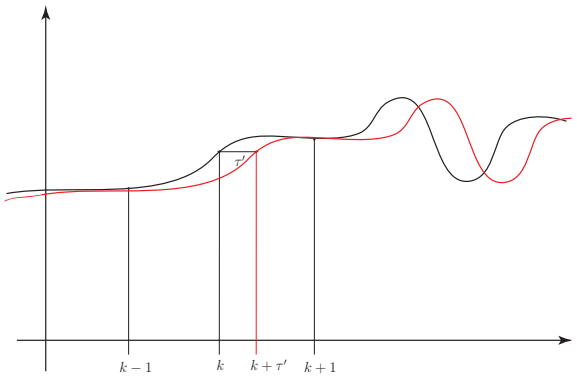


Figure 6.7: Prediction and delay compensation

The Kalman filter could be separated in two steps, the prediction and the correction.

Prediction step In the prediction step, it tries to estimate the next state given the estimation of the present state and the control input sent to the process previously,

$$\hat{x}_{k+\tau'|k} = \Phi_\tau \hat{x}_{k|k} + \Gamma_\tau u_{k-1} \tag{6.18}$$

$$\begin{aligned} \hat{x}_{k+1|k} &= \Phi_h \hat{x}_{k|k} + \Gamma_h u_k \\ &= \Phi_{h-\tau} \hat{x}_{k+\tau'|k} + \Gamma_{h-\tau} u_k \end{aligned} \tag{6.19}$$

$$P_{k+1|k} = \Phi_h P_{k|k} \Phi_h^T + R_1, \tag{6.20}$$

where P_k is defined as the variance of the estimation error,

$$P_k = E\left[\left(\hat{x}_k - E(\hat{x}_k)\right)\left(\hat{x}_k - E(\hat{x}_k)\right)^T\right] \tag{6.21}$$

$$\tau' = \frac{\tau^{sc}}{h} \tag{6.22}$$

In (6.19) we apply a delay compensation because the sample that we get is delayed, and it arrives after the sample time h.

Correction step When a measurement is received the predicted state can be corrected by calculating the new Kalman gain,

$$K_k = P_{k|k-1}C^T\left(CP_{k|k-1}C^T + R_2\right)^{-1} \tag{6.23}$$

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k\left(y_k - C\hat{x}_{k|k-1}\right) \tag{6.24}$$

$$P_{k|k} = P_{k|k-1} - K_k C P_{k|k-1} \tag{6.25}$$

6.1.3.2 Linear Quadratic Regulator

The linear-quadratic (LQ)-control is solved for the case of complete state information. We assume the deterministic case, where $v_k = 0$ and $e_k = 0$. The discrete state-feedback law

$$u_k = -L_k\hat{x}_k. \tag{6.26}$$

It minimizes a discrete cost function equivalent to the continuous cost function

$$J = \lim_{T\to\infty}\frac{1}{2T}\int_{-T}^{T}(\hat{x}_t^T Q\hat{x}_t + u_t^T R u_t)dt. \tag{6.27}$$

We determine the discrete state-space model,

$$\hat{x}_k = A_h\hat{x}_k + B_h u_k \tag{6.28}$$

where,

$$\hat{x}_k = \begin{bmatrix} \hat{x_c} \\ \hat{\theta} \\ \dot{\hat{x}_c} \\ \dot{\hat{\theta}} \end{bmatrix} \tag{6.29}$$

We get control gain by using the `dlqr` function in Matlab.

To summarize, the steps needed in LabView are:

1. Correct the state $\hat{x}_{k|k}$ given the measurements of y_k.
2. Predict the actual state $\hat{x}_{k+\tau|k}$ given the state $\hat{x}_{k|k}$ and the control signal u_k.
3. Predict the next state $\hat{x}_{k+1|k}$ given the state of $\hat{x}_{k+\tau}$ and the control signal $u_{k+\tau}$.
4. Apply the LQR to compute the control signal at $k+\tau$ given the estimation of $\hat{x}_{k+\tau|k}$

$$u_{k+\tau} = -L\hat{x}_{k+\tau|k} \tag{6.30}$$

6.1.4 Performance evaluation

Along this section, we use the process for 2 minutes, and during this time, we have stable period, without any problems and few packets errors, and two different interferences intervals. To make interference on the network we use a *scrambling* mote.

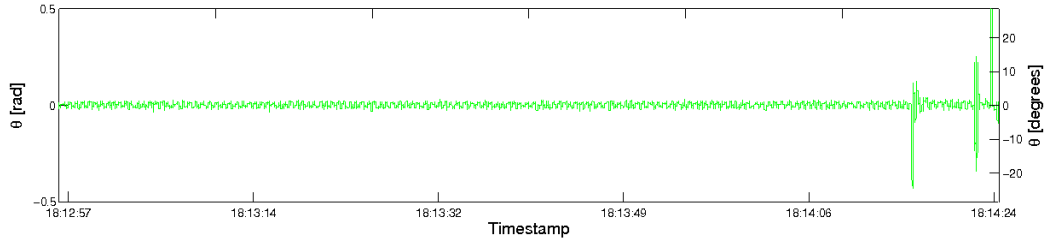
The *scrambling* mote consists in a mote sending dummy data to an non-existent mote using the same channel that the sensors and coordinator are using, but transmitting in a unslotted CSMA-CA. This node tries to simulate high traffic in the network and interferences. Note that with the highest sampling rate, the mote influences in the beacon reception of the motes, making them losing beacons, as we have seen in Section 3.3.2.2.

Below, we show different figures which make it is possible to detect this behaviour from different data, and/or from different points of view. First we see the exported data from LabView, where we see the sensor data and the control output. Then we show different information from the protocol analyser board, where from the time between packets we can detect when we get retransmissions and packet loss. Moreover, once we analyse the data we show the accumulated errors for each node, and the instant of failure.

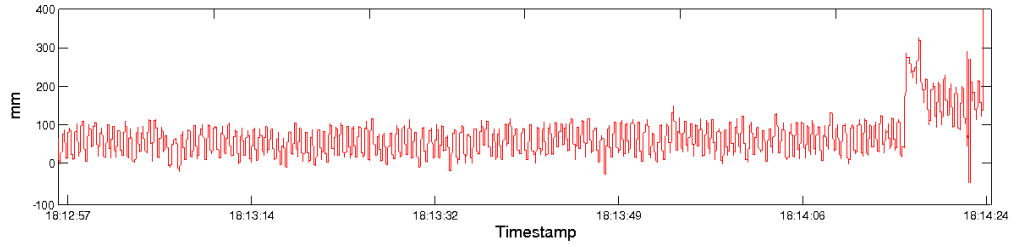
Between the start time at 18:12:57 and 18:13:49 the inverted pendulum process is stable, with few packets loss and few oscillation in the pendulum angle and cart position. After this time, the *scrambling* mote starts sending data with a 100 ms period. It makes the mote number 1, cart position, start failing and not sending packets, making some read from the carts position invalid, but the pendulum is still upright. Once we increase the sampling rate with the *scrambling* mote, at 18:14:25, the mote number 2 starts failing as well, which is a very critical measurement for the control of the pendulum, and the pendulum falls.

6.1.4.1 Sensing values

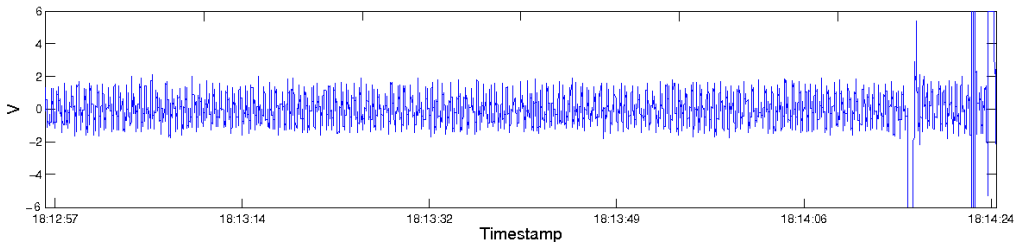
Figure 6.8 shows the information read from the coordinator, when Figure 6.8a shows the pendulum angle θ, and Figure 6.8b the cart position. During the stable position,



(a) Angle θ of the pendulum



(b) Cart position

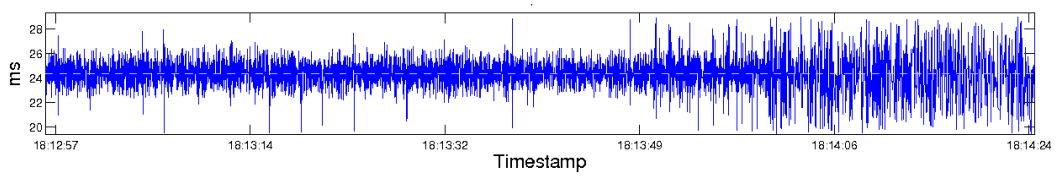


(c) Control output to apply to the cart motor

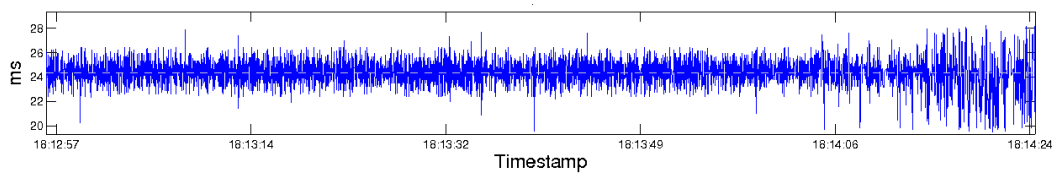
Figure 6.8: Values read from the mote while running the pendulum in a stable position

θ varies around $\pm$ 0.02 rad (1.146 degrees) and the cart position 80 mm. Note that all the figures related to the inverted pendulum have the same timestamps. Hence they are all related although we get ones from LabView and other from the protocol analyser.

Observe that the cart position values fail around 18:14:15 after 24 seconds of low rate scrambling data. In the same instant, Figure 6.8c, the control voltage changes. The new voltage levels made the cart move faster, so the pendulum angle oscillations that we get are different than the stable position. After that, the pendulum comes to the stable position again for 5 seconds, and at this moment, after a lot of packets errors from a mote, the pendulum falls and it does not come back on track again. Here we can measure that packet losses critically affect the pendulum, mainly packet losses in node 2 which transmit critical packets with the pendulum angle.



(a) Time between packets for mote ID=1 (cart position)



(b) Time between packets for mote ID=2 (pendulum angle)

Figure 6.9: Time between packets for motes 1 and 2

6.1.4.2 Time between packets

Figure 6.9 shows the time between sent packets for each node. Observe a delay jitter of 3 ms with a mean value of 25 ms. The packets under the mean value and the jitter are due to retransmissions and the packets above are packets loss. Therefore, in Figure 6.9a we observe problems in the transmission from 18:13:50 but in Figure 6.9b the errors start around 18:14:10.

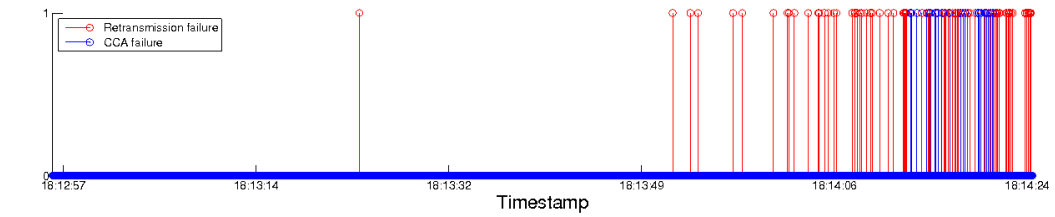
The difference between the instant 18:13:49 and 18:14:06 is the position of the scrambling mote. During these times, the scrambling mote is moved from mote ID=1 towards mote ID=2.

6.1.4.3 Reliability and probability of error

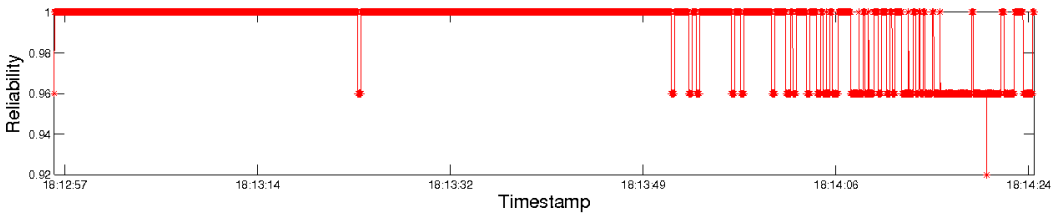
In Figure 6.10 we show the probability of error and reliability. Figure 6.10a shows the instants of errors differentiating between the errors coming from a CCA failure and retransmission failure. Mainly we have errors due to a retransmission failure, meaning that the influence of the clock drift, saw in Section 3.3.2.2, does not influence too much and they are transmitting in different backoff slots. In Figure 6.10b, we see the equivalent plot, providing the reliability with a window of 20 packets. So each point in the plot, represents the reliability for 20 packets.

6.1.4.4 Accumulated errors

Figure 6.11 shows the evolution and behaviour of the error packets, the number of accumulated packet errors. Until 18:13:50 it is almost constant. Then it increase slowly until 18:14:15 and finally it increases really fast. It is easy to show with that



(a) Instant of failure



(b) Reliability with sliding window

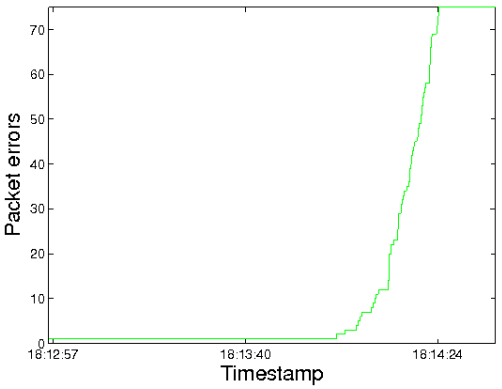
Figure 6.10: Comparative between reliability, or probability of failure and the control output send by the PC

figure the influence of the *scrambling* mote, for both motes. The node 1, in Figure 6.11a has more errors because the *scrambling* mote is close to him than the node 2. Notice that even with a huge number of error between 18:13:40 and 18:14:24, is not until 18:14:24 that the pendulum does not fail. Again, we can notice the criticality of node 2.

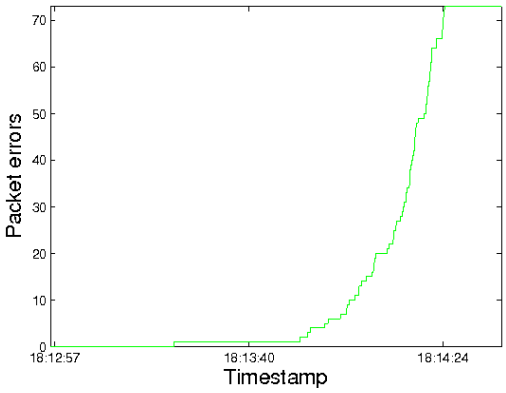
To summarize, in this chapter we have seen the indispensable tools and platforms needed for the wireless inverted pendulum. The delay and the tight real-time requirements of the process, have forced to use a robust control design. The Kalman filter with the delay compensator has been the pillar of the control, with this estimator the influence of the delay has been relieved. From now onward, we could see the IEEE 802.15.4 as the standard for wireless process control tight real-time process control.

6.2 Home smart grid

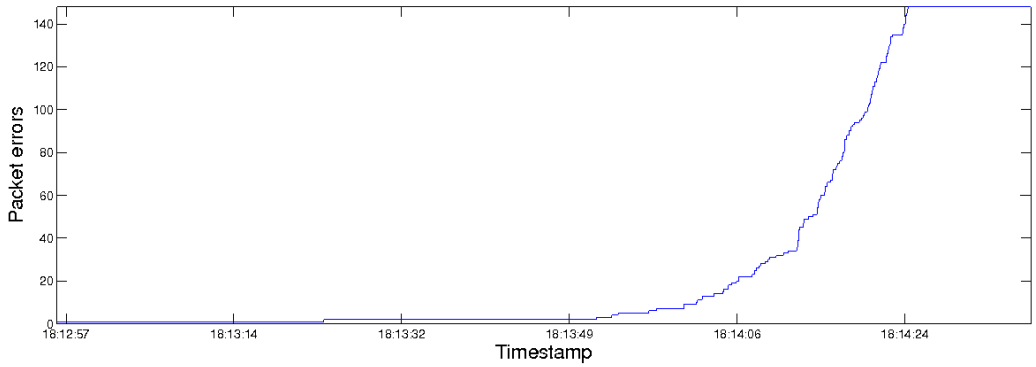
Another application where the IEEE 802.15.4 would give benefits is the Home smart grid. Where a large deployment is needed and the packets losses have to be reduced.



(a) Packets errors accumulation for mote ID=1 (Cart position)



(b) Packets errors accumulation for mote ID=2 (Pendulum angle)



(c) Total packets errors accumulation

Figure 6.11: Packets errors accumulation while running the pendulum in a stable position

6.2.1 Introduction

In this work, a system to detect water and electricity consumption is introduced to identify resource waste and inform the consumers about it. For this purpose, many sensors are deployed in the kitchen of the department in order to gather data of the consumption. These sensors are an alternative to other ways of getting this information that are usually considered invasive, like a camera. Due to the amount of sensors involved and the large area covered, this application is highly suitable to the use of IEEE 802.15.4 to gather data from all the sensors.[33]

Our contribution in this project is the adaptation of the communication. By default, the TinyOS uses CSMA as a channel access method, which does not give any collision avoidance or any ACK mechanism. In order to provide more a robustness to the application, we adapt the programs to use the IEEE 802.15.4.

Figure 6.12 shows the communication diagram in this deployment The topology

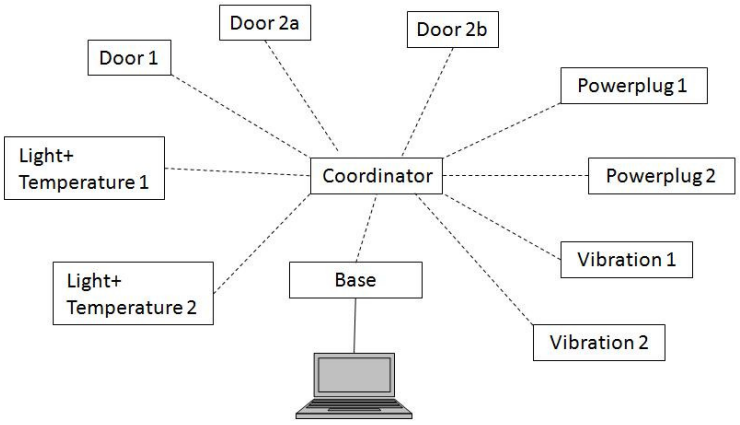


Figure 6.12: Communication in the wireless sensor network Source: [33]

used is a start network where we have some RFD devices, which are the sensors, a FFD who makes the connection between the PC and the PAN coordinator. The coordinator is situated in the middle of the kitchen, some sensors around it, and a base station connected to the PC in order to log the date coming from the base station. The coordinator in case of the IEEE 802.15.4, actuates as the PAN coordinator sending the beacon frame periodically, and as a bridge between the network and the PC, where it retransmits all the packets received from the sensors to the base station.

6.2.2 Performance evaluation

6.2.2.1 Reliability

In this application, a low sample rate is used, so comparing with results showed in Chapter 5, we expect a very high reliability.

The analysis done in this section are based on the sniffer data. As far as it is an external device, that does not play any role on the network, the results that are shown in this section are not accurate, because the protocol analyser could not receive all the packets.

CSMA shows the reliability using the default channel access method implemented in TinyOS. When the sender is ready to transmit data, it checks if the physical medium is busy. It senses the medium continually until it becomes idle, and the it transmits the frame. The collision are not detected by the sender, so it assumes that the packet has received without problems.

TDMA shows the reliability using the default radio communication implemented in TinyOS with a previous scheduled times. It is not a strict Time Division

Multiple Access (TDMA), because we do not have any time slots mechanism, but the sender try to transmit in a different instant of time with this protocol. It provides a synchronization mechanism in order to have this behaviour.

IEEE 802.15.4 shows the reliability in a beacon-enabled transmitting during the CAP. As we have seen, we transmit using CSMA-CA, a modification of CSMA where Collision Avoidance (CA) is used to improve the performance of CSMA by attempting to keep the less busy the channel. If the channel is sensed busy before transmission then the transmission is deferred for a random interval.

Sensor	**Reliability** [%] CSMA	TDMA	CSMA-CA 802.15.4
Powerplug 1	94.00	91.03	99.91
Powerplug 2	95.81	99.45	99.98
Vibration fridge	95.92	99.44	99.91
Vibration sink	84.19	96.20	100.00
Light sensor 1	95.75	99.43	99.98
Light sensor 2	94.56	98.61	100.00
Temperature sensor 1	95.14	98.85	99.98
Temperature sensor 2	94.19	98.71	100.00
IR 1	95.68	99.50	99.98
IR 2	64.47	98.72	99.98
IR 3	95.29	99.47	100.00
Total	92.26 %	97.53 %	99.97 %

Table 6.2: Reliability for the different communications algorithms and protocols. Source: [33]

Table 6.2 shows the reliability for the different sensors deployed in the kitchen. In the columns we have the different types of communications. The first column (CSMA) has the worst reliability because there are no retransmission and no collision detection, where the motes are not aligned each other. In the second column (TDMA) we have the motes aligned and transmitting in different instants of time, so the are no collision between each other, but as far as it is not a strict TDMA and no ACK mechanism and no retransmission, in case of failure they do not retransmit the packet. Therefore, in the IEEE 802.15.4 transmitting using CSMA-CA during the CAP, we have a collision detections with a ACK mechanism and retransmission, which give the highest reliability.

Figure 6.13 shows the number of accumulated error that we get during one hour of running the application.

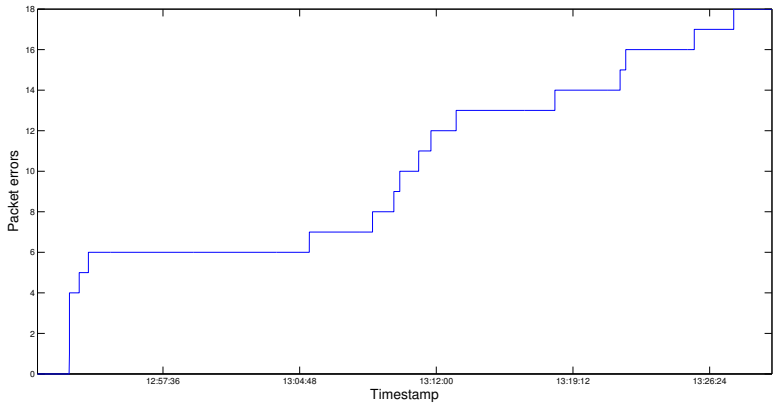


Figure 6.13: Accumulated errors in Home Smart Grid application

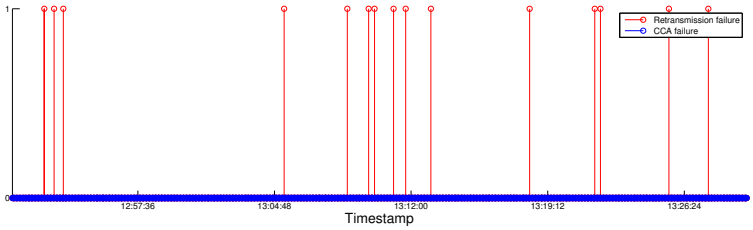


Figure 6.14: Instant of packet failure

Figure 6.14 shows the instant of time where we get a failure, where it could be due to (1) CCA failure or (2) Retx failure.

Chapter 7

Conclusion and future work

7.1 Conclusion

In this thesis, we describe the necessary steps to reach the wireless inverted pendulum process. First of all, we have described briefly the IEEE 802.15.4 in order to know the benefits and problems that the protocol provides.

The first critical point was the selection of a protocol implementation that provides us a protocol as close as possible to the standard defined in [26]. A comparison between the two main implementations is done through the experiments to validate the feasibility of the implementations. The IPP implementation [9], against our thoughts, had critical problems. It is impermissible that the motes stop working after some time and the timing for the beacons were not as good as it was possible given the motes used. The TKN15.4 provides a solid implementation but it was not completely implemented. For this reason we expand the functionalities of the implementation and we provide the necessary mechanisms to transmit during the CFP period. These mechanisms involves: GTS allocation, GTS deallocation, GTS expiration and GTS reallocation.

Dealing with a real implementation, we face challenges related to the technological limitations of the platforms under use. All of these present more challenges that had to be mitigated to enable the experimental validation and a released implementation. During this particular implementation and experimental efforts, some of those difficulties were re-encountered, namely in what concerns the behaviour of the Tmote Sky/TelosB motes: (i) *Memory constraints* (ii) *Hardware platforms and debugging* (iii) *CC2420 transceiver limitations* (iv) *Timing and synchronization requirements* (v) *TinyOS task scheduler* .

On the other hand, with a IEEE 802.15.4 implementation we provided a evaluation with which the behaviour is proven. The performance evaluation shows which is

the real behaviour of our implementation for different network scenarios, in terms of packet deliver loss and delay. For the slotted version, we analysed the characteristics and performance transmitting during the CAP and CFP. Analysing the reliability and delay as a function of the MAC parameters, we have seen the importance of the *macMinBE* respect to *macMaxCSMABackoffs* and *macMaxFrameRetries* in term of average delay. It is also important to mention that with the number retransmissions n we increase the reliability, but for a given n, it saturates and keep constant. An analysis for the hybrid MAC was also performed.

Finally, with the IEEE 802.15.4 working and with the performance evaluation done that assures that the protocol implementation gives a proper performance, it was the moment to continue and focus on the main goal of this thesis, prove and show the benefits in wireless process control that the standard would give. We use the process with tight real-time requirements available in our lab, a single inverted. We prove that is possible to use IEEE 802.15.4 in a tight real-time control system, opening the possibility to prove different researches in a practical environment.

Another application were the IEEE 802.15.4 provides large benefits is in Home Smart Grid deployment. Using the protocol in the deployment of [33], the reliability increases around 7 % against a simple CSMA transmission mechanism.

7.2 Future work

Along this thesis, we have seen different chapters with its own "Future work": GTS implementation in Section 4.4 (1) Performance evaluation in Section 5.3.Moreover we posed some discussion in the GTS implementation in Section 4.3.

For the IEEE 802.15.4 implementation, we have seen some issues that does not allow to have a standard compliant implementation. Some improvements are related to software and hardware. As soon as the new MSP430 series have been launched and the Zolertia company has the motes with this MCU, the acquisition of those and future modification of the clock will reduce the present hardware limitations. Furthermore, in the software development is needed an extensive debugging in order to conform every part of the standard and improve the performance of the code.

Having a full standard compliant implementation, a comprehensive performance evaluation and comparison with the theoretical model would give us more information and the certification that the implementation is correct. In papers [30, 29, 28] exist a complete and extensive description of the theoretical model with its characteristics to compare with.

There are some researches where hybrid MAC is applied for control applications. With a IEEE 802.15.4 implemented and the possibility to transmit during the CAP and CFP, we have opened the door to apply different control mechanism, event-driven, hybrid control, etc. [20]. To see projects related to our lab visit [24].

References

[1] A. Alemdar and M. Ibnkahla. Wireless sensor networks: Applications and challenges. *Signal Processing and Its Applications, 2007. ISSPA 2007. 9th International Symposium on*, pages 1–6, feb. 2007.

[2] L. Angrisani, M. Bertocco, G. Gamba, and A. Sona. Effects of RSSI impairments on IEEE 802.15.4 wireless devices performance susceptibility to interference. *Electromagnetic Compatibility - EMC Europe, 2008 International Symposium on*, pages 1–6, sep. 2008.

[3] Karl J. Åström and Björn Wittenmark. *Computer-controlled systems: theory and design (3rd ed.)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1997. ISBN 0-13-314899-8.

[4] CU Boulder. MultimodAl NeTworks of In-situ Sensors. Technical report. URL `http://mantisos.org/`. Accessed September 2010.

[5] Chipcon. CC2420 datashet. Technical report. URL `http://inst.eecs.berkeley.edu/~cs150/Documents/CC2420.pdf`. Accessed September 2010.

[6] Intel Corporation. Intel mote 2 engineering platform. Technical Report Rev. 2.0. URL `http://ubi.cs.washington.edu/files/imote2/docs/imote2-ds-rev2.0.pdf`. Accessed September 2010.

[7] Moteiv Corporation. Tmote sky data sheet. Technical report, San Francisco, June 2006. URL `http://www.bandwavetech.com/download/tmote-sky-datasheet.pdf`. Accessed August 2010.

[8] Crossbow. MICAz datasheet. Technical report. URL `www.openautomation.net/uploadsproductos/micaz_datasheet.pdf`. Accessed September 2010.

[9] André Cunha, Mário Alves, and Anis Koubâ. IPP Hurray!: An IEEE 802.15.4 protocol implementation (in nesc/tinyos): Reference guide v1.2. Technical Report TR-061106, ISEP-IPP, May 2007.

[10] Ricardo Augusto Rodrigues da Silva Severino. On the use of IEEE 802.15.4/Zigbee for Time-Sensitive Wireless Sensor Network Applications . Master's

thesis, ISEP, October 2008. URL http://www.cooperating-objects.eu/fileadmin/dissemination/2009-thesis-award/severino.pdf. Accessed September 2010.

[11] Adam Dunkels. Contiki - the operating system for connecting the next billion devices - the internet of things. Technical report, 2010. URL http://www.sics.se/contiki/. Accessed September 2010.

[12] Sinem Coleri Ergen. ZigBee/IEEE 802.15.4 Summary, September 2004.

[13] David Gay, Philip Levis, Robert von Behren, Matt Welsh, Eric Brewer, and David Culler. nesc 1.1 language reference manual. Technical report, May 2003.

[14] David Gay, Philip Levis, Robert von Behren, Matt Welsh, Eric Brewer, and David Culler. The nesc language: A holistic approach to networked embedded systems. volume Proceedings of Programming Language Design and Implementation (PLDI), jun. 2003.

[15] Jan-Hinrich Hauer and Adam Wolisz. TKN15.4: An IEEE 802.15.4 MAC Implementation for TinyOS 2. Technical report, Technical University Berlin - Telecommunication Networks Group, March 2009. URL http://www.tkn.tu-berlin.de/publications/papers/TKN154.pdf. Accessed September 2010.

[16] Jan-Hinrich Hauer, Adam Wolisz, and Vlado Handziski. Experimental Study of the Impact of WLAN Interference on IEEE 802.15.4 Body Area Networks . *Proc. of 6th European Conference on Wireless Sensor Networks (EWSN)*, February 2009.

[17] National Instruments. Ni pci-6221. Technical report, 2010. URL http://sine.ni.com/nips/cds/view/p/lang/en/nid/14132. Accessed September 2010.

[18] Texas Instruments. Smartrf packet sniffer - user manual. Technical Report Rev. 1.10. URL http://focus.ti.com/docs/toolsw/folders/print/smartrftm-studio.html. Accessed September 2010.

[19] IPP. OpenSource Toolset for IEEE 802.15.4 and Zigbee. Technical report, 2010. URL http://open-zb.net/. Accessed September 2010.

[20] José Araújo, Yassine Ariba, Pangun Park, and K. H. Johansson. Control Over a Hybrid MAC Wireless Network . Technical report.

[21] Philip Levis. Tinyos programming. Technical Report 1.3, October 2006. URL http://www.tinyos.net/tinyos-2.x/doc/pdf/tinyos-programming.pdf. Accessed September 2010.

[22] MAXIM. MAX232 - Datasheet. Technical report, 2006. URL http://datasheets.maxim-ic.com/en/ds/MAX220-MAX249.pdf. Accessed September 2010.

[23] UCLA NESL. SOS Embeddeed operating system. Technical report. URL `https://projects.nesl.ucla.edu/public/sos-2x/doc/`. Accessed September 2010.

[24] KTH | NetCon group. URL `http://www2.ee.kth.se/web_page/netcon/`. Accessed September 2010.

[25] Zolertia | Z1. URL `http://www.zolertia.com/products/Z1`. Accessed September 2010.

[26] IEEE Std 802.15.4-2996, September, Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs) , September 2006. URL `http://standards.ieee.org/getieee802/download/802.15.4-2006.pdf`. Accessed September 2010.

[27] Pangun Park. *Protocol Design for Control Applications using Wireless Sensor Networks*. PhD thesis, KTH, Stockholm, 2009. ISBN 978-91-7415-441-5.

[28] Pangun Park, C. Fischione, and K.H. Johansson. Performance Analysis of GTS Allocation in Beacon Enabled IEEE 802.15.4. *Sensor, Mesh and Ad Hoc Communications and Networks, 2009. SECON '09. 6th Annual IEEE Communications Society Conference on*, pages 1–9, jun. 2009.

[29] Pangun Park, Karl H. Johansson, and Carlo Fischione. Performance analysis of hybrid medium access control in IEEE 802.15.4 protocol. *Submitted*, September 2010.

[30] Pangun Park, Piergiuseppe Di Marco, Pablo Soldati, Carlo Fischione, and Karl H. Johansson. A generalized markov chain model for effective analysis of slotted IEEE 802.15.4. *Mobile Adhoc and Sensor Systems, 2009. MASS '09. IEEE 6th International Conference on*, pages 130–139, oct. 2009.

[31] Joonas Pesonen. Stochastic estimation and control over wirelessHART networks: Theory and implementation. Master's thesis, KTH, February 2010.

[32] M. Petrova, Lili Wu, P. Mahonen, and J. Riihijarvi. Interference Measurements on Performance Degradation between Colocated IEEE 802.11g/n and IEEE 802.15.4 Networks. *Networking, 2007. ICN '07. Sixth International Conference on*, pages 93–93, apr. 2007.

[33] Oriol Prats. Smart office: Wireless sensor network for energy monitoring and user profiling. Master's thesis, KTH | Electrical Engineering, August 2010.

[34] D. Puccinelli and M. Haenggi. Wireless sensor networks: applications and challenges of ubiquitous sensing. *Circuits and Systems Magazine, IEEE*, 5(3): 19–31, 2005. ISSN 1531-636X.

[35] Quanser. *Single Inverted Pendulum - Instructor Manual.*

[36] Quanser. *Single Inverted Pendulum - Sudent handout.*

[37] Quanser. *Single Inverted Pendulum - User Manual.*

[38] B. Rubio, M. Diaz, and J.M. Troya. Programming Approaches and Challenges for Wireless Sensor Networks. *Systems and Networks Communications, 2007. ICSNC 2007. Second International Conference on*, pages 36–36, aug. 2007.

[39] P. Suriyachai, U. Roedig, and A. Scott. Implementation of a MAC protocol for QoS support in wireless sensor networks. *Pervasive Computing and Communications, 2009. PerCom 2009. IEEE International Conference on*, pages 1–6, mar. 2009.

[40] Systems, LSI Computer. LS7366R - Quadrature counter with serial interface. Technical report. URL `http://www.lsicsi.com/pdfs/Data_Sheets/LS7366R.pdf`. Accessed September 2010.

[41] Crossbow Technology. Crossbow telosb. Technical report, San Jose, California. URL `http://www.willow.co.uk/TelosB_Datasheet.pdf`. Accessed August 2010.

[42] TinyOS. TEPs. Technical report. URL `http://docs.tinyos.net/index.php/TEPs`. Accessed September 2010.

[43] TinyOS. Tinyos. Technical report, 2010. URL `http://www.tinyos.net/`. Accessed September 2010.

[44] A. Willig. Wireless sensor networks: concept, challenges and approaches. *e & i Elektrotechnik und Informationstechnik*, 123:224–231, 2006. ISSN 0932-383X. URL `http://dx.doi.org/10.1007/s00502-006-0351-1`. 10.1007/s00502-006-0351-1.

[45] A. Willig. Recent and emerging topics in wireless industrial communications: A selection. *Industrial Informatics, IEEE Transactions on*, 4(2):102–124, May 2008. ISSN 1551-3203.

[46] A. Willig, K. Matheus, and A. Wolisz. Wireless technology in industrial networks. *Proceedings of the IEEE*, 93(6):1130–1151, jun. 2005. ISSN 0018-9219.

Appendices

Appendix A

Inverted pendulum

Nomenclature

Symbol	Description
x_c	Cart linear positions
$\frac{d}{dt}x_c$	Cart linear velocity
α	Pendulum angle from the upright position
$\frac{d}{dt}\alpha$	Cart linear velocity
F_c	Cart driving force produced by the motor
I_p	Pendulum moment of inertia
M_p	Pendulum mass
l_p	Pendulum length from pivot to the centre of gravity
B_{eq}	Equivalent viscous damping coefficient for cart
B_p	Equivalent viscous coefficient for pendulum
M_c	Lumped mass of the cart system, including the Rotor Inertia
X	State vector
A	State-Space Matrix
B	State-Space Matrix
U	Control signal

Table A.1: Model nomenclature

Steps for our approach

We think that it is important to show the evolution and different steps that we made to have the inverted pendulum sensing through wireless and the IEEE 802.15.4. It could be a good section to read to whom want to reach such a challenge with its own.

1. **Simulink with serial blocks:** Quanser provides a large number of files with documentation, maple files with the model, matlab files with the control design and a simulink schema to run the application with its hardware or any DAC board that simulink supports, like ours.

   To be able to get the sensing data from the BaseStation mote instead of the DAC board, it is needed the communication between mote and PC through serial (Universal Serial Bus (USB)). But due to the use of a Real-time environment in Simulink, it was impossible to apply and use the serial blocks.

2. **Simulink with S-functions** Our second try was the implementation of functions to read the serial port. We had tried to program functions in C and C++, and include them as S-functions, but Simulink did not allow use the USB in any way.

   As we can see we tried some options with Matlab, because we think that it was the best option to apply control process, reducing delays, easy way to implement transfer functions, matrix multiplications,.... but it was necessary to give it up in order to reach our challenge.

3. **LabView:** The next option available, once we did not find any options with Simulink, was LabView. LabView is a programs focus in the communication between external hardware and the PC. For our approach, with a high sample rate, we were quite sceptical about its behaviour but it was the unique option in mind. The control toolbox for LabView had some timing problems and it was not an option for our applications. Therefore, we continue with the MathScript toolbox which allows us to implement the control manually.

   When LabView was running the LQR control and controlling the pendulum in real-time, we realized that there was a huge bias between the real value read from the NI board and the value that we are receiving with the BaseStation. Moreover, there was a delay between the samples. It was around 50 ms, time that would crash the pendulum because it is a tight process.

   So, in this step we realized that we got a bias. This bias is due to the no task priority in the TinyOS. Although it exists some mechanism of pre-emption like task, asynchronous and synchronous functions, it was not enough to read and listen all the events coming from the DAC.

   The delay introduced on the samples it could be introduced by the communication between mote and PC, the buffers on the operating system, the

communication between OS and LabView, or the time to process the serial data in LabView. But we can exclude the time between the mote and PC because we know that the delay introduced by the mote is around 5 ms.

4. **Labview and LS7366R to avoid bias** As soon as we realized that there were nothing to improve in the mote application, the only way to continue dealing with the pendulum was the introduction of an external counter. It will liberate the mote and we will get the exact value.

   The counter selected was a quadrature counter with SPI communication, LS7366R [40], which provides us a wide settings and different counter configurations, and an easy communication with the mote.

   Finally with that external counter, we completely remove the bias but the delay problem was still there. For this, we tried to use the RS232 communication instead of using the USB.

5. **LabView and LS7366R and RS232:** To test the RS232, we get the data from the input of the FDMI driver, pin 25, in the mote and apply this signal to a MAX232 to converter the signal levels to a RS232 standard ones [22, "Applications examples"]. But the delay accomplished with the RS232 was the same as the USB, and the packets loss increased.

6. **LabView and LS7366R with Kalman filter:** Following with our delay deal, we came up on the idea of estimate the state of the process and compensate the delay. For this we applied a Kalman filter with delay compensation which allowed us to predict the next state of the plant and moreover manage the delay. So, applying the estimated states to the LQR we were able to have the inverted pendulum stable for long time.

Counter board

Figure A.1 shows the schematic of the circuit needed for the SIP external counter. The components are: (a) 3.3 Voltage regulator, FAN2504 (b) Operational Amplifier (OPAM) LM324 (c) The SPI counter LS7366 (d) Crystal oscillator, CMACKD 20.00 (e) Capacitors and resistors . The connector at the bottom (U13) is the expansion pack in the mote.

LabView

Figure A.4 shows the time loop used read the values from the BaseStation. Actually, we read the values from the Transport Control Protocol (TCP) port, where we are

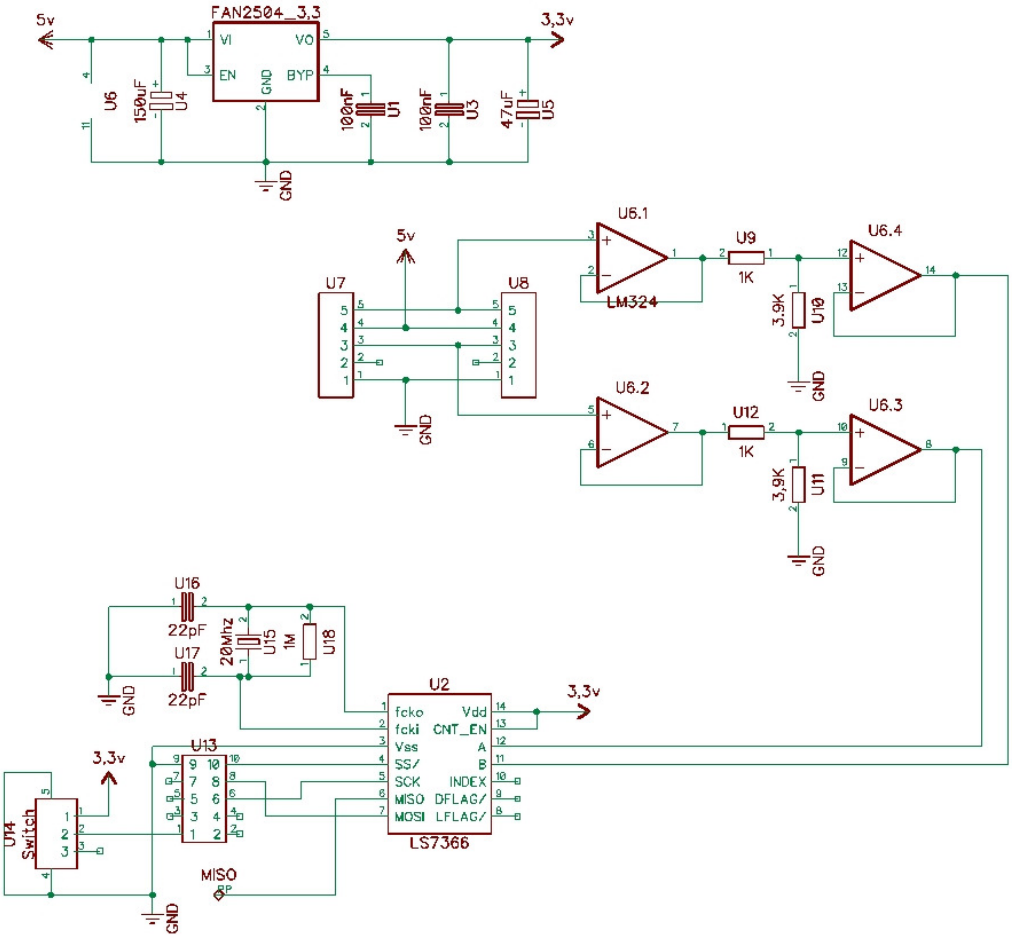


Figure A.1: Schematic of the circuit

running a Serial Forwarder. Serial Forwarder check the Frame Check Sequence (FCS) and sends to LabView the correct packets.

Figure A.5 shows the time loop used for compute the control output. On the left, we see the reads of the motes, as a local variables, in the middle the *MathScript node* where the control in calculated, and on the right we see the *DAQ assistant* block used to configure the DAQ board and send the output voltage back to the motor. We also used a *Write measurements* block in order to be able to save the data and compare them with the sniffer data.

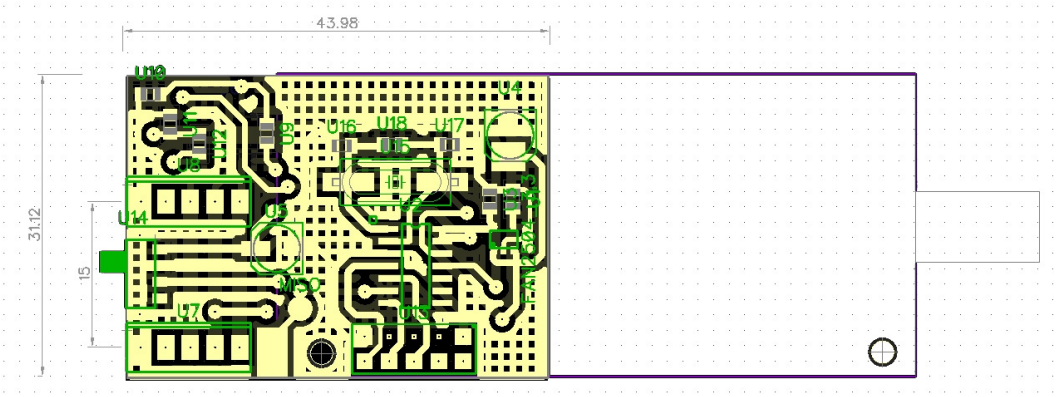
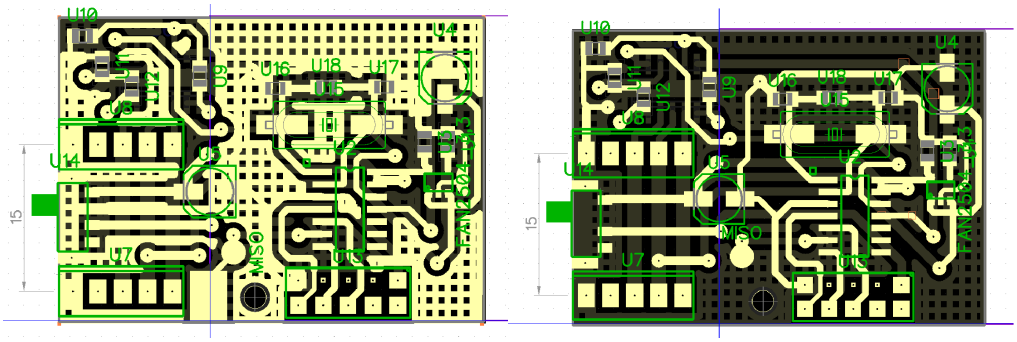


Figure A.2: General layout with the board and the mote



(a) **(b)**

Figure A.3: Detailed layout with the board

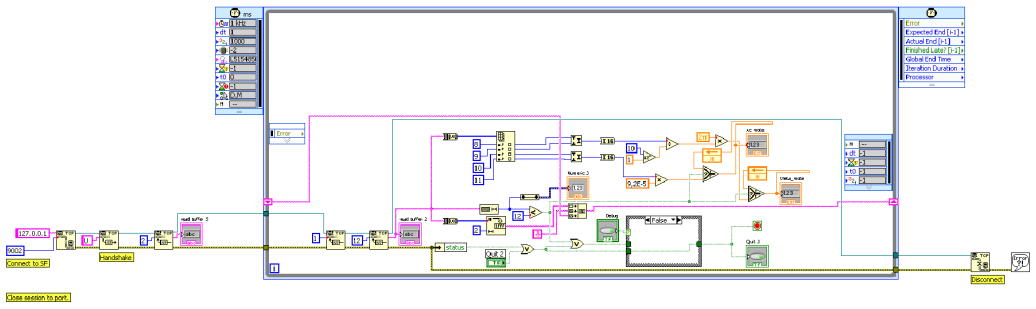


Figure A.4: Screenshot of the VI file from LabView with the time loop to read the values from the BaseStation

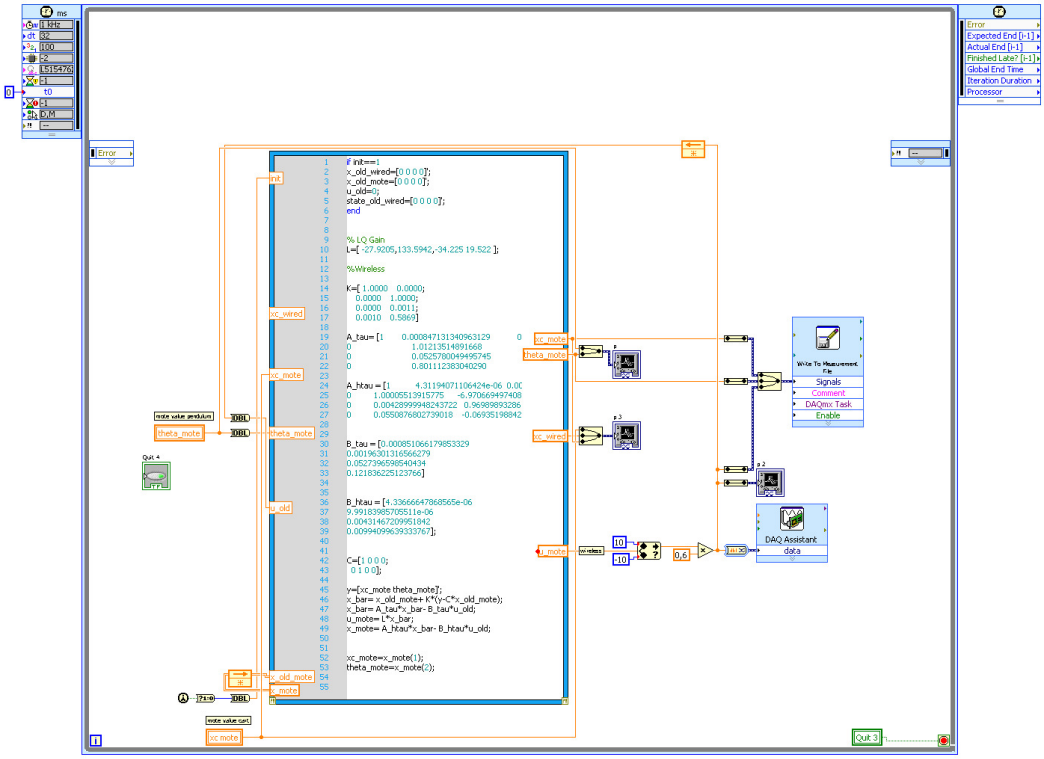


Figure A.5: Screenshot of the VI file from LabView with the time loop for control