Universidade de Aveiro

2017

Departamento de Eletrónica, Telecomunicações e Informática

André Marques Temprilho

Pastor Virtual

Virtual Shepherd

Universidade de Aveiro

2017

Departamento de Eletrónica, Telecomunicações e Informática

André Marques Temprilho

Pastor Virtual

Virtual Shepherd

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Eletrónica e Telecomunicações, realizada sob a orientação científica do Professor Doutor Paulo Bacelar Reis Pedreiras do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro e do Professor Doutor Pedro Alexandre Sousa Gonçalves da Escola Superior de Tecnologia e Gestão de Águeda

Este trabalho é financiado pelo Fundo Europeu de Desenvolvimento Regional (FEDER), através do Programa Operacional Competitividade e Internacionalização (COMPETE 2020) do Portugal 2020
[Projeto SHEEPiT com o nº 017640 (POCI-01-0247-FEDER-017640)

Cofinanciado por:

COMPETE 2020

PORTUGAL 2020

UNIÃO EUROPEIA
Fundo Europeu de Desenvolvimento Regional

Dedico este trabalho aos meus pais.

Agradeço-lhes por todo o carinho, pela paciência e por toda a confiança que sempre depositaram em mim.

Acredito que sem vós, este percurso que agora termina não teria começado.

Obrigado Pai.
Obrigado Mãe.

o júri

presidente	**Professor Doutor Alexandre Manuel Moutela Nunes da Mota** Professor Associado, Universidade de Aveiro
arguente	**Professor Doutor Frederico Miguel do Céu Marques dos Santos** Professor Adjunto, Instituto Superior de Engenharia de Coimbra
coorientador	**Professor Doutor Pedro Alexandre Sousa Gonçalves** Professor Adjunto, Universidade de Aveiro

agradecimentos

São várias as pessoas que merecem um forte agradecimento da minha parte pela ajuda ou motivação que me prestaram no desenrolar deste trabalho.

Começo por agradecer aos meus orientadores, professor Paulo Pedreiras e professor Pedro Gonçalves, por toda a dedicação a este projeto e disponibilidade prestados. Mais do que orientadores, fizeram parte do meu grupo de colegas pela sua solidariedade e à vontade com que me receberam.

Ainda no âmbito do trabalho, não podia deixar de mencionar o nosso colega Luís Miguel Nóbrega. Agradeço-lhe pelo acompanhamento que fez das várias fases do projeto e por toda a sua disponibilidade prestada.

Agradeço a todos os meus amigos pelos momentos de descompressão e lazer. Recarregar energias foi fundamental.

Por último, quero agradecer à minha família. Agradeço-lhes por toda a motivação nos momentos de maior desânimo e por toda a confiança que sempre tiveram em mim.

A todos vós,
Um muito obrigado!

palavras-chave

Redes de sensores, internet das coisas, protocolos de comunicação wireless, monitorização animal

resumo

A remoção de ervas e outras espécies vegetais em propriedades agrícolas é um trabalho árduo que precisa de ser repetido periodicamente. Recorre-se habitualmente a máquinas agrícolas e herbicidas para o efeito, o que não só é dispendioso como também levanta preocupações de cariz ambiental. A utilização de gado na remoção das espécies vegetais indesejadas é um método já testado e que permite não só a diminuição do uso de herbicidas como também dos fertilizantes artificiais. No entanto, pelo facto dos animais tendencialmente também se alimentarem de algumas espécies cultivadas, impossibilita a sua utilização durante todo o ano de cultivo. O projeto SheepIT pretende criar uma solução para este problema através de um sistema de monitorização animal que, de forma autónoma, corrige comportamentos indesejados, tais como, quando os animais se alimentam das espécies de cultivo. Aliado ao facto de permitir a recolha de dados acerca do comportamento dos animais e da sua localização, o sistema é também uma ferramenta de gestão do gado e do seu bem-estar.

Neste trabalho é apresentada uma rede de sensores (WSN) com vista à monitorização de um rebanho de ovelhas. O protótipo desenvolvido permite a obtenção de dados em tempo real provenientes dos sensores alojados em coleiras para uso em ovelhas domésticas. São utilizadas tecnologias rádio que operam na banda dos 433 MHz. O sistema de comunicações é baseado num protocolo proprietário desenvolvido especificamente no âmbito deste projeto e cuja arquitetura é apresentada nesta dissertação.

keywords Wireless sensor networks, internet of things, wireless communication protocols, animal monitoring

abstract

Weed control in agriculture is a toilsome job that needs to be periodically repeated. Agricultural machinery and herbicides are frequently employed in this task, but these methods are expensive and raise environmental concerns. Livestock can take this job by feeding on with the undesirable species and reduces the usage of herbicides and artificial fertilizers. However, because these animals can also feed on some crops, grazing livestock is not a feasible alternative method for weed control because it can’t be used through the entire year. The SheepIT project aims to solve this problem, by providing an autonomous system for animal monitoring that corrects undesirable behaviors, such as when animals are feeding on crops. The system should also enable data gathering about the animals’ behavior and their localization, which turns out to be a livestock management and welfare control tool.

This dissertation presents a wireless sensor network (WSN) to monitor a flock of domestic sheep. The implemented prototype enables data gathering in real time from sheep-borne collars. Wireless communications use a radio link in the 433 MHz band. A proprietary protocol was specifically developed for this project and its architecture is presented.

Table of Contents

Table of Figures

Table of Tables

List of Equations

Table of Abbreviations

BS	Beacon Synchro (packet)
BPREP	Beacon Pairing Reply (packet)
BPREQ	Beacon Pairing Request (packet)
B2B	Beacon-to-Beacon (packet)
CN	Collar Notification (data structure)
CPREP	Collar Pairing Reply (packet)
CPREQ	Collar Pairing Request (packet)
C2B	Collar-to-Beacon (packet)
CSMA	Carrier Sense Multiple Access
DMA	Direct Memory Access
GW	Guarding Window
IoT	Internet of Things
MAC	Medium Access Control
MC	Macro-cycle
RSSI	Received Signal Strength Indicator
SoC	System-on-a-chip
SW	Synchronization Window
TAW	Turn-around Window
TDMA	Time Division Multiple Access
$\mathbf{TT_{RX}}$	Time-to-receive
$\mathbf{TT_{TX}}$	Time-to-transmit
VTW	Variable Traffic-type Window
WSN	Wireless Sensor Network
µC	Micro-cycle

Chapter 1

Introduction

This dissertation was conducted under the SheepIT project [1], [2], which seeks an innovative solution for weeding vegetable species based on an IoT architecture while being environmentally friendly and cost effective. The system relies on grazing sheep to overcome the weed species problem in vineyards, which implies monitoring of the flock to prevent them from feeding from fruits and branches.

1.1 Motivation

The presence of weeds in vineyards constitute a threat to plantations because they compete for soil nutrients and water with the vines. Land owners are therefore enforced to invest in methods to remove the undesirable weeds. Weeding methods based on herbicides and machinery equipment are still common [3], but environmental concerns have arisen due to the regular usage of chemicals. Moreover, the need to repeat the application of these methods several times over the year, renders them expensive [4].

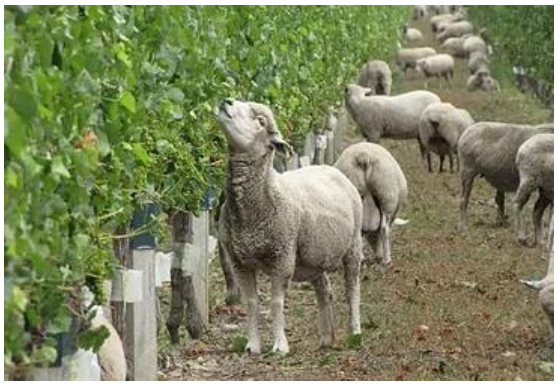


Figure 1.1 – Grazing sheep in a Portuguese vineyard [5]

Ecological and less expensive approaches have been tried with livestock [6]. These animals naturally feed on with the undesirable weeds and can fertilize the soil. However, they can also feed on grapes and on the lower branches of the vines (Figure 1.1), impacting on the productions. This problem limits the usage of the method to the period before the first production grapes start to emerge. The presence of shepherds to monitor the behavior of several hundreds of sheep is also not feasible.

To overcome the aforementioned problem, the SheepIT Project proposes to develop an autonomous system capable of monitoring and correct the sheep posture [7] while they are weeding the vineyards. By also enabling virtual fencing techniques, animals shall be kept in desirable locations without the need of expensive electrical fences.

1.2 Objectives

The objective of this dissertation was to develop and deploy the foundations of a wireless sensor network (WSN) to be integrated in the SheepIT Project. Concerns about the energy efficiency were taken to make the system a viable solution. The desire of monitoring several hundreds of sheep through the usage of wirelessly connected sensor nodes imposed communication issues that were addressed. These objectives can be summarized as follows:

- Study currently available solutions for animal tracking and monitoring systems;
- Study WSN protocols that could fit the proposal SheepIT solution;
- Design and deploy a WSN that meets the requirements of the project.

1.3 Structure

The remainder part of this document is organized as follows:

- **Chapter 2**: presents the State of the Art regarding WSN protocols and animal monitoring systems;
- **Chapter 3**: presents the overall SheepIT architecture, with focus on the WSN's MAC layer;
- **Chapter 4**: presents the implemented node's state-machines to accomplish the MAC layer architecture;
- **Chapter 5**: presents some experimental results as well as some theoretical computations about the MAC layer;
- **Chapter 6**: final remarks are taken about the whole dissertation and future work is proposed.

Chapter 2

State of the Art

This chapter presents the State of the Art regarding wireless sensor networks (WSN) and animal monitoring systems. Firstly, the WSN term is introduced and some relevant protocols in this field are presented. Following this, projects related with remote animal monitoring systems found on literature are discussed.

2.1 WSN protocols and architectures

Wireless Sensor Networks (WSN) have a wide variety of definitions. They can be considered as a set of individual nodes with environment sensing and control capabilities that through wireless communications cooperate to fulfill a common task [8]. The applications of these networks can range from medicine and healthcare products to wildfire detection systems or household surveillance, just to name a few. Despite this diversity, it is possible to classify WSNs according to the type of sensing [8]:

- **Event detection**: in this type of WSN, nodes report the occurrence of an event, like the presence of smoke in a room;

- **Periodic sensing**: in this kind of WSN, nodes periodically measure and report environment parameters, like the temperature of a room.

Whatever is the type of sensing, nodes are always addressed to report, at some time, information to a so called *sink* – a node targeted to receive the sensor nodes data [9]. The design of a WSN architecture has, therefore, to deal with these communications through a common link, which can be thought as a resource that has to be shared by the nodes. These issues are addressed in the Medium Access Control (MAC) layer of the network.

The challenges imposed in the design of a MAC layer are dependent on its application type. Some may be more targeted into energy efficiency [10] if the nodes are required to run on small batteries. For other applications this problem was minimized by managing to power the nodes by other means, like wireless charging, enabling more complex communications criteria to be addressed [11]. These different challenges led to the development of many *MAC protocols*, some very specifically targeted to unique applications while others had a more generic utilization goal. Nevertheless, all implement mechanisms to enable the nodes access to the communication channel in order to cope with minimize or even eliminate *packet collisions* – when two or more protocol messages (packets) are transmitted at the same time in a shared medium causing their degradation.

Based on the medium access mechanisms, MAC protocols can be classified in the following two categories [8]:

- **Contention-based protocols**: in these protocols, nodes compete for the medium without prior knowledge about their neighbor's access times, which can have a random nature. To reduce the probability of packet collisions, techniques based on *Carrier Sense Multiple Access* (CSMA) [12] are used.

- **Schedule-based protocols**: as opposed to contention-based, schedule-based protocols require the nodes to schedule their transmission times. This demands some type of synchronization to perform a *Time Division Multiple Access* (TDMA) [13] scheme.

The former type is usually designated to event detection networks, due to the random nature of the communications. The last type is more suitable for the periodic sensing networks or when the traffic is so dense that CSMA techniques are not able to handle the rise of packet collisions. It suffers however from increased latency due to the schedule nature of the communications. Nevertheless, scheduled-based protocols may still require a setup-phase in which data is exchanged in a contention-based manner, before a proper access time schedule can be created. Other protocols still continue to support both types of traffic after the setup-phase, giving rise to more flexible traffic management that some literature calls **hybrid protocols** [14]. In these protocols, the non-scheduled traffic is exchanged in a dedicated time frame named *contention window*. Some authors also classify the protocols as *synchronous* or *asynchronous* [9] depending on the existence or absence of a synchronization scheme to govern the nodes communications.

2.1.1 Energy efficiency in WSN

As mentioned before, some WSN applications don't require the nodes to run on batteries, minimizing the impact of the network's architecture energy efficiency. Notwithstanding, this dissertation proposes a solution based on portable nodes, thus, this parameter is an important requirement that should be studied.

A key component of a WSN node is its *transceiver*, enabling the transmission and reception of radio packets. This component is usually responsible for most of the energy consumption in a typical WSN node as shown in Figure 2.1, accounting for around 40 % of the overall energy consumption.

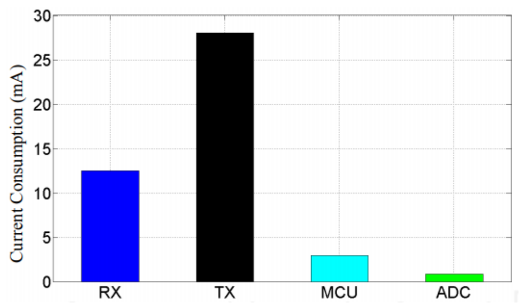


Figure 2.1 – Current consumption in a typical WSN node [15]

To minimize the energy consumption, manufacturers enable less energy demanding states, usually referenced as *idle* or *sleep*.

The main goal in a MAC layer design related with the energy efficiency is to take advantage of these low power consumption states without compromising the overall communications performance. Authors call these protocols, which take the approach of cycling through different power states, **duty-cycled protocols** [16]. Moreover, the prominent use of the low power consumption states over the others gave rise to the term *low duty-cycle protocols*. However, some issues can still degrade the energy efficiency if not properly addressed [8]: packet collisions, overhearing, idle listening and overhead.

Packet Collisions

Packet collisions lead to energy waste since the transmitted data becomes corrupted. Upon that, packet re-transmissions are frequently asked to be performed when collisions are detected.

This problem is minimized through the usage of either CSMA or TDMA techniques. In some contexts, when CSMA is opted, control packets may also be required. This is especially true in multi-hop networks where the *hidden terminal* problem is well documented [17].

Overhearing

The term *overhearing* relates with the reception of packets by nodes that are not intended to do so. An unwanted packet reception is obviously a waste of energy and avoiding it might not be a simple solution in a wireless medium. Packets with recipient fields do not minimize overhearing because nodes are still required to decode that field, thus, required to receive the packet even if it is not addressed to them.

A solution to minimize this problem is a scheduled based communication scheme in the sense that nodes should be aware when the traffic of their interest is scheduled to be exchanged. This enables nodes to keep in lower power consumption states for the rest of the time.

Idle listening

Idle listening differs from overhearing in the sense that the former one relates with the transceiver being ready to receive a packet even though it was not necessary, while overhearing refers to receive a packet not intended to. Although receiving a packet requires more energy than being able to receive one (being "*awake*"), idle listening still incurs in energy waste that can be minimized using scheduled transmissions.

Protocol overhead

Greater protocol headers or trailers incur in longer packets, thus more energy is required to transmit and receive them. Moreover, some architectures rely in *handshake* mechanisms to ensure that transmitted data reaches the recipient. These handshakes require the exchange of control packets, like the RTS / CTS [18], also increasing the protocol overhead.

In the following section, a survey of MAC protocols targeted to WSN that took different approaches to deal with some of the aforementioned problems is presented.

2.1.2 WSN protocols survey

A set of representative protocols found in literature, targeted to WSN applications, is surveyed in this section. They are organized in contention-based, scheduled-based (or hybrid) categories according to its MAC layer policies. Much more protocols besides the ones that are presented could be found on literature. The criterion used in the selection of the protocols being presented relates with the novelty they have introduced regarding the medium access, which in some cases forms the basis of the MAC layer of other protocols.

Contention-based protocols

- **CSMA-PS – Preamble-based protocol**

Most of the wireless contention-based protocols take advantage of a technique called *preamble-sampling* to avoid packet collisions. It consists in sending a preamble prior to the data transmission. By detecting the preamble, neighbor nodes recognize that the medium is busy, abort their transmissions (if there were any to be performed) and prepare to receive the packet.

This technique was combined with the old ALOHA protocol [19] by El-Hoyidi giving rise to the CSMA-Preamble Sampling (CSMA-PS) protocol [20]. Receiver nodes periodically *wake-up* from low power states and set their transceivers to sense the medium. If a preamble is detected they stay in RX mode, otherwise go back to sleep. A drawback of this technique is that the duration of the preamble must be longer than the wake-up periods to guarantee that recipient nodes are listening to the preamble. Moreover, this incurs in overhearing because nodes are required to listen to the end of the preamble even if the data packet is not addressed to them. Figure 2.2 shows an example of a medium access based on CSMA-PS with acknowledgement.

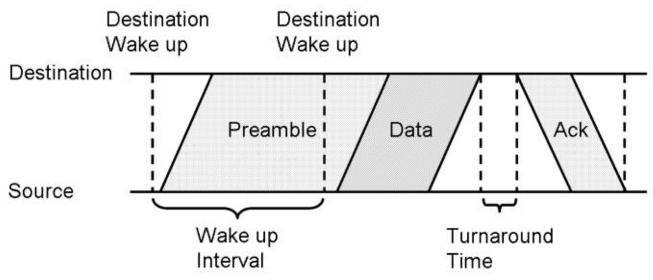


Figure 2.2 – CSMA-PS with acknowledgement [21]

- **X-MAC – Short-preambles**

The overhearing problem of CSMA-PS is reduced by sending shortened preambles and using them to address the data packets' recipients. By doing it, receivers are only required to decode one of those short preambles, but idle listening still occurs between preamble receptions. This MAC scheme was introduced by Michael Buettner *et al.* in the X-MAC protocol [22], exemplified in Figure 2.3

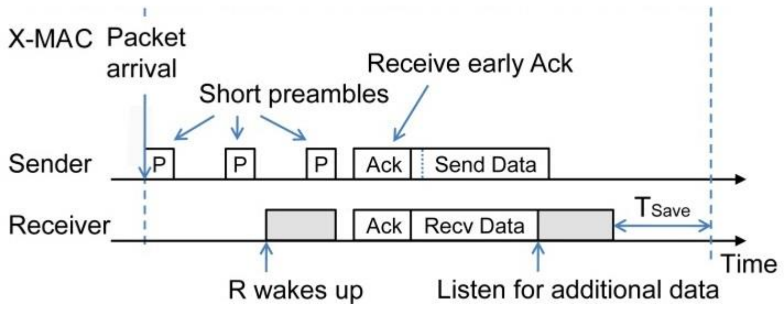


Figure 2.3 - X-MAC medium access example [21]

Additionally to decreasing the overhearing, X-MAC also minimizes the medium access delay by enabling the sender node to start the data packet transmission as soon as the recipient acknowledges one preamble. Moreover, the receiver is asked to stay in RX after the data packet reception to be able to quickly respond to another sender's preamble without delaying this action by its turn-around time (delay caused by switching from RX to TX and *vice-versa*).

- **BP-MAC – Preamble sensing with additional random backoff**

The ability of protocols that only rely in medium sensing to avoid packet collisions is directly affected by the network's traffic load and by the transceivers *Clear Channel Assessment* (CCA) delay and *turn-around time* (TT). The CCA delay states how much time the transceiver takes to detect a busy medium while the TT is the required time to switch between RX and TX modes. A node will not be able to detect a preamble if its transmission starts within an interval shorter than CCA and TT [21]. This limitation increases the probability of packet collisions as the traffic raises.

A. Von Bodisco *et al.* faced this issue with his proposal *Backoff Preamble MAC* protocol (BP-MAC) [23]. Following the example of Figure 2.4, an explanation of the MAC policy is given.

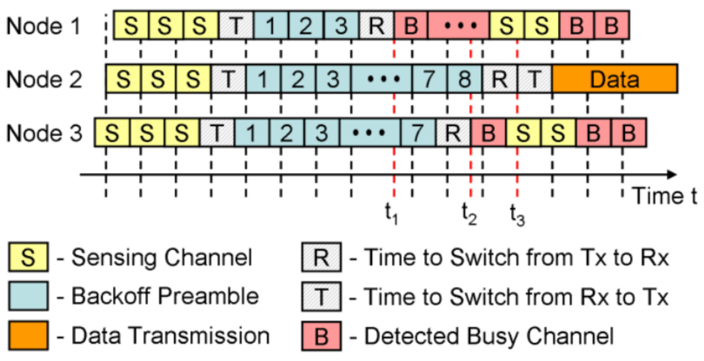

Node 1
S S S T 1 2 3 R B ••• S S B B
Node 2
S S S T 1 2 3 ••• 7 8 R T Data
Node 3
S S S T 1 2 3 ••• 7 R B S S B B
Time t
t_1
t_2
t_3
S - Sensing Channel
- Backoff Preamble
- Data Transmission
R - Time to Switch from Tx to Rx
T - Time to Switch from Rx to Tx
B - Detected Busy Channel

Figure 2.4 – BP-MAC medium access example [23]

The BP-MAC protocol divides the medium access period in time slots equal or larger than the CCA delay or TT (the largest of them). When a node wants to send data, first listens to the medium for a duration of three slots. If it founds the medium is free, it switches the transceiver to TX (one additional slot) and proceeds to send a preamble (*backoff preamble*) with a duration equal to a random number of time slots. After transmitting the preamble, the node switches back to RX (one more slot) and listens again to the channel. The node can only continue to the data packet transmission if the medium is still idle (one more slot to toggle back to TX), otherwise the medium access procedure shall be restarted. To conserve some energy, when an ongoing transmission is acknowledged the node may switch off its transceiver and re-attempt the medium access procedure after a number of slots greater than one. In the above figure's example, because all nodes took different preamble durations and were required to re-sense the medium after their preamble transmissions, a packet collision was avoided even though they both have sensed a free medium in first place almost at the same time (a difference shorter than one CCA or TT).

BP-MAC reduces the collision probability in a contention-based architecture, but collisions still can occur if nodes choose the same preamble duration and start their transmissions at the same time. Moreover, the backoff preambles incur in protocol overhead.

Scheduled-based (or hybrid) protocols

- **S-MAC – contention-based with synchronized duty-cycle**

Sensor-MAC (S-MAC) is a hybrid protocol that implements a periodic wake-up strategy to conserve energy [24] and it still is one of the most cited protocols for WSN. The transceivers' duty-cycle is divided in a "*listen*" and a "*sleep*" period whose lengths are the same for every node (Figure 2.5). The former one is also used to transmit packets and is divided in three sub-frames: SYNC, RTS and CTS.

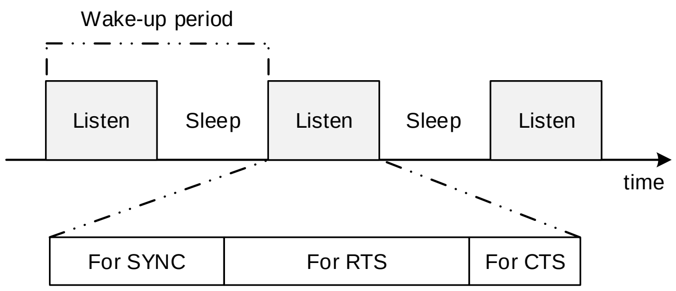


Figure 2.5 – Medium access frame division for an S-MAC node (adapted from [8], [25])

S-MAC attempts to synchronize the wake-up periods of neighbor nodes forming a *virtual cluster*. For that matter, nodes periodically send synchronization packets in their own SYNC sub-frames to inform other nodes about their own *schedule*, that is, when are they going to sleep and wake-up again.

A node that doesn't have a schedule yet, listens for a complete *synchronization period* in order to receive at least one SYNC packet from one of its neighbors. This period can be longer than the wake-up period because nodes are not required to send SYNC packets in every listen phase. If a SYNC packet is received, the node becomes a *follower* of the announced schedule, otherwise it randomly picks a schedule for itself, broadcasts it and becomes a *synchronizer*. The protocol takes no provisions against nodes who learn more than one schedule. This can happen when a node is in the *border* of two clusters. The existence of border nodes can be, at some point, desirable to enable the broadcasting of packets from one cluster to another. However, if they are required to follow more than one schedule their sleeping times will be downsized.

The SYNC packets are exchanged in a contention-based manner with additional backoff. During the RTS frame, nodes listen for RTS packets (*Request to Send*) from their neighbors and respond to them in the CTS frame (*Clear to Send*) also using a contention-based scheme. Nodes that have successfully exchanged RTS/CTS packets in the previous frames, proceed to the data packets exchange during the sleep frame, only falling into their sleep schedules after the complete data exchange. The data traffic uses a CSMA scheme with ACK.

Packet collisions, idle listening and overhearing are reduced using NAV (*Network*

Allocation Vector). Each packet contains a duration field, so when a node receives a packet that is not addressed to it, it records its duration time in a variable named NAV. Therefore, it knows for how long the medium continues to be busy and so it can avoid listen to the medium to perform carrier sense during that time. The major drawback of S-MAC is its difficulty to adapt to load changes, since the frames have fixed size, plus, the issues related with multiple clusters are not well defined. The use of RTS, CTS and ACK control packets incurs in protocol overhead.

- **Z-MAC – dynamic switching between CSMA and TDMA**

The difficulties in adapt S-MAC to varying load conditions are suppressed with the *Zebra-MAC* (Z-MAC), an hybrid protocol that tries to take advantage of CSMA under low contention and TDMA when the traffic load increases [26]. The energy consumption efficiency is also addressed using a low duty-cycle approach with a sleep and an active period that need to be synchronized amongst neighbor nodes.

The goal of this protocol is to give nodes, who were assigned a TDMA slot, the ability to contend for other nodes' slots. Under favourable conditions (low traffic density), the latency is expected to be drastically reduced, in contrast to a pure TDMA scheme. On the other hand, when traffic becomes denser, nodes will not be able to successfully contend for other nodes' slots and will use the slot that they were given to. This means that the protocol can dynamically adapt between a contention-based and a schedule-based scheme according to the load conditions.

The medium access, following the Z-MAC architecture goes as follows:

1. All nodes are assigned with a TDMA slot that it is granted to be unique under a two-hop distance (so the same slot can be re-used by nodes who aren't in the same radio interference zone). The node who gets the TDMA slot is called the *owner* while the others are the *nonowners* of that slot.

2. The *time frame* is synchronized, so the first slot of the active period (*slot 0*) starts at the same time in every node. Moreover, the *maximum slot number* (MS) is known to everyone.

3. Nodes work under a *low contention level* mode (LCL) by default, meaning that they can compete for each other's slots to transmit their data packets. A node operates in a *high contention level* mode (HCL) only when it receives an *explicit contention notification* (ECN) from a neighbor. This message informs that nodes must only use their own slots or contend for slots that are free within their radio interference zone.

4. Under a LCL operation, when nodes want to transmit data they compete for a slot using CSMA with backoff. The owner of the current slot has priority over it. This

is achieved by forcing the nonowners to backoff a random time superior to the slot's owner when contending for it.

5. Nodes measure the *noise level* to seek for high contention levels. When the noise is higher than a threshold, the HCL mode is triggered and the node sends an ECN message to its neighbors.

The Z-MAC requires a complex setup phase in which nodes discover their neighbors and TDMA slots are assigned using DRAND, a distributed algorithm for TDMA scheduling [27]. When a node joins a network, on the run, DRAND is locally performed on that node but the outcome may change the maximum slot number and this needs to be propagated across all the network. Moreover, the reduction of latency by recycling TDMA slots, makes it difficult to adapt the protocol to applications in which the nodes don't have a fixed position through time.

- **IEEE 802.15.4 – a standard for WSN**

The *IEEE 802.15.4* is a communication standard for low data rate wireless personal area networks (WPAN), provided by the *Institute of Electrical and Electronics Engineers* (IEEE) [28]. The protocol was unleashed in 2003 [29] and continues to receive amendments [30].

The standard covers both the PHY (physical) and MAC layer. It supports a total of 16 channels when operating with a 250-kbps data rate in the 2.4 GHz ISM band (2.4 – 2.485 GHz, with 5 MHz of spacing between carriers). Other configurations are also permitted: 20 kbps with 1 channel (868 – 868.6 MHz) and 40 kbps with 10 channels (905 – 928 MHz). The channel usage is limited to 1 channel at a time.

Respecting the MAC layer, the protocol uses both contention-based and schedule-based schemes. It can also be classified as asymmetric in the sense that different types of nodes are subject to different roles [8].

Two types of nodes are distinguished by the MAC layer: *reduced function devices* (**RFD**) and *full function devices* (**FFD**). The former ones are usually more constrained, either in terms of energy consumption or computational power, and thus are not required to perform all the tasks that can be assigned to the later ones. While the RFD's only operate as **devices** (namely, simple nodes), FFD's can play the role of **coordinators** and **PAN coordinators** (Personal Area Network manager). A device can only communicate via a coordinator. The protocol either supports a *star topology* (with devices connected to a single PAN coordinator) or a *peer-to-peer topology* (with some coordinators forwarding packets as routers) [31].

The medium access can either be performed in *beaconed mode* (also referred as *slotted CSMA*) or in *non-beaconed mode* (*unslotted CSMA*). The decision over which mode the network should operate is also one of the PAN coordinators task. The two MAC operation modes are described next.

802.15.4 in beaconed mode

The coordinator manages its devices channel access based on a super-frame (Figure 2.6). It follows the duty-cycle approach, with active and inactive periods to allow devices (and the coordinator itself) to turn-on and off their transceivers. These periods are synchronized with a *beacon frame* – a periodic coordinator message sent in the beginning of each super-frame.

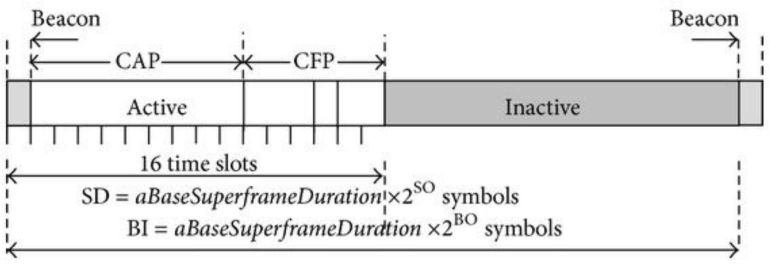

Beacon
CAP
CFP
Beacon
Active
Inactive
16 time slots
SD = *aBaseSuperframeDuration* ×2^{SO} symbols
BI = *aBaseSuperframeDurdtion* ×2^{BO} symbols

Figure 2.6 – IEEE 802.15.4 super-frame structure [32]

The active period is in turn divided in a total of 16 time slots, being the beacon frame the first. The remaining slots are grouped in two windows – the *contention access period* (CAP) and the *contention free period* (CFP). In the first group, slots are disputed with random backoff amongst devices that have data to send, whereas in the last group slots can be granted (*granted time slots* - GTS) to devices upon previous request to the coordinator. The CSMA algorithm executed in CAP, requires the nodes to detect an idle channel during a certain number of consecutive times (namely, *backoff periods*) prior to assume that they have won the contention.

The coordinator must be active during the entire active period. Devices must wake-up prior to the beacon-frame and only stay active during CAP if they have data to *upload* to the coordinator or if the coordinator has addressed them (in the beacon-frame) to prepare to receive a packet. During CFP only devices that have been assigned a GTS wake-up in their slot (either to receive or transmit).

802.15.4 in non-beaconed mode (contention-based)

In a non-beaconed mode, there is no super-frame governing the devices communications. The latency can be downsized at the expense of forcing coordinators to stay active during all time. On the other hand, since there is no synchronization, devices are free to choose their own sleep schedules. The medium's access is provided with a simpler CSMA mechanism in which a single CCA (clear channel assessment) is performed when the medium is found idle.

Researchers have found that the non-beaconed mode has better performance respecting throughput [33]. Yet, the 802.15.4 standard does not take provisions against the hidden terminal problem when contention takes place [34], [35].

Some protocols were built on top of the 802.15.4 standard, namely, Zigbee® [36], WirelessHART [37], 6LoWPAN [38], [39] and Queue-MAC [14], [40], just to name a few.

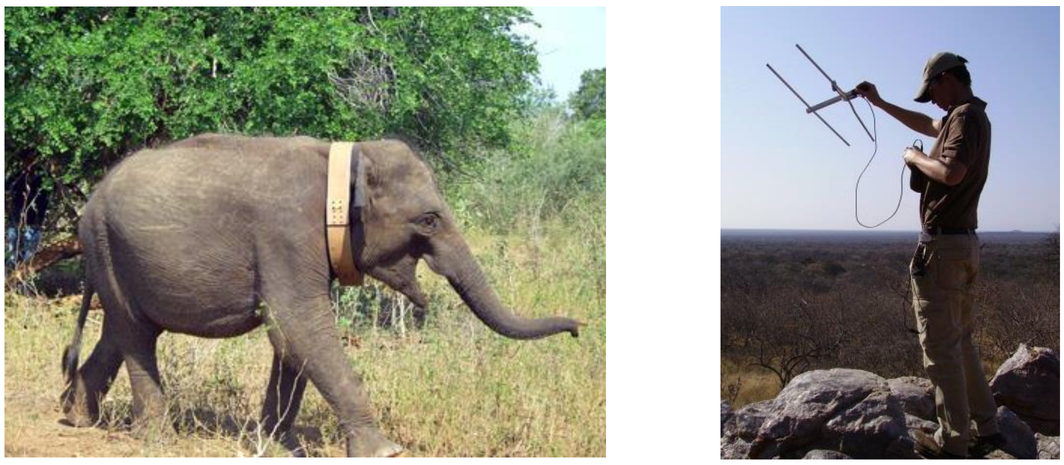
2.2 Animal monitoring and tracking systems

This section introduces some techniques found on literature regarding animal monitoring and tracking systems. We'll emphasize on livestock monitoring and data uplink solutions based on SheepIT's needs.

Tracking Systems

- **VHF telemetry**

Animal tracking systems have gained interest amongst wildlife researchers since a long ago [41]. One of the first known technologies to be employed consists in the usage of a VHF transmitter carried by the animal (usually placed in a collar) and a receiver tuned at the same transmitter's frequency with a directional antenna. By rotating the receiver's antenna until the strongest signal is found, a human operator can estimate in which direction the transmitter is, relative to the receiver. The operator can then follow the RF signal towards the animal carrying the transmitter. This tracking method is not suitable for daily coverage, so other techniques have started to be used for this purpose.



(a) **(b)**

Figure 2.7 – An elephant carrying a VHF collar (a) [42] and an operator tracking a signal (b) [43]

- **Geolocation (satellite based)**

Satellite based systems are being employed in animal tracking providing geolocation. There are mainly two systems currently being used for this purpose:

- **Argos System:** this is a satellite system developed for wildlife and environmental research [44]. It provides geolocation as well as a data uplink.

- **GPS:** the *Global Positioning System* provides geolocation but an uplink is absent.

Despite the usage of Argos as an animal tracking system, its mention serves only as an historical and contextual perspective, like we did with the VHF telemetry. Its narrow focus on the nature research community precludes its usability outside of this field.

The use of GPS to monitor animal locations and their welfare are the most documented solutions in literature. It is particularly interesting to this dissertation the literature referencing the employment of tracking systems in livestock monitoring, due to SheepIT's project objectives regarding this topic.

Rutter *et al.* studied the grazing areas of domestic sheep using their own developed animal behavior and tracking device based on GPS [45]. The sheep-borne system was compound of a GPS receiver, a microprocessor and some behavior sensors (to determine when sheep were grazing), being the GPS receiver the component with the highest power consumption (1.3 W). A large lithium battery pack (30 Ah) was used to power the device, accounting for an overall weight that exceed 1 kg that was carried on the backs of the animals (Figure 2.8).

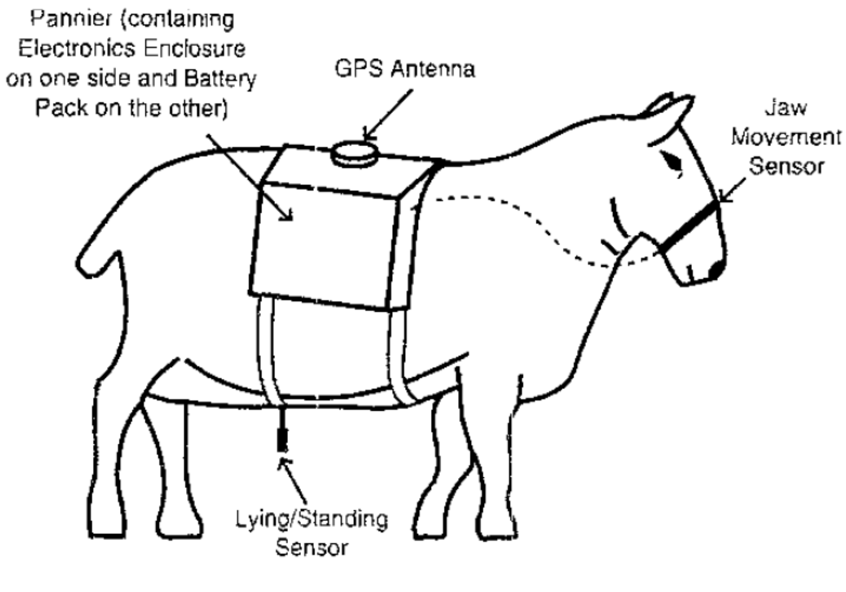

Pannier (containing Electronics Enclosure on one side and Battery Pack on the other)
GPS Antenna
Jaw Movement Sensor
Lying/Standing Sensor

Figure 2.8 – Rutter *et al.* GPS based tracking system for domestic sheep [45]

In the aforementioned study, three GPS power management schedules were tested:

- **Schedule 1:** GPS continuously powered with a new position fix request every 60 seconds (s);
- **Schedule 2:** GPS turned on for 150 s and off for 150 s with a new position fix request every 300 s;
- **Schedule 3:** GPS turned on for 90 s and off for 90 s with a new position fix request every 180 s.

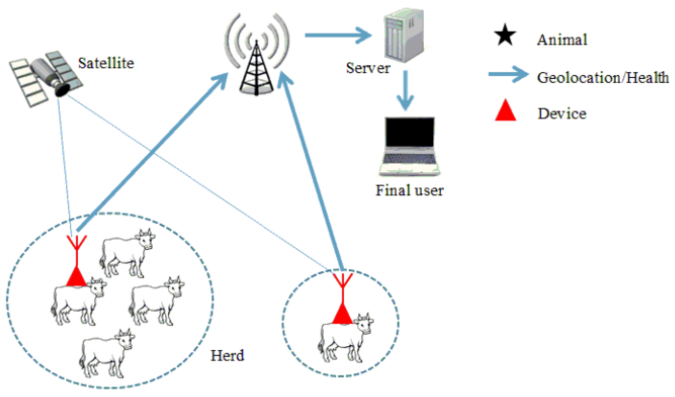
Three tracking devices were deployed in different animals, each one with a different power management schedule. The most relevant results to our interest are shown in Table 2.1.

Power Management Schedule	**Battery life (30 Ah)**	**GPS fix success rate**
Schedule 1	4 days, 5 h	98 %
Schedule 2	6 days, 18 h	100 %
Schedule 3	7 days, 19h	97.5 %

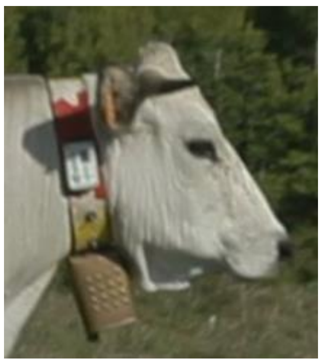
Table 2.1 – Battery life of Rutter *et al.* sheep tracking system (adapted from [45])

The authors of this studied concluded that the use of a 50 % duty-cycle did not affect the ability of the GPS receiver to successfully respond to new position fixes, increasing the battery life up to more than 60 %. However, the same study enlightens us about the GPS receivers high power consumption. An optimal duty-cycle for the GPS receiver was not concluded in this study, but authors stated that because these animals were found to be lying down during 40 % of the time, the battery life could be increased if GPS receivers were turned off when no animal activity was found.

The integration of accelerometers in these sort of systems is also prominent in literature, complementing the GPS receiver [46]. Regarding livestock tracking, e-Pasto [47] is a platform to monitor the localization and welfare of cattle in mountain pastures that makes use of GPS receivers with accelerometers. Geolocation measurements are only requested when the sensors detect a valid animal movement, minimizing the power consumption [48]. This project is targeted to farmers, by proposing itself to be a pasture resource management tool. The collected geolocation data allows farmers to keep track of the grazing areas and enables the triggering of alarms when animals cross undesired boundaries. The practical implementation of this system resulted in a battery autonomy up to 7 months with GPS position captures at a fix rate of one per hour. The motion sensor algorithm was only validated respecting the localization accuracy relative to a continuously powered GPS receiver of a smartphone. At the present time, no further tests have been documented so it's hard to predict the impact of this solution in the battery life of the system due to the animals' behavior dependency. The system architecture and a collar device is shown in Figure 2.9.



(a)



(b)

Figure 2.9 – The architecture of e-Pasto platform (a) and their developed geolocation device positioned on an animal's collar (b) [47]

Still in respect of GPS technology drawbacks, another documented limitation is the loss of satellites connection [49] which imposes more difficulties in finding the correct balance between the duty-cycle level and the localization performance.

- **Low-cost alternatives**

The high energy consumption of satellite based systems gave rise to the research of alternative localization methods based on a radio link, namely, angle of arrival (AoA) [50], time difference of arrival (TDOA) [51] and received signal strength indicator (RSSI).

The use of RSSI to estimate the relative localization of a given node became an interesting research topic due its very low-cost implementation. Once a network based on a radio link is set-up to gather data from several nodes, the same link can also be used as a relative localization method since most of the radio transceivers already incorporate the RSSI parameter. This minimizes the hardware requirements and power consumption since no additional hardware is needed apart from the nodes' CPU and radio transceivers to establish a wireless link.

The background theory behind RSSI localization is that the received power (in dBm) is inversely proportional to the square of distance between the receiver and the radio signal source when in line of sight [52]. This, in theory, enables a fairly good estimation of the distance between a transmitter and a receiver node. However, experimentation results emphasize that RSSI is not an accurate method for outdoor localization [52], [53].

Notwithstanding some pessimistic results, RSSI has already been tested in proposal cattle monitoring systems [49], [54] with self-claimed acceptable results [55]. Therefore, the inclusion of radio transceivers with RSSI support should be considered if a network based on radio technology is planned to be deployed, giving support for further investigation in algorithms to improve the RSSI relative localization.

Data uplink solutions

Not every localization technology provides an uplink on its own. This usually leads to the inclusion of a dedicated communication data uplink in addition to the geolocation system. The most documented data uplink solutions being employed for this purpose can be summarized as follows:

- Satellite communication;
- Cellular networks;
- Low-power wide area networks (LPWAN);
- Wireless Sensor Networks.

Satellite communications are more suitable for remote areas but its cost along with limited data upload frequency can be a drawback [56].

The usage of cellular networks, through the inclusion of GSM modems in the animal-borne systems, is found in e-Shepherd project [57], but it presented some communication problems.

The e-Pasto platform [47] takes advantage of SIGFOX [58], a LPWAN. These networks provide wide coverage with minimum power consumption on a subscription basis, but the low data rates (up to 600 bits per second) must be taken into account when opting for them.

Wireless sensor networks can be deployed to collect data from the geolocation equipped devices. The use of Zigbee protocol for this purpose is common, being examples of its usage in cattle monitoring the proposal solutions [50] and [56]. The main drawbacks of WSNs for this purpose are related with the difficutlties and cost to support wide areas coverage.

2.3 Summary

In this chapter, we've introduced the concept of a WSN. Regarding the MAC layer, different strategies are proposed whose decision over which best fits a project scenario usually assumes some trade-off between communication performance and energy efficiency. These strategies are commonly grouped as contention-based and scheduled-based.

Being one of SheepIT's project concerns the energy efficiency of the system, we've opted to identify the issues in the MAC layer and protocol stack that lead to energy waste: packet collisions, overhearing, idle listening and protocol overhead. Contention-based and scheduled-based MAC policies treat these issues differently, so, to make some sense out of the concepts, some WSN protocols were surveyed and grouped in these two categories.

Contention-based protocols are best suited for scenarios in which communications happen at non-predictable instants. They are characterized for having low latency, but packet collision is reduced at the expense of overhearing and idle listening (preamble-sampling). When traffic raises, packet collisions become more frequent. Thence, schedule-based protocols become a valid alternative by eliminating this issue, but they increase the communications latency. Moreover, they require synchronization between peer nodes which increases the protocol's complexity and its overhead (synchronization packets).

Related work about existing technologies for animal tracking systems was also presented. The most prominent technology employed in outdoor localization is GPS, but its high power consumption and loss of satellite connection is emphasized in some literature. As an alternative to satellite geolocation, development of localization algorithms that rely on radio signal received strength (RSSI) has been documented. Respecting the accuracy for outdoor localization, the results found in literature are not fully satisfactory, howbeit, due to its inexpensive implementation, this localization approach is still an interesting research topic.

Chapter 3

System Architecture

This chapter presents the SheepIT architecture. Based on the project's motivation, the requirements are firstly introduced while an overview of the proposal system is discussed to address those requirements. The remaining parts of the chapter will focus on the MAC layer of the WSN.

3.1 Project scenario constraints and requirements

The project requires a system capable of monitoring the behavior and the localization of several hundred sheep in an autonomous and energy efficient way.

A portable device will be tied up to the sheep's neck – *collar* device. It will include sensors to monitor the head position and actuators to apply a stimulus to correct its behavior and position. A human operator should be able to visualize remotely the collected data from the sensors and the localization of each sheep in a herd. A wireless communication infrastructure must then be designed to accommodate the communications of each sheep device in an entire flock. The scalability of the network is important to provide support for different flock sizes and terrain topologies. A relative localization system based on RSSI techniques alongside with a set of stimulus encompassed on collars, enables the implementation of a virtual fence mechanism using the same link provided for data communications.

The requirements for the SheepIT Project can be summarized as follows [2]:

- **Size, weight and autonomy**: provided that collars will be carried by sheep, it is important to make these devices comfortable to wear and with an autonomy up to 4 months [57].

- **Posture control**: Sheep will be weeding close to grapes, so it is important to address methods for controlling their behavior, namely, control their head's position and correct them via stimulus application to prevent them from feeding on the lower branches of the vines and from the grapes [7].

- **Virtual fence**: A virtual fence shall be implemented to enable the confinement of sheep to certain areas. It is desirable to localize single animals with a resolution close to the average error of GPS but with less energy consumption.

- **Local processing**: The application or absence of stimulus is a decision that should be take locally, on the collars

3.2 Network overview

The proposed network incorporates a Wireless Sensor Network (WSN) layer, a cloud computing layer and an application layer.

The WSN layer is composed of mobile and fixed nodes. The mobile nodes (*collars*) are carried by the sheep, collect data from sensors about the animal's posture and upload that information to the network via a radio link. Besides the sensors, the collars also include actuators to apply stimulus to correct the sheep behavior. The reported data from each collar is gathered by the *beacons*, fixed nodes strategically placed on the field. These, relay the received packets between them until they are delivered to a *gateway*, a beacon connected to a computer based system with internet access, where the WSN layer meets the *cloud*. By measuring the RSSI of the communications between the beacons and the collars, the localization of each collar relative to a certain beacon can be estimated. A more precise localization of the collars can be managed if the areas covered by individual beacons overlap and if the location of each beacon is well-known, by providing a GPS device on each one.

Respecting the MAC layer of the prosed WSN protocol, SheepIT takes advantage of the hybrid approach. Communications are governed by a periodic time frame in which every node is assigned with a unique TDMA slot for their packet transmissions. A contention based window is also present in the protocol to allow the exchange of unscheduled traffic to dynamically support the admission of nodes that are asking to pair with the network.

Summarizing, the SheepIT network is architected in order to converge data from several *collars* to a *cloud*. To allow covering arbitrarily large areas, the information is relayed by *beacons* until it reaches a *gateway* connected to the Internet. The processed information on the cloud can then be delivered to the user through a web application.

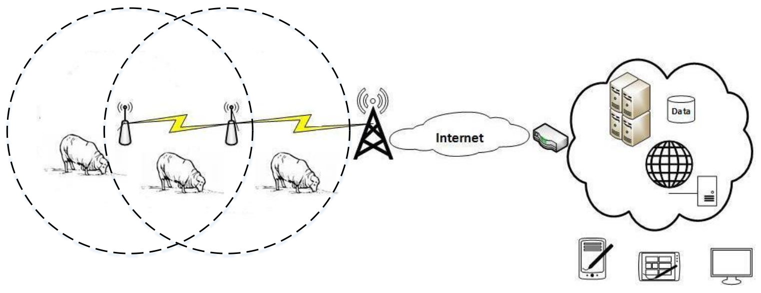


Figure 3.1 - SheepIT's proposed network

3.3 Network nodes

This section focuses on the various types of network nodes and their hardware requirements to successfully fulfil their purposes regarding only the WSN layer. The “lifecycles” of these nodes are also briefly explained.

Beacon

The purpose of the beacons is to collect the data sent by the collars and relay it to their neighbors until it reaches the end point of the WSN layer. Moreover, they are responsible for the communications’ frame synchronization through periodic synchronization messages.

Covered areas of neighbor beacons should overlap. This will let them maintain a radio link to properly exchange data and also enables the address of triangulation techniques to carry out the collars’ localization based on the periodic synchronization messages RSSI.

Regarding the hardware requirements, we highlight that memory usage might be an issue, because in the worst-case scenario an entire flock data must need to be buffered prior to be relayed. These devices will be powered by batteries, but since they are not meant be portable a large pack can be used to fit the power consumption and autonomy needs. A radio transceiver has to be present.

The lifecycle of a beacon is depicted in Figure 3.2. After being booted-up, the beacon tries to synchronize the communication frame by listening to the gateway or neighbor beacons synchronization packets. Once the frame is synchronized, the beacon will wait for the contention-based window to ask for an admission in the network (*pairing stage*). After being admitted in the network, a unique ID is granted to the beacon and its traffic is scheduled according to it.

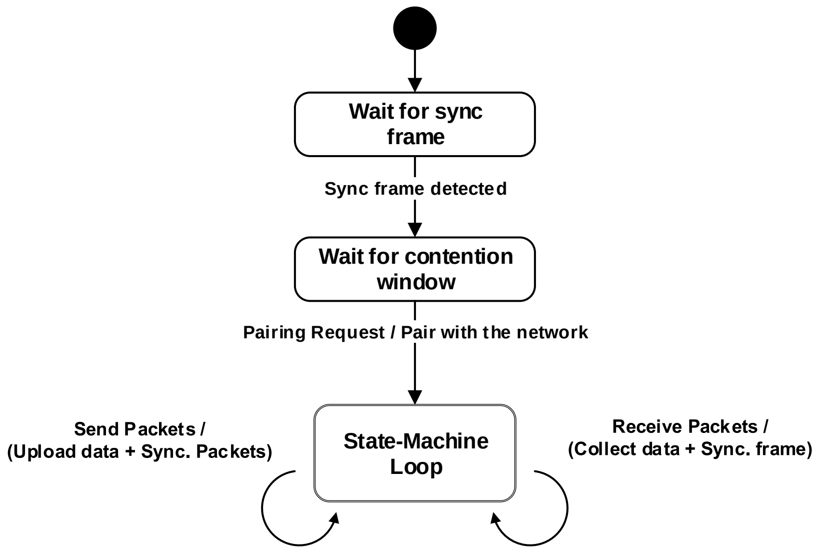

Wait for sync frame
Sync frame detected
Wait for contention window
Pairing Request / Pair with the network
Send Packets / (Upload data + Sync. Packets)
State-Machine Loop
Receive Packets / (Collect data + Sync. frame)

Figure 3.2 – Beacon’s lifecycle

After the pairing phase, the infinite state-machine loop starts to be executed. It consists in listening to the exchanged traffic, upload internal tables and transmit packets. The traffic listening serves both for data collecting, either from collars or from relayed data by other beacons, and to synchronize the communications' frame. Two packet types can be transmitted by a beacon: a frame synchronization packet and a message to upload the collected data.

Gateway

Gateways are the final nodes of the WSN layer. These nodes encompass a beacon wired to a computer based system with Internet access, interconnecting the SheepIT WSN with the cloud. Collars' data is dumped to the gateway, after being relayed by beacons. Gateways also answer to nodes' *pairing requests*, by assigning them a unique network ID.

By taking advantage of the bigger processing power of the PC, more complex data (network management algorithms) processing should be delegated to the gateway, keeping the rest of the nodes' firmware as light as possible.

Collar

This is the only mobile node on the network. It is the device that every sheep on the mob will carry on their neck, monitoring the animal's posture and location and it is powered by a rechargeable battery. Due to this, severe device dimension, weight and energy consumption requirements are imposed.

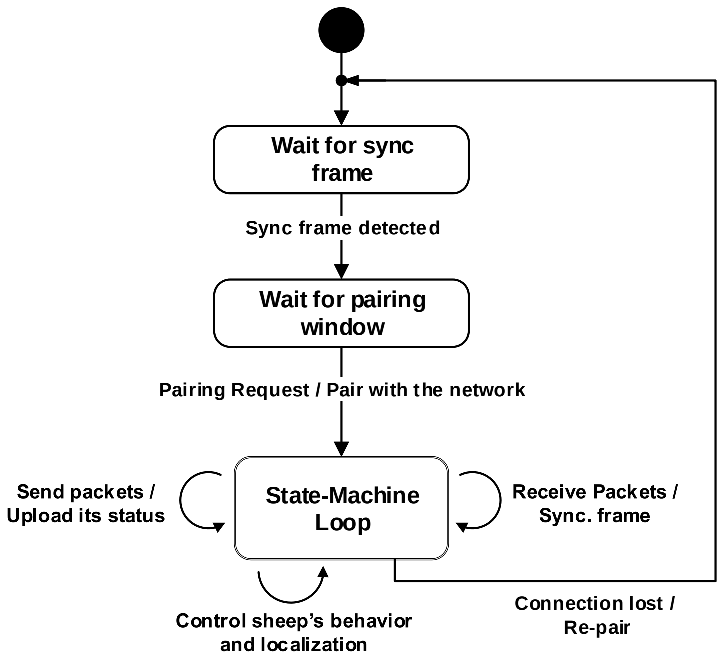

Wait for sync frame
Sync frame detected
Wait for pairing window
Pairing Request / Pair with the network
Send packets / Upload its status
State-Machine Loop
Receive Packets / Sync. frame
Control sheep's behavior and localization
Connection lost / Re-pair

Figure 3.3 – Collar's lifecycle

Besides the hardware for posture control [7], the collar needs to include a transceiver to enable communication via a wireless link. The same link should enable relative localization via RSSI techniques, providing that the radio modules support this parameter.

The lifecycle of a collar is depicted in Figure 3.3. Once a collar is booted-up, before it can be mounted on the sheep's neck a human-aided registration process takes place, in which the collar is associated with a sheep.

After the registration, the collar initiates the pairing-request procedure (admission in the network).

As soon as the device gets registered and paired with the network, the state-machine's infinite loop begins execution. It comprises the execution of the posture control and localization algorithms and the upload of relevant data about the collar / sheep status. When a collar detects it has lost the connection with every beacon (possibly due to an out of fence condition) it is suggested that it restarts the pairing algorithm. By listening to the beacons' synchronization packets the collar is able to synchronize the communications' frame allowing it to duty-cycle its activity (enter "sleep" modes) for power consumption reduction.

3.4 Medium Access Control (MAC)

As described in Section 3.3, SheepIT's network is composed of distinct equipment that need to maintain a reliable communication with each other using the same link. This leads to several problems, namely, collisions, packet losses and inefficient bandwidth usage that need to be addressed. In this section, the proposed solution to handle the medium access issues is presented.

We'll begin with the definition of two key concepts – the micro-cycle and the macro-cycle.

3.4.1 Micro-cycle (µC) and Macro-cycle (MC) definitions

Due to the large number of collars that are expected to be part of a SheepIT network and because it is desirable to periodically collect data from each one of them, the WSN type that best fits this purpose is a schedule-based network. Nevertheless, to support the admission of new collars in a deployed network without the need to repeat a complete set-up phase, requires the existence of contention based windows to allow new nodes to ask for admission permissions (using CSMA). Moreover, collar's traffic and beacon relay traffic will also be scheduled to be exchanged in different periods. To accommodate different types of traffic, it is proposed that they are exchanged in a periodic pattern [59].

For each traffic type, a time-frame is devoted. These were given the name of **micro-cycles** (**µC**). A periodic sequence of micro-cycles is called a **macro-cycle** (**MC**), which will also periodically repeat over time. Figure 3.4 illustrates these two concepts.

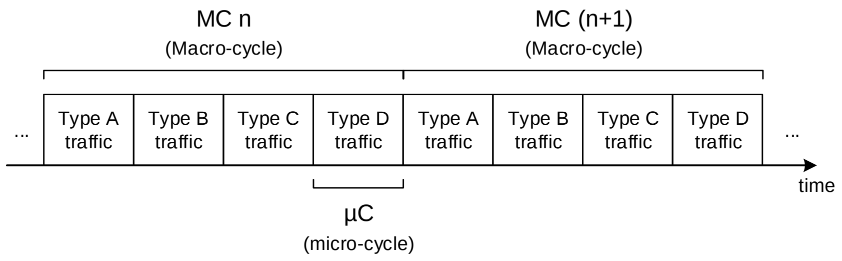


Figure 3.4 – Micro-cycle (µC) and Macro-cycle (MC) illustrative example.

Traffic Types A, B, C and D are not meant to represent four different types of traffic. They are only intended to represent a sequence of four micro-cycles, as an example, (devoted to any kind of traffic type) that is repeated in a periodic manner.

Micro-cycle (µC) internal structure

The micro-cycles are themselves divided in smaller frames, named **windows**. A total of three windows make up one µC: Synchronization Window **(SW)**, Turn-around Window (**TAW**) and a Variable Traffic-type Window (**VTW**).

These smaller frames are arranged always in the same order and the sequence is depicted in Figure 3.5. Hereby, after the VTW of one µC comes the SW of the following µC.

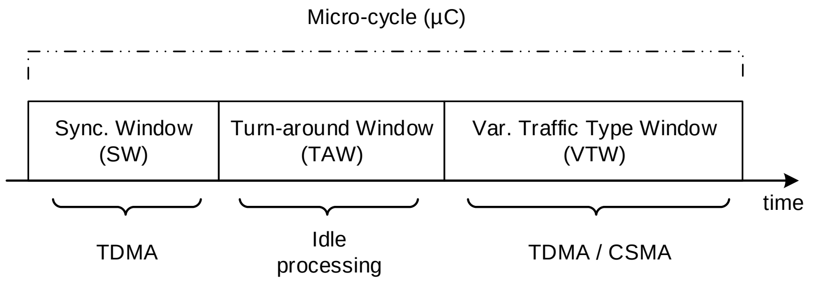


Figure 3.5 – Internal structure of a micro-cycle (µC)

The SW is devoted to the µC frame synchronization. The TAW is reserved for most of the processing. Lastly, the VTW's purpose is to allow the exchange of the last mentioned different types of traffic. Moreover, we will categorize the µCs by the type of traffic that can be exchanged during their VTW.

▪ Synchronization Window (SW)

During the SW, all devices synchronize the µC frame and the type of traffic that should be exchanged on the VTW is announced.

Every node is asked to be awake at this stage. The beacons are responsible to send the synchronization packets (one per beacon) and they are addressed to every node (broadcast). These packets were given the name of *Beacon Synchro* (**BS**). They provide an input to the node's synchronization algorithm and announce what data type should be exchanged during the VTW. The BS packets are scheduled by granting a unique time slot to each beacon (Figure 3.6)

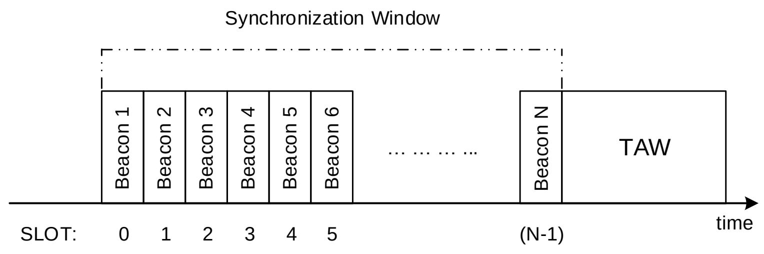


Figure 3.6 – Synchronization Window split in multiple beacon time slots

The beacon time slots are conceptually contiguous. However, this picture does not represent the needed separation between packet transmissions.

Besides announcing the VTW traffic, the packet sender ID is also announced in the synchronization packet. By relating the announced beacon ID with the corresponding time slot, nodes that receive a BS packet can compute the time stamp of that packet. This enables the receiver node to synchronize with the transmitter. Section 3.4.5 is devoted to this synchronization algorithm.

- **Turn-around Window (TAW)**

Different activities will be addressed to beacons and collars during this frame. Nevertheless, the TAW duration is the same for every node and during the full time it takes no radio communications happen.

We saw that synchronization packets are sent by beacons during the previous window. In addition to providing an input to the synchronization algorithm, those messages let beacons know about the existence of their neighbors. This information needs to be relayed, so beacons without a direct link can still know each other. Furthermore, any collar data collected by a beacon also needs to be relayed. The need for packet relay, or at least of some of their contents, enforces the received messages to be buffered.

It's during the TAW that buffered packets received during the previous windows can be processed (Figure 3.7).

The beacon's TAW operations consist mainly in identifying the next µC type (based on a known sequence), updating internal tables (beacons' and collar's status) and creating new packets.

While beacons are doing their tasks, collars read their sensors, perform posture and localization control algorithms and decode the current µC type based on the buffered BS packets. If the current µC's VTW is dedicated to the collars traffic, then a packet should be created to upload its current status (sensor values, battery level, etc.).

The TAW length must be dimensioned for the worst-case scenario, that is, the sequence of most time-consuming tasks must fit within this frame duration.

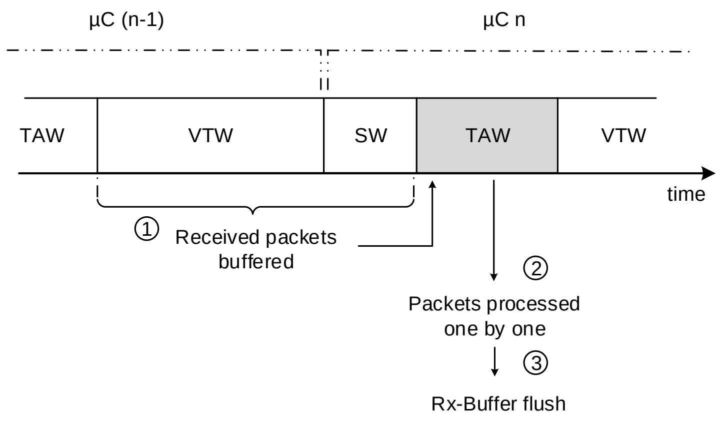


Figure 3.7 – Buffering of received packets prior to be processed during the TAW (beacons)

Collars also buffer the received synchronization packets but not VTW traffic

- **Variable Traffic-type Window (VTW)**

Along this window, data corresponding to the traffic type announced on the previous SW is exchanged.

While TDMA is always used in the SW, different MAC policies schemes are used in the VTW, depending on the traffic type (scheduled or contention-based). We'll talk about this in the next section, as we categorize the micro-cycles by their VTW's traffic type.

3.4.2 µC Types

Depending on the data exchanged on the VTW, µC's can be classified in three types (Table 3.1).

Nodes that have already been paired with the network own a unique ID. This enables the schedule of their communications using a TDMA scheme by transposing their ID to an exclusive time slot. These TDMA scheduled communications happen in every SW and in VTWs of type 2 and 3.

For type 2 µC's, the VTW is devoted to collar traffic. The packets sent during this VTW were given the name of *Collar-to-Beacon* (**C2B**). As the name suggests, the messages contain collar data intended to be collected by the nearest beacons. A single time slot is granted to each paired collar to transmit its C2B packet and they are assigned in ascending order of collar ID's (slot 0 for the collar with lowest ID, etc.).

Type 3 µC's VTWs are dedicated to inter-beacon relay messages exchange. These messages were given the name of *Beacon-to-Beacon* (**B2B**). Time slots are assigned using the beacon ID's the same way it is perpetrated in the SW.

For unpaired nodes, it is impossible to predictably schedule their communications because they haven't announced themselves to the network yet. To ask for an ID, unpaired beacons and collars should wait for a type 1 µC VTW to send their pairing requests in a contention-based manner. Beacons that listen to pairing requests of their peers or collars, relay them in B2B packets in the subsequent type 3 µC's VTW. The relayed pairing requests are intended to reach the gateway which is responsible to address them in a form of pairing replies that are also relayed in B2B packets. A beacon with pairing replies buffered in memory, transmits them in a contention-based manner during the subsequent type 1 VTW. Summing up, a VTW of a type 1 µC serves as a contention window for both pairing requests and pairing replies to pending requests.

µC Type	**Purpose (VTW)**	**MAC Policy (VTW)**
1	Node Pairing	CSMA / TDMA
2	Collar Communication	TDMA
3	Inter-beacon Relay	TDMA

Table 3.1 - Micro-cycle types

3.4.3 µC sequence design (MC)

As seen in Section 3.4.1, a macro-cycle (MC) is defined as a periodic sequence of micro-cycles (µC). Now that the three µC types are presented we can discuss the concept in more detail.

The sequence of µC types that form the MC is not strictly defined by the architecture. This can be adjusted to suit the specific needs that the vineyard size and terrain topology imply. As the property size increases, certainly more beacons will be demanded to increase the covered area, thus, more type 3 µC's need to exist to enable the data relay.

Some remarks can still be made about the µC sequence when defining one:

1. A type 1 µC needs to exist to enable the pairing of new nodes during runtime.
2. A type 2 µC needs to exist to collect the collars' data.
3. The minimum number of type 3 µC's needed is the number of hops that separate the gateway from the furthest beacon.

3.4.3.1 Sequence convergence

The sequence of micro-cycles (µC) that form the periodic macro-cycle (MC) is known to every beacon, but its progression needs to be synchronized to prevent beacons from announcing different types for the same µC.

By taking advantage of a centralized scheme for network nodes' admission, this sequence can be initiated by the gateway's beacon and followed by the others. If the MC has one µC whose type is unique in the entire MC (let's say there is only one µC of type 1), then waiting for the announcement of this unique µC would have been enough to synchronize the entire sequence. However, because the dynamic network node pairing wasn't tested yet and different approaches may be considered in future, such as a decentralized scheme, it might be reasonable to adopt a more robust method of synchronizing the sequence. For that purpose, the synchronization packets additionally to announce the µC type also announce the order of the current µC within the MC frame (as shown in Figure 3.8).

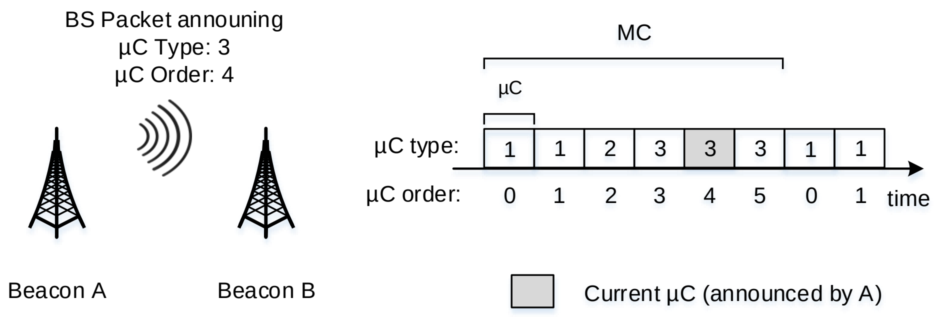


Figure 3.8 – Example of a beacon announcing the µC type and its order within the MC frame

3.4.4 Time constraints

This section addresses the MAC layer time constraints, namely, the time slot definitions and µC's length.

3.4.4.1 Time slot definition

During earlier experiments, it was concluded that the time a receiver node takes to process the packet header and buffer the message isn't negligible. This affects the minimum required waiting time between successive packet transmissions. We will call this duration $\boldsymbol{TT_{RX}}$ (*time-to-receive*). Moreover, the time spent to transmit a packet will also be considered to define a slot duration – $\boldsymbol{TT_{TX}}$ (*time-to-transmit*).

The time slot length needs to comprise both the $\boldsymbol{TT_{RX}}$ and $\boldsymbol{TT_{TX}}$. These parameters are dependent on the size of the packets, so different time slot lengths will be defined for different types of traffic (with different sizes). Hardware specifications, namely the clock frequency and tx data rate, also influence these lengths.

Synchronized clocks may continue to drift between the synchronization instants. Moreover, time uncertainties due to the timers' resolutions and different data processing loads, can lead to the shortening of the interval between two packet transmissions to less than TT_{RX}. If this interval is not respected it is not guaranteed that receiver nodes are ready to receive when a given packet transmission starts, resulting in packet losses. To deal with these imprecisions, a **guarding window** (**GW**) is added between consecutive slots to allow the packet transmission instants (tx_{time}) to swing back and forth without overlapping the TT_{RX}. Figure 3.9 portrays a set of contiguous time slots separated by one GW. The computations of TT_{RX}, TT_{TX} and GW lengths were carried out in Section 5.1.2 for the specific SoC data rate and timers' accuracy/resolution in which the SheepIT protocol was tested.

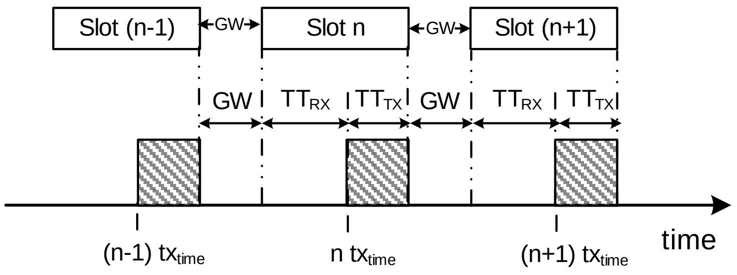


Figure 3.9 – A set of contiguous time slots separated by a guarding window (GW)

3.4.4.2 µC length

To simplify the architecture, it was opted to address each micro-cycle with the same length. The minimum length that fits each µC type needs to encompass the minimum length required for each window. Only the VTW minimum length is dependent on the µC type because it needs to address different number of slots for a type 2 (one slot per collar on the network) and for a type 3 (one slot per beacon on the network). It is assumed that the minimum VTW length that fits both type 2 and 3 TDMA traffic will be enough to serve as contention window for the CSMA traffic of type 1 µC's. Using the complete representation of the µC structure (with schedule-based VTW) portrayed in Figure 3.10, the minimum length of each communication window will be deduced.

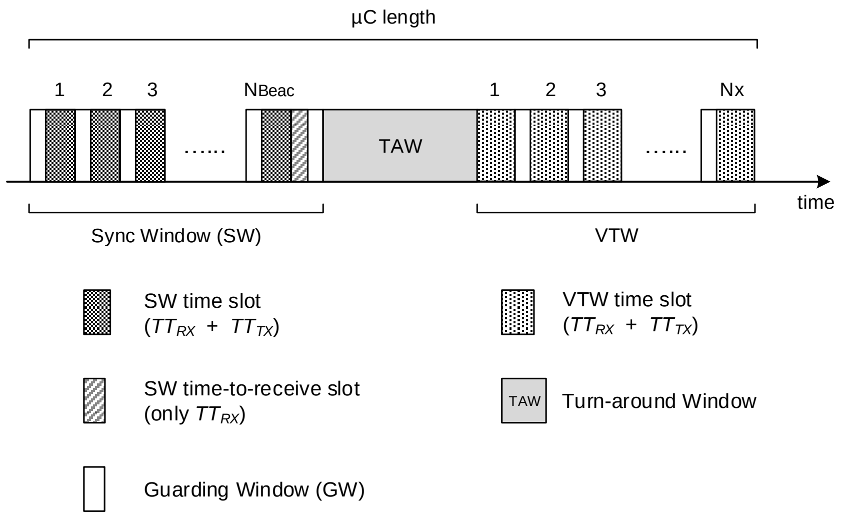

µC length
1
2
3
NBeac
TAW
1
2
3
Nx
......
......
time
Sync Window (SW)
VTW
SW time slot (TT_{RX} + TT_{TX})
VTW time slot (TT_{RX} + TT_{TX})
SW time-to-receive slot (only TT_{RX})
TAW
Turn-around Window
Guarding Window (GW)

Figure 3.10 – Example of a complete type 2 / type 3 µC

Respecting the SW length, note that the last SW time slot is followed by a single TT_{RX} (plus one GW). That is the *time-to-receive* of the previous time slot packet. The length of this window can be computed as in (1), in which $BS_{TT_{RX}}$ and $BS_{TT_{TX}}$ are the *time-to-receive* and the *time-to-transmit* of a Beacon Synchro packet, respectively, and N_{Beac} is the total number of beacons. GW is the length of one guarding window.

$$SW_{length} = N_{Beac} \times \left(BS_{TT_{RX}} + BS_{TT_{TX}} + GW\right) + \left(BS_{TT_{RX}} + GW\right) \tag{1}$$

The VTW length for type 2 µC's is computed as in (2), in which N_{col} is the total number of collars and the $C2B_{TT_{RX}}$ and $C2B_{TT_{TX}}$ times are the Collar-to-Beacon packets' *time-to-receive* and *time-to-transmit.*

$$VTW^2_{length} = N_{col} \times \left(C2B_{TT_{RX}} + C2B_{TT_{TX}} + GW\right) - GW \tag{2}$$

For type 3 µC's, the VTW length is computed as in (3), in which N_{Beac} is the total number of beacons and the $B2B_{TT_{RX}}$ and $B2B_{TT_{TX}}$ times are the Beacon-to-Beacon traffic packets' *time-to-receive* and *time-to-transmit.*

$$VTW^3_{length} = N_{Beac} \times \left(B2B_{TT_{RX}} + B2B_{TT_{TX}} + GW\right) - GW \tag{3}$$

Therefore, the length of a type 2 and a type 3 µC can be computed as in (4) and (5), respectively, in which TAW_{length} is the length of the turn-around window.

$$µC^2_{length} = SW_{length} + TAW_{length} + VTW^2_{length} \tag{4}$$

$$\mu C^3_{length} = SW_{length} + TAW_{length} + VTW^3_{length} \tag{5}$$

The minimum µC length that fits both types is the maximum of the two previous computed lengths:

$$\mu C_{\min length} = \max(\mu C^2_{length}, \mu C^3_{length}) \tag{6}$$

3.4.5 Synchronization algorithm and drift compensation

The synchronization procedure takes place inside the SW in which two events are synchronized: the synchronization packets transmission instants ($BStx_{time}$) and the TAW start of execution (TAW_{time}).

To simplify the following expressions, BS_{slot} will stand for one Beacon Synchro packet slot, encompassing both TT_{RX} and TT_{TX}.

$$BS_{slot} = BS_{TT_{RX}} + BS_{TT_{TX}} \tag{7}$$

Using this notation, the expected spanned time between *"k"* BS time slots from the beginning of the SW and separated by one GW is therefore

$$\Delta t_{expect_{k\ SW\ slots}} = k \times (BS_{slot} + GW) \tag{8}$$

If BS slots are attributed to beacons by ascending order of beacon ID's, the last expression can be generalized as

$$\begin{aligned} &\Delta t_{expect}(bID_i, bID_j) = (bID_i - bID_j) \times (BS_{slot} + GW), \\ &bID_i \in \mathbb{N} \cap [1, MAX\_BEACONS], \\ &bID_j \in \mathbb{N} \cap [1, i[\end{aligned} \tag{9}$$

where bID_i and bID_j represent two beacon ID's whose slots are separated by a time duration of $\Delta t_{expect}(bID_i, bID_j)$.

Finally, the drift between BS slots is defined as the difference between the elapsed time and the expected time.

$$\Delta t_{drift}(bID_i, bID_j) = \Delta t_{elapsed}(bID_i, bID_j) - \Delta t_{expect}(bID_i, bID_j) \tag{10}$$

The previous notations will be used to explain how beacons synchronize their BS transmission instants ($BStx_{time}$) and the TAW's start of execution (TAW_{time}). Collars only synchronize the TAW_{time}.

Synchronization of BS transmission instants ($BStx_{time}$)

A beacon synchronizes its $BStx_{time}$ with the received BS packets from previous time slots. The beacon that is synchronizing computes its $BStx_{time}$ as a function of its ID (*thisID*) and the sender's ID announced in the received packet (*rxID*), as in (11).

$$BStx_{time}\,(thisID, rxID) = (thisID - rxID)\times\left(BS_{TT_{RX}} + GW\right) + (thisID - rxID - 1)\times BS_{TT_{TX}} \tag{11}$$

It's easier to make some sense out of equation (11) using an example. Let's say that beacon 3 has just received a packet from beacon 1. Prior to do any buffering, beacon 3 computes its $BStx_{time}$ by substituting *thisID* by 3 and *rxID* by 1 in expression (11). The result is:

$$BStx_{time}(3,1) = 2\,BS_{TT_{RX}} + 2\,GW + 1\,BS_{TT_{TX}}$$

We can see that this makes sense if in Figure 3.9 (page 29) we let the slot (n-1) be the beacon's 1 slot and the slot (n+1) be the beacon's 3 slot. It's clear that the separation between the end of 1's transmission and the start of 3's transmission (tx_{time}) is the last computed value.

Synchronization of the TAW's start of execution instant (TAW$_{time}$)

Nodes synchronize this event as a function of *rxID* (the synchronization packet sender's ID). It consists in subtracting the elapsed time since the beginning of the SW to the length of the SW (SW_{length}).

$$TAW_{time}(rxID) = SW_{length} - [\,rxID \times (SW_{slot} + GW)\,] \tag{12}$$

Clock drift compensation

Nodes compute the spanned time between two transmissions by doing a time stamp when a new packet is received (prior to do any buffering). This value is compared with the expected time between the two events and the drift is computed as in (10).

The drift is then added to the two computed synchronized instants (11) and (12) as the expressions (13) and (14) show. The $BStx_{time}$ needs to be truncated to a minimum value equal to the BS packet *time-to-receive* (TT_{RX}), so the minimum separation between two packet transmission instants is always enough to receive and buffer both packets.

$$BStx_{time}(thisID, rxID) = BStx_{time}(thisID, rxID) + \Delta t_{drift} \ , \tag{13}$$

$$BStx_{time}(thisID, rxID) \in [BS_{TT_{RX}}, \infty[$$

$$TAW_{time}(rxID) = TAW_{time}(rxID) + \Delta t_{drift} \tag{14}$$

The next time a synchronization packet is received, the expected elapsed time takes in consideration the previous computed drift, meaning that the result of (9) is updated to (15).

$$\Delta t_{expect}(bID_i, bID_j) = \Delta t_{expect}(bID_i, bID_j) + \Delta t_{drift_{previous}} \tag{15}$$

Figure 3.11 portrays an example in which the beacon of slot (n-1) has advanced its *BStx_{time}* by ½ GW. Beacon of slot n computes the drift after receiving beacon's (n-1) packet and adds this value to its *BStx_{time}* also computed after receiving the packet. The result is an extra delay in its *BStx_{time}* equal to the drift of beacon (n-1). The beacon of the following slot, (n+1), has also computed this drift and so it is expecting (Δt_{expect}) that nth packet is delayed by ½ G (to compensate the previous drift), so the next drift should be 0 if this is what happens.

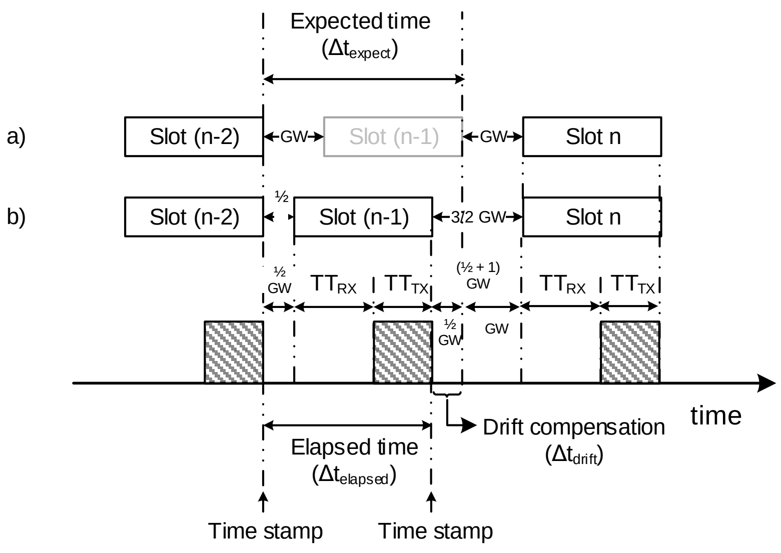


Figure 3.11 – Representation of a compensated clock drift

The beacon of slot (n-1) has advanced its $BStx_{time}$ in ½ GW. The beacon of slot n senses this drift by computing the elapsed time between the last two transmissions and comparing it with the expected time. The difference is added to the waiting time for its $BStx_{time}$, thus, compensating the drift. The slot sequence a) is the optimal synchronized slot sequence while b) is the direct representation of the sequence of packet transmissions viewed as consecutive time slots. Note that due to the clock drift compensation, slot n should start at its predetermined time, not affected by the drift of its predecessor.

If the opposed had happen (a delay of (n-1)th packet transmission by ½ GW), the compensation algorithm would have shortened the time separation between (n-1) and nth packet transmissions to ½ GW.

3.5 Energy efficiency awareness

The SheepIT protocol addresses the energy efficiency of collars with the classical duty-cycle approach.

A collar is only required to enter the transceiver's RX state during synchronization windows (SW). The node's microcontroller stays active in the turn-around window (TAW) and it will only keep in this mode after TAW's ending if a stimulus is applied. If a stimulus wasn't applied and the µC's VTW is not devoted to collar's traffic, a collar can *sleep* during the entire time the VTW takes, only waking-up in the next SW. On the other hand, for VTW's that are addressed to collar communications, it's the schedule-based architecture of this window that still enables nodes to use energy wisely. With a unique slot assigned to each node, a collar can still be able to enter low power consumption states while it is waiting for their transmission slot and during the remaining time for the next SW. Figure 3.12 depicts the collar's duty-cycle during a type 2 µC whether Figure 3.13 depicts it for non-type 2 µC's.

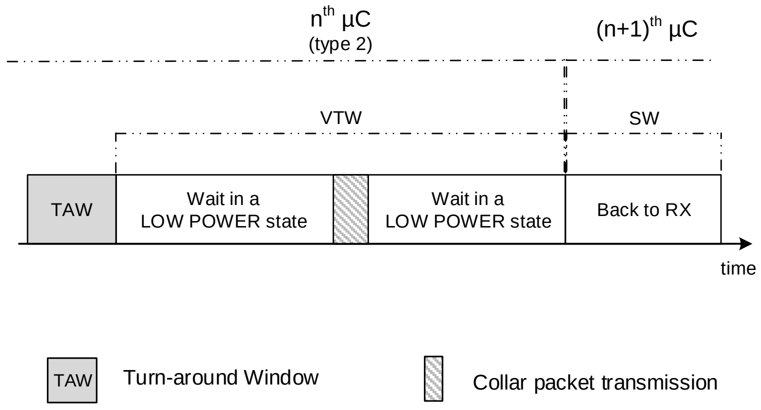

nth µC
(type 2)
(n+1)th µC
VTW
SW
TAW
Wait in a LOW POWER state
Wait in a LOW POWER state
Back to RX
time
TAW
Turn-around Window
Collar packet transmission

Figure 3.12 – Collar's duty-cycle (type 2 µC)

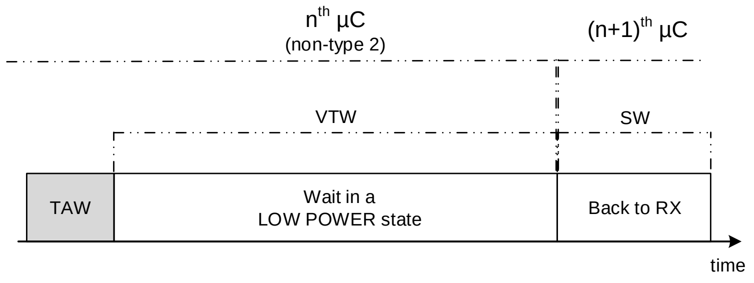


Figure 3.13 – Collar's duty-cycle (non-type 2 µC). Legend in Figure 3.12

The duty-cycle for type 2 µC's can be computed as follows:

$$DC_{type\,2} = \frac{SW_{length} + TAW_{length} + C2B_{TT_{Tx}}}{\mu C_{length}} \tag{16}$$

For non-type 2 µC's, the DC is given by:

$$DC_{non-type\,2} = \frac{SW_{length} + TAW_{length}}{\mu C_{length}} \tag{17}$$

The overall duty-cycle is dependent on the number of type 2 µC's' within the MC. It can be computed as in (18) in which $N\mu C^2$ is the number of type 2 µC's within a MC and NµC is the number of µC' that form one MC.

$$DC = \frac{N\mu C^2}{N\mu C} \times DC_{type\,2} + \frac{(N\mu C - N\mu C^2)}{N\mu C} \times DC_{non-type\,2} \tag{18}$$

Note that stimulus application was not considered in DC computation. This is dependent on the animal's behavior, but since it is expected to be sporadically we find these equations still meaningful.

3.6 Protocol Messages

The presentation of the system's architecture, regarding the WSN layer, is completed with the definition of its protocol messages. The seven protocol messages have already been revealed in previous sections, nevertheless we find important to discuss some of their contents in more detail.

Beacon Synchro (BS)

These are the synchronization packets. They serve the purposes of frame synchronization, routing exchange and provide support for the collar's RSSI based localization.

Beacon Synchro Packet (BS)

Field	**Size (Bytes)**
Beacon ID (sender)	1
Sequence Number (BS-SN)	1
Property ID	2
Route to Gateway (next hop Beacon ID)	1
µC_type | µC_order	1

Table 3.2 – Beacon Synchro (BS) packet

The use of 1 Byte ID's should be more than enough to univocally identify each beacon in a deployed SheepIT network. The inclusion of an incremental sequence number allows the receiving node to detect packet losses by keeping track of the received packet sequence. A property ID is meant to identify the property (a vineyard for instance) that the packet sender belongs to. This helps to filter packets of neighbor properties' nodes, by rejecting them if they happen to be received. The use of 2 Bytes for this field should be enough to uniquely identify a reasonable number of client properties. The route to gateway field is reserved for the announcement of the beacon's next hop ID towards the gateway. No routing mechanisms have been discussed yet, but the inclusion of this field is meant to endow this protocol version with relevant features for future work solutions. The last byte is divided in a µC_type (4 bits) and µC_order (4 bits) field. The former one is reserved for the current µC type (1, 2 or 3) announcement. The last is intended to announce the order of the current µC within the macro-cycle frame.

Collar-to-Beacon (C2B)

These are the packets that collars send to report their status. A 2 Byte ID number identifies the collar and consequently the sheep. The currently reported status are the number of steps that the sheep has taken (1 Byte), the number of times it has been out of the virtual fence (1 Byte), the battery status (1 Byte), the ID of the property who owns the collar, an array with RSSI values carried out for each last received BS packet (1 Byte per RSSI value), an accelerometer sensor read value (1 Byte) and a sequence number (1 Byte).

Collar-to-Beacon Packet (C2B)

Field	Size (Bytes)
Sender ID (Collar)	2
# of steps	1
Number of fence infracts	1
Number of posture infracts	1
Battery status	1
Property ID	2
Received RSSI Array	MAX_BEACONS × 1
Accelerometer value	3
Sequence Number	1

Table 3.3 – Collar-to-Beacon (C2B) packet

Collar Pairing Request (CPREQ)

After the human aided operation of associating a collar with a sheep, a collar sends a CPREQ to the network in order to ask for an ID. A 4 Byte serial number identifies the requester collar. A 2 Byte ID tag identifies the sheep that is carrying the collar. A CPREQ may also be sent by previously paired collars when they have been out of beacons' coverage for too long, since the network may have "recycled" the ID and assigned it to another collar.

Collar Pairing Request Packet (CPREQ)

Field	Size (Bytes)
Collar Serial Number	4
Sheep ID tag	2

Table 3.4 – Collar Pairing Request (CPREQ) packet

Collar Pairing Reply (CPREP)

A CRREP is a response to a CPREQ. It is generated by the gateway and encapsulated inside an inter-beacon relay packet. A collar serial number field (4 Bytes) identifies the collar whose last CPREQ is answered by this CPREP. A 2 Byte ID field announces the assigned ID to that collar. The property ID (2 Byte) is also retrieved by the property's gateway.

Collar Pairing Reply Packet (CPREP)

Field	Size (Bytes)
Collar Serial Number	4
Collar ID (assigned)	2
Property ID (assigned)	2

Table 3.5 – Collar Pairing Reply (CPREP) packet

Beacon Pairing Request (BPREQ)

This is the beacon counterpart of a collar pairing request. In this message, only the beacon transceiver serial number (4 Byte) is sent as a mean to identify the beacon.

Beacon Pairing Request (BPREQ)

Field	Size (Bytes)
Beacon Serial Number	4

Table 3.6 – Beacon Pairing Request (BPREQ) packet

Beacon Pairing Reply (BPREP)

This is the reply to a BPREQ generated by the gateway. These messages are also encapsulated inside an inter-beacon relay packet, as the CPREP messages. It contains the beacon serial number (4 Byte) whose BPREQ is addressed by this reply. The assigned beacon ID is announced (1 Byte) as well as the property ID (2 Byte).

Beacon Pairing Reply Packet (BPREP)

Field	Size (Bytes)
Beacon Serial Number	4
Beacon ID (assigned)	1
Property ID (assigned)	2

Table 3.7 – Beacon Pairing Reply (BPREP) packet

Beacon to Beacon (B2B)

These are the inter-relay beacon messages. They are compound of an header of fixed size plus a trailer with relayed data (variable size). The header identifies the packet sender (1 Byte), the property ID of the beacon (2 Bytes), an incremental sequence number (1 Byte), the battery level of the beacon (1 Byte), its GPS coordinates (8 Bytes) and an array with its neighbor ID's. Additionally, the number of relayed data is announced. After the header, relayed data is sent always in the same order: collar notifications (special data structure to encapsulate collars' status), CPREQ packets, CPREP packets, BPREQ packets and BPREP packets.

A collar notification is a data structure formed by one C2B packet appended to the beacon ID that has received the C2B packet.

Collar Notification Structure

Field	Size (Bytes)
Listener Beacon ID	1
Collar-to-Beacon (C2B packet)	*sizeof* (C2B)

Table 3.8 – Collar Notification (relayed collar data structure)

Beacon to Beacon Packet (B2B)

	Field	Size (Bytes)
HEADER (fixed size)	Sender ID (Beacon)	1
	Property ID	2
	Sequence Number	1
	Battery	1
	GPS Coordinates	8
	Neighbor ID's	MAX_BEACONS × 1
	N_CNOT	1
	N_CPREQ | N_CPREP	1
	N_BPREQ | N_BPREP	1
	Relayed Collar Notifications	N_CNOT × *sizeof* (Collar Notification)
	Relayed Collar Pair. Requests	N_CPREQ × *sizeof* (CPREQ)
	Relayed Collar Pair. Replies	N_CPREP × *sizeof* (CPREP)
	Relayed Beacon Pair. Requests	N_BPREQ × *sizeof* (BPREQ)
	Relayed Beacon Pair. Replies	N_BPREP × *sizeof* (BPREP)

Table 3.9 – Beacon-to-Beacon (B2B) packet

3.7 Summary

SheepIT's WSN layer aims to collect sensor and localization data from each collar carried by sheep and sink it to a gateway using a common radio link. To expand the covered area, beacons are distributed throughout the field keeping a wireless link between them and their closest neighbors. Collars periodically broadcast their sensor and localization data which is listened by the nearest beacons. The collected data is relayed between beacons whose radio link coverage overlaps, which ultimately will end up in the beacon's gateway.

The MAC layer is designed to accommodate the collars' data gathering from collars' traffic and the retransmission of this data through the beacons' relay traffic. An architecture based on TDMA is proposed with a periodicity for each data type of traffic to be exchanged. Moreover, CSMA is also supported to enable nodes to dynamically join the network through a process of pairing requests and pairing replies, which are exchanged in a contention-based manner.

The different types of traffic are scheduled according to a periodic pattern. A time-frame devoted to a certain type of traffic is called a micro-cycle (µC) and the periodic sequence of µC's was given the name of macro-cycle (MC). The architecture defines three types of µC's based on the traffic exchange that they are devoted to: type 1 for pairing requests and replies, type 2 for collars' traffic and type 3 for beacons' relay packets.

A µC frame is in turn divided in three windows. The first one is called the synchronization window (SW) and serves the purpose of frame synchronization between all network nodes through the exchange of beacon synchronization packets. The second window is the turn-around window (TAW) in which no communications happen, packet processing and creation is performed and sensor reading plus control algorithms' execution takes place in collars. The last window is the variable traffic type window (VTW) and is devoted to the exchange of pairing requests/replies, collars' traffic or beacons' relay traffic according to the µC's type.

When a packet is received, a few consistency checks are made to accept that packet, such as decoding the message type (BS, B2B or C2B) and sender ID. The packet gets buffered, so other fields can be analyzed in more detail later. The initial consistency checks required to accept a packet and its buffering may take non-negligible amounts of time. Because this procedure is repeated for every packet that is received, TDMA time slots were defined in order to accommodate both the packet transmission and these consistency checks and buffering times. The former is called the time-to-transmit (TT_{TX}) and the later the time-to-receive (TT_{RX}). A time slot is composed of both a TT_{TX} and a TT_{RX}. To deal with clock drift and other imprecisions, a guarding window (GW) separates consecutive time slots, which can be thought as an extra time added up to TT_{RX}.

Chapter 4

System Implementation

This chapter presents the implemented system architecture. It focusses on the nodes' state-machines and how the synchronous communications were addressed. Due to the size and complexity of the overall system's architecture, the issues that could be addressed in the scope of this dissertation are firstly disclosed.

4.1 Prove of concept requirements

The SheepIT architecture is a collaborative work whose field tests are dependent on a reliable system of communications. Fort that reason and due to the limited effort that could be devoted to a project on the scope of a MSc dissertation, only the more critical functionalities that enable data gathering from field tests have been implemented.

Protocol Messages

The implemented protocol is able to process the following messages with some restrictions:

- **Beacon Synchro (BS):** These messages were implemented with the *Route to Gateway* parameter being a dummy value, since no routing scheme was addressed.
- **Collar-to-Beacon (C2B):** These were fully implemented with real collar's data.
- **Beacon-to-Beacon (B2B):** These messages are currently relaying only collar data. The message header fields *Beacon ID*, *Sequence Number* and *N_CNOT* are the only ones with meaningful values.

Pairing phase

Nodes are statically assigned with an ID, so the registration and pairing processes were not addressed.

Listening phase

Because the implemented prototype does not address the centralized scheme of network nodes admission, type 1 µC's were not implemented. Moreover, every beacon is free to start the sequence of micro-cycles by sending its BS packets once it boots-up. However, to avoid a desynchronized progression of µC types, once a beacon boots up it first experiences a listening phase in order to sense if there's already a neighbor beacon sending synchronization packets. If BS packets are received within this listening time, the beacon stores the announced µC order and follows the µC sequence from there. If no BS packets are received, the beacon assumes it is the first to have booted up and sends a BS packet announcing the first µC of the macro-cycle frame.

Nevertheless, two beacons may start to send BS packets prior to acknowledge the existence of each other. This potentially leads to a desynchronized macro-cycle frame if they keep their progression once they acknowledge the presence of its peer. To prevent this from happening, when a beacon receives BS packets announcing different µC orders or when its current µC order status is different from the announced by their peers, a beacon opts to follow the µC order announced by the beacon with the lowest ID (it could be the beacon itself).

Synchronization

Every node performs the synchronization algorithm as it is stated in the system's architecture chapter (Section 3.4.5).

4.2 SoC main peripherals

The chosen radio transceiver was the **Texas Instruments CC1110** [60]. This SoC comprehends an 8051 CPU with a radio module operating on the 433 MHz ISM band.

The key features that were considered to pick this transceiver were:

- low power consumption in active modes (RX and TX)[1];
- the ability to operate under a low power consumption mode (*sleep*)[2];
- RSSI support;
- the microcontroller unit, with a large number of general purpose I/O pins available as well as useful peripherals like USART, I2S interface and ADC (important features for the collar's posture control hardware design [7]).

On the following subsections, the modules of the CC1110 that were used to implement

[1] Maximum ratings (clock speed at 26 MHz, 433 MHz, 250 kBaud, 10 dBm output) [60]:
RX – 20.5 mA; TX – 33.5 mA

[2] Three low power consumption modes, ranging from 4.3 mA to 0.3 µA of current consumption [60]

the SheepIT's protocol on beacons, collars and gateways are presented.

Radio Module

It either works as a digital radio receiver and a transmitter operating on the 433 MHz band. Moreover, the RSSI support is also used to estimate a relative localization.

The use of variable size packets forces the inclusion of a length field in the packet. Figure 4.1 depicts the packet format.

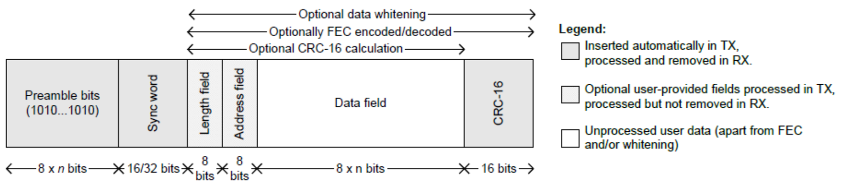

Optional data whitening
Optionally FEC encoded/decoded
Optional CRC-16 calculation
Preamble bits (1010...1010)
Sync word
Length field
Address field
Data field
CRC-16
8 x n bits
16/32 bits
8 bits
8 bits
8 x n bits
16 bits
Legend:
Inserted automatically in TX, processed and removed in RX.
Optional user-provided fields processed in TX, processed but not removed in RX.
Unprocessed user data (apart from FEC and/or whitening)

Figure 4.1 – Packet format on the TI CC1110 [60]

- **Packet transmission**

  A buffered packet is transferred byte-by-byte to the RF Data 1-Byte Register (RFD).

- **Packet reception**

  Every time a new payload byte is received, 1 Byte is transferred from the RFD register to a buffer in memory.

- **Radio Errors**

  If a new byte arrives to the RFD register before the previous one was read, the radio will enter the **RX-OVERFLOW** state.

  On the other hand, if the number of bytes transferred to the RFD register is less than what the radio expects, it will enter th **TX-UNDERFLOW** state.

Timers

Table 4.1 shows the list of available timers on the SoC CC1110.

On collars, timers T3 and T4 are reserved for other purposes related with the sensors and actuators, and for this reason their usage for protocol implementation is limited.

Timer T2 is an 8-bit timer associated with a counter. It is aimed to count multiple integers of a time slot once its execution starts.

TSLEEP is a timer that can interrupt a low power consumption state, thus, waking-up the SoC [60]. While the SoC is in a sleep state, TSLEEP uses a Low Power RC Oscillator, less accurate than the default HS Crystal Oscillator.

Available Timers

	Beacons	Collars
T1 (16-bit Timer)	✓	✓
T2 (MAC Timer, 8-bit)	✓	✓
T3, T4 (8-bit Timers)	✓	✓/✗
TSLEEP (sleep Timer)	✓	✓

Table 4.1 – List of available timers in the CC1110 SoC, regarding the protocol implementation

DMA controller

DMA will be used to handle data transfers from memory to the RF Data Register and vice-versa, triggering an interrupt every time a transfer is complete.

4.3 Beacon's firmware

In the following sections, the beacon's state machine is introduced. An overview lies in Section 4.3.1. Since the state transitions are closely related with the synchronization aspects of the network, to help a better understanding of the state machine itself, the synchronization procedure is presented in section 4.3.2. A deeper view of each state may be found in section 4.3.3.

4.3.1 State-machine

The state machine (SM) is illustrated in Figure 4.2.

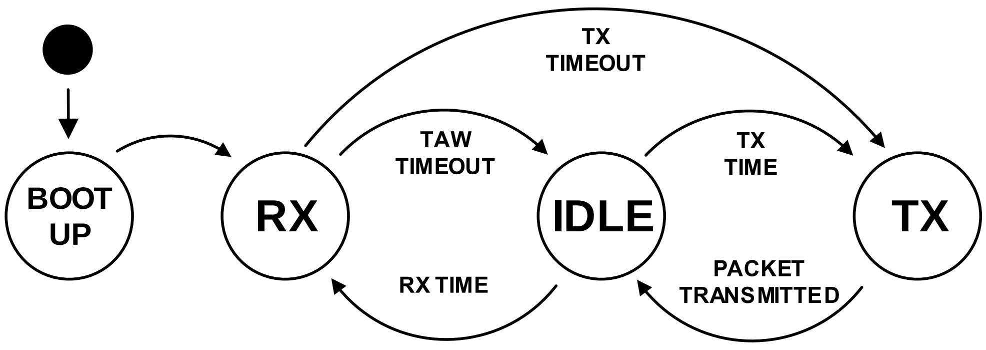


Figure 4.2 – Beacon's state-machine

The SM is based on three main states: *RX* (receiving) when the beacon wants to listen to the network; *TX* (transmitting) when it wants to send data; *IDLE* for data processing when communications are not expected to happen. The term “idle” is, in this context, related with the transceiver’s status (*switched off / idled*) at this state. The CPU continues to be in full operational mode.

When the beacon turns on, the initial state (**Boot-Up**) is entered. At this state, radio module, peripherals and I/O ports are configured. A static ID is programmatically assigned to the node. Constant values related with state transitioning events are also computed. These permit the beacon to operate autonomously when no other beacon is nearby to synchronize with. A BS packet is created to announce the first µC, for the case the beacon senses no other node awake.

After booting up, the beacon assumes it may be in the presence of other beacons and so it enters **RX** state to listen to the network (Listening Phase). It sets the listening phase timeout and waits for BS packets. If no BS packets are received during this time, **TX-TIMEOUT** is triggered (Listening Phase time expired) and the BS created in the initial state is sent after switching to **TX**. However, if a BS packet is received, the beacon acknowledges the presence of a neighbor. This aborts the transmission of the initial BS packet and the TAW_{time} is synchronized with the received packet sender to trigger the **TAW-TIMEOUT**. The reasoning behind this procedure is that nodes only create new packets (updated with the current network status) on the subsequent TAW (see Figure 3.7 to recap the buffering actions).

When in **IDLE** State due to the triggering of a **TAW-TIMEOUT**, the TAW’s operations are executed. At the end of the TAW’s execution, the beacon switches back to RX or to TX if it possesses the first B2B slot and the µC is of type 3.

After a packet transmission, the beacon returns always to the IDLE State to evaluate when the next timeout (TAW-TIMEOUT or TX-TIMEOUT) should be triggered and sets the timers accordingly before switching to RX. This “pause” in the IDLE State, in between the transition from TX back to RX, should not affect the performance of the beacon in terms of packet reception because it occurs while the other nodes are still receiving and buffering the last sent packet and so no communications should happen.

The synchronization procedure will be clarified in the next section.

4.3.2 Synchronization procedure

Before explaining how the synchronization was implemented, there are some considerations about the timers’ usage that Table 4.2 clarifies.

Since timer T1 is the only 16-bit timer available, it was chosen to be used as a “master” timer because it allows to count time in the order of a µC duration with a sufficient amount of resolution[3]. It will be used to synchronize the events of TAW_{time} and $BS\text{-}tx_{time}$ (Beacon Synchro transmission time).

[3] T1’s resolution ≈ 0.079 ms

Timer T4 will be used to implement the proposed clock drift compensation algorithm (Section 3.4.5), working alongside with T1 during the synchronization window. For the sake of simplicity, it's assumed in the following timing diagrams that T1's synchronization implies drift compensation using T4 to perform the time stamps (as in Figure 3.11), even though it is not explicit in the figures.

Timers' usage in synchronization and state transitioning events

	T1	**TMAC**	**T4**
TAW TIMEOUT (TAW_{time})	●	-	-
TX TIMEOUT			
- $BS\text{-}tx_{time}$	●	-	-
- $B2B\text{-}tx_{time}$	-	●	-
Timestamp / Drift Comp.			●

Table 4.2 – Timers' usage in synchronization and state transitioning events (Beacons)

An example of how the synchronization works in a type 3 µC is depicted in Figure 4.3. When a BS packet coming from a beacon whose time slot precedes the receiver's slot, T1 is used to synchronize the $BS\text{-}tx_{time}$, overriding whatever is the current T1's final counter value (at t_1 in Figure 4.3).

After sending a BS, T1 is set with its default TAW_{time} value relative to its own BS time slot (equation (14), substituting *rxID* by the beacon's ID) and waits for in RX State (t_2). If another BS packet is then received, the beacon will use T1 to synchronize the TAW_{time}, and overrides the current T1's final counter value (at t_3 in Figure 4.3).

When the TAW's execution is finished (at t_4 in Figure 4.3), TMAC is set to count the number of B2B slots that precede the beacon's slot, generating a TX-TIMEOUT ($B2B\text{-}tx_{time}$) when done.

The remaining time from the B2B packet transmission to the next TX-TIMEOUT ($BS\text{-}tx_{time}$) is again configured as a T1 timeout (t_5) and the beacon will wait for it in RX State.

From the TAW's ending (t_4) to the instant a new BS packet is received (t_6) no synchronization happens.

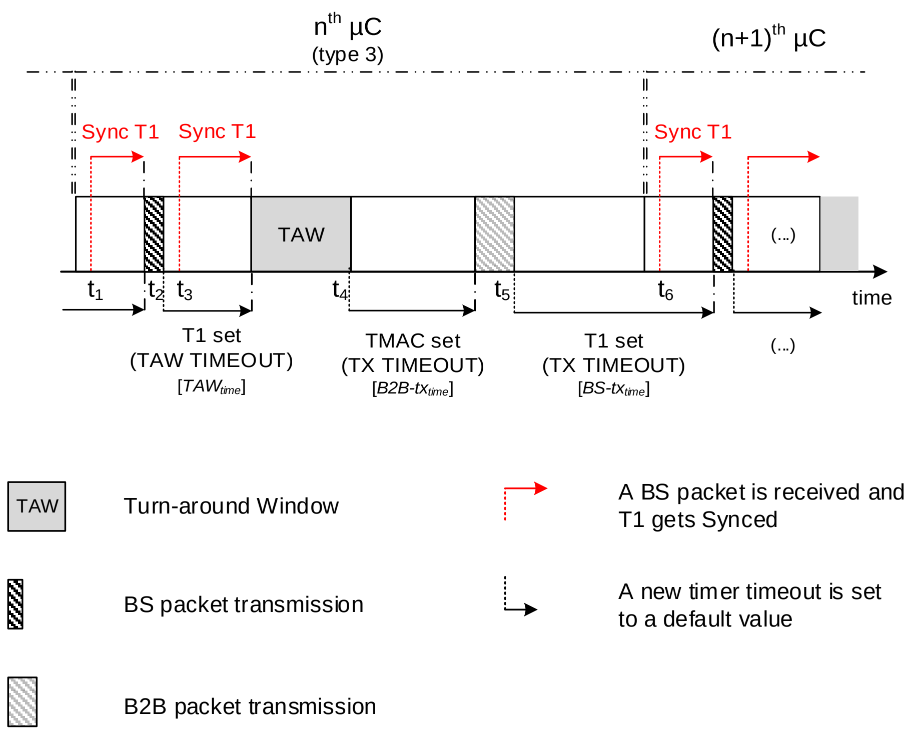


Figure 4.3 – Beacon's synchronization procedure during a type 3 µC.

The procedure during a non-type 3 µC is slightly easier because during the VTW beacon stays all the time in the RX state. An example is shown in Figure 4.4.

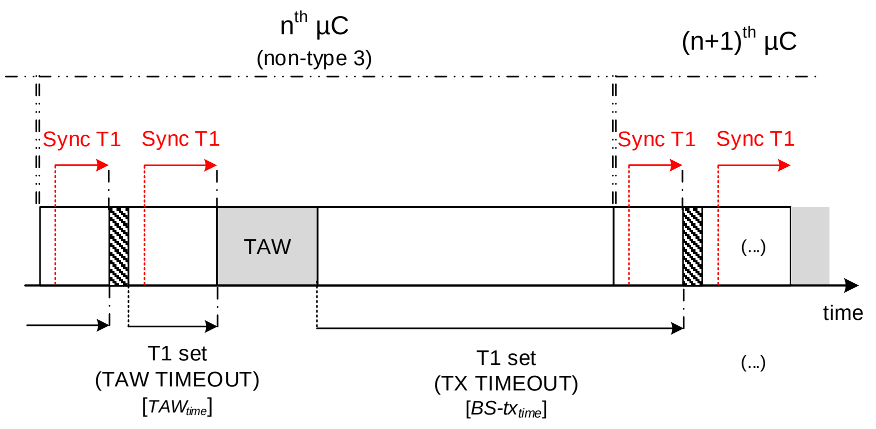


Figure 4.4 - Beacon's synchronization procedure during a non-type 3 µC. Legend in Figure 4.3.

After the TAW's ending, T1 is set with the next *BS-tx*$_{time}$. The procedure during the first window is the same as for a type 3 µC.

4.3.3 Beacon's states – Flowcharts

In this section, we'll take a look into each individual state. We'll start with the RX State where we'll meet the process of receiving / accepting a packet. About the TX State we'll justify the importance of having more than one tx-buffer. Finally, in the last subsection, the most complex state (IDLE) is presented where we'll pay special attention to the TAW's operations.

4.3.3.1 RX and TX States

RX State

The RX State workflow is typified in Figure 4.6 (a), page 51. The RX buffer needs to be cleared first. The radio will wait for a packet before moving to the next stage where the message type is decoded. Based on that, a *Get Packet* function is called, or the packet is simply ignored if the message type is unknown. These *Get Packet* functions do a series of consistency check (packet size, CRC, ID's, etc.). Moreover, the *Get Packet – BS* includes the synchronization algorithm.

While the radio is waiting for a packet to be received, an **RX-OVERFLOW** error may occur. This error is due to the use of variable size packets which force the radio module to interpret the first data field byte as a *packet length field*. In the presence of noise, this byte may be misinterpreted causing the DMA controller to stop transferring bytes before the noisy packet is fully "received", leading to an overflow if more noisy bytes are sensed later.

The RX State's loop can only be broken by a timer interrupt. Once it does, the respective timer *ISR* function is executed and the next state is called from there.

- **T1's Synchronization (executed inside Get Packet-BS)**

A code snippet of T1's synchronization is presented in Figure 4.5.

```
/* Sync BS */
if ( (rxID < thisID) && bs_updated) {
    tx_time = timeTo_My_BS_Slot(rxID, thisID);             // remaining time to my slot
    nextSMstate_after_T1Timeout(STATE_TX);                 // Go to TX after T1's ISR
    T1_count( tx_time );                                   // Update T1's counter
}
/* Sync TAW */
else {
    taw_time = timeTo_TAW(rxID, thisID, MAX_BEACONS);  // remaining time to TAW
    nextSMstate_after_T1Timeout(STATE_IDLE);               // Go to IDLE after T1's ISR
    T1_count( taw_time );                                  // Update T1's counter
}
```

Figure 4.5 – Code snippet for T1's synchronization (Beacons)

The variable *rxID* refers to the ID field of the received BS packet (the sender's ID) while *thisID* is the receiver beacon's ID. The flag *bs_updated* only gets to be true inside the first TAW whose start of execution was synchronized with another beacon. This prevents the beacon from sending a BS packet created prior to the reception of the first Beacon Synchro (as explained in the introductory Section 4.3.1). The constant *MAX_BEACONS* refers to the maximum number of beacons allowed in the network, which will affect the SW_{length} (equation (1)) and consequently the amount of time needed to wait for TAW_{time}. The drift compensations are implied in the *timeTo()* functions.

TX State

A packet stored in a tx-buffer is sent via the radio link while in TX State, Figure 4.6 (b).

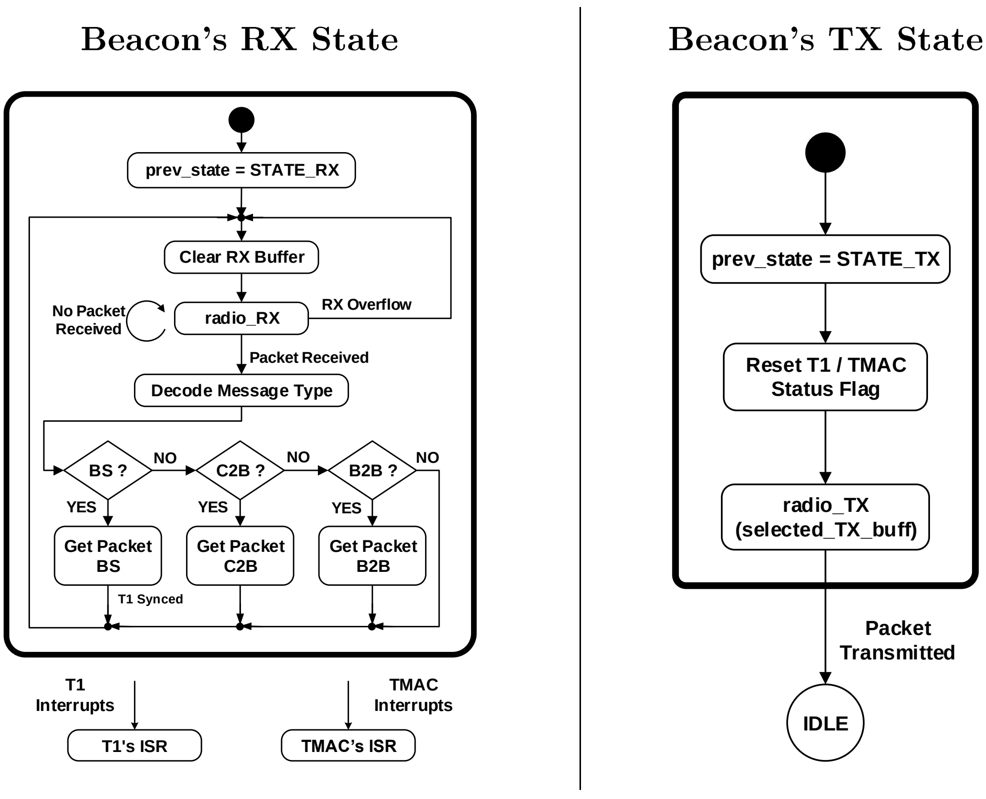
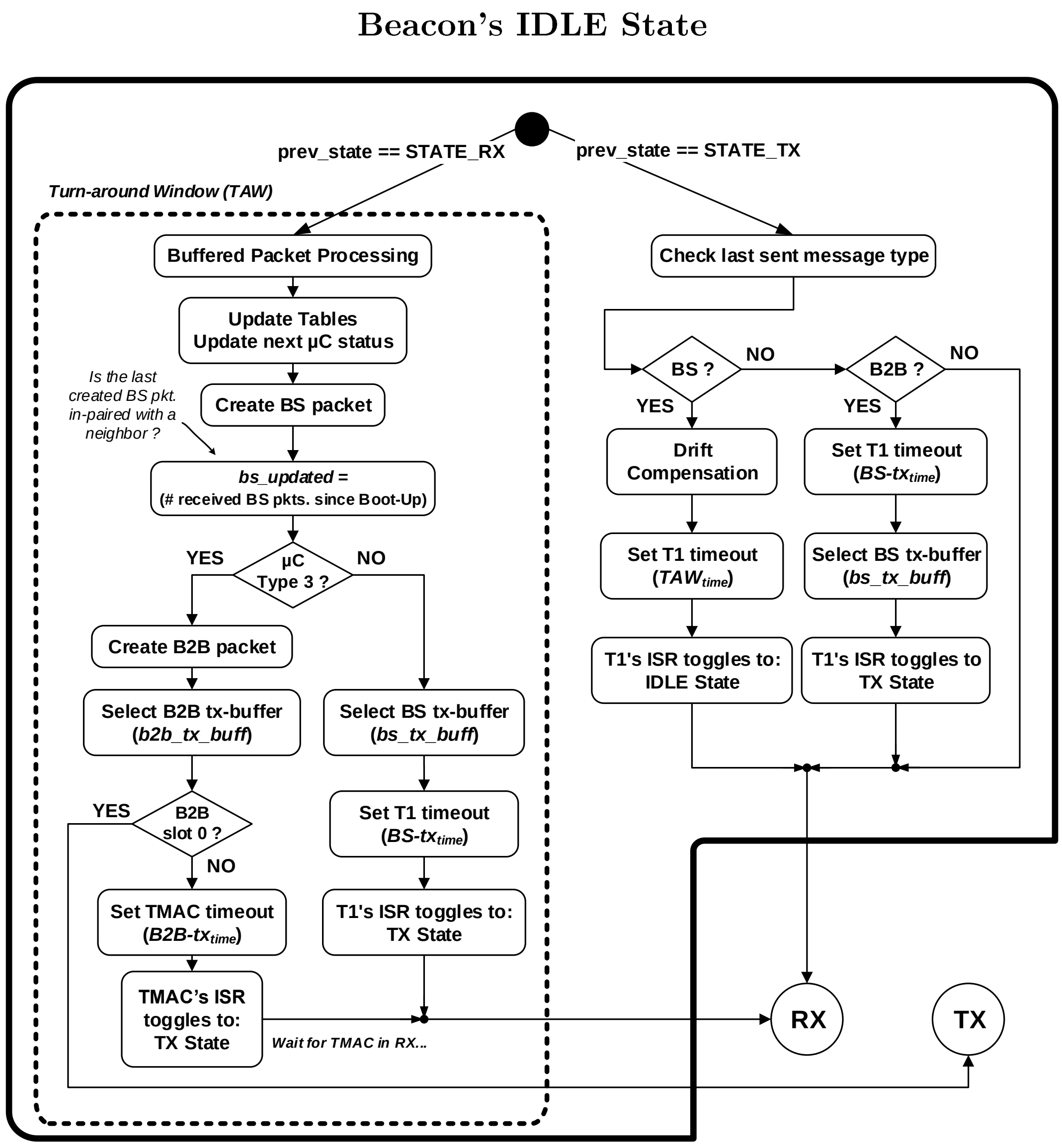


Figure 4.6 – Beacon's RX (a) and TX (b) State flowcharts

Because beacons may need to send two packets (BS and B2B) in the two subsequent windows after their creation in the TAW, two tx-buffers are needed to temporarily store them before they can be transmitted. The selection of the buffer to be used as the DMA controller's data source (*selected_TX_buff*) to feed the RFD Register is done in the IDLE State (Section 4.3.3.2).

4.3.3.2 IDLE State

When the IDLE State (Figure 4.7) is preceded by the RX State, a TAW-TIMEOUT was triggered to initiate the TAW's operations.



Beacon's IDLE State
prev_state == STATE_RX
prev_state == STATE_TX
Turn-around Window (TAW)
Buffered Packet Processing
Update Tables
Update next µC status
Is the last created BS pkt. in-paired with a neighbor ?
Create BS packet
bs_updated = (# received BS pkts. since Boot-Up)
YES
µC Type 3 ?
NO
Create B2B packet
Select B2B tx-buffer (b2b_tx_buff)
Select BS tx-buffer (bs_tx_buff)
YES
B2B slot 0 ?
NO
Set T1 timeout (BS-tx_time)
Set TMAC timeout (B2B-tx_time)
T1's ISR toggles to: TX State
TMAC's ISR toggles to: TX State
Wait for TMAC in RX...
Check last sent message type
BS ?
NO
YES
B2B ?
NO
YES
Drift Compensation
Set T1 timeout (BS-tx_time)
Set T1 timeout (TAW_time)
Select BS tx-buffer (bs_tx_buff)
T1's ISR toggles to: IDLE State
T1's ISR toggles to TX State
RX
TX

Figure 4.7 – Beacon's IDLE State flowchart

The TAW starts with the processing of the last buffered packets followed by the update of the beacon's internal tables and the update of the next µC type based on the current known order. A BS packet is then created. If at least one Beacon Synchro was received since Boot-Up, it means that the BS packet created during this TAW was conceived after acknowledging the current network status (µC order decision was already made after listening to the neighbors' announcements). Hence, permission is granted to synchronize the

BS transmission time during the next SW (*bs_updated* gets to be true).

If the current µC is of type 3, a B2B packet is created, consisting in the filling of this message header and the attachment of the collar's notifications table to be relayed. The buffer in which this packet is stored is selected to be the DMA's data source. The beacon that owns the first VTW slot only needs to wait for TAW's ending to switch to TX. If a different slot was assigned, TMAC is configured with the beacon's $B2B\text{-}tx_{time}$ prior to switch to RX State to receive the B2B packets of preceding slots. If the current µC is of another type, the buffer holding the last created BS packet is selected instead. Timer T1 is configured with the next beacon's $BS\text{-}tx_{time}$ and the node waits for it in RX State.

If IDLE State was preceded by TX, the last transmitted packet type is checked. If it was a BS, the upcoming TAW-TIMEOUT (TAW_{time}) is prepared. A time stamp is also done and clock drifts are evaluated and compensated before updating T1 (equation (14), substituting *rxID* by the ID of this beacon as if it has received a packet from itself). If the last transmitted packet was a B2B, then the next timeout will be a $BS\text{-}tx_{time}$. Hence, T1 is configured with the $BS\text{-}tx_{time}$ relative to the beacon's B2B slot, and the tx-buffer holding the BS packet created in the previous TAW is chosen to be the DMA's data source.

4.4 Collar's firmware

Following the same presentation strategy of Section 4.3 - Beacon's firmware, we'll start with a brief description of the collar's state-machine (Section 4.4.1). Because of its complexity, the state's transitions are better explained with some timing diagrams examples shown in Section 4.4.2. Lastly, it will be the moment to dive into each individual state and reveal their innards in Section 4.4.3.

4.4.1 State-machine

The collar's state-machine is depicted in Figure 4.8. It is composed of seven major states plus the initial state (Boot-Up). Most of the action will take place on the three middle states of Figure 4.8. They will serve the same purposes of their beacons' counterpart: RX for listening to the network, TX to send data and IDLE to do some processing when no communications are happening.

The initial hardware and peripheral configurations are done in the **Boot-Up** state. A unique ID is also statically assigned to each collar.

The remaining four states are a result of some hardware constraints and the necessity of lowering the power consumption without compromising the other two biggest concerns: keep the synchronization stable and being able to monitor / react to the sheep's behavior. We'll introduce them as some pitfalls will naturally arise when describing the state-machine from a simplistic perspective directly inferred from the system's architecture.

After Boot-Up, the collar switches to **RX** State. It will be stuck in this state as long as no BS packets are received. Once it happens, the TAW_{time} is synchronized and when the **TAW-TIMEOUT** is triggered the collar swaps to the **IDLE** State to begin the execution of the TAW's operations.

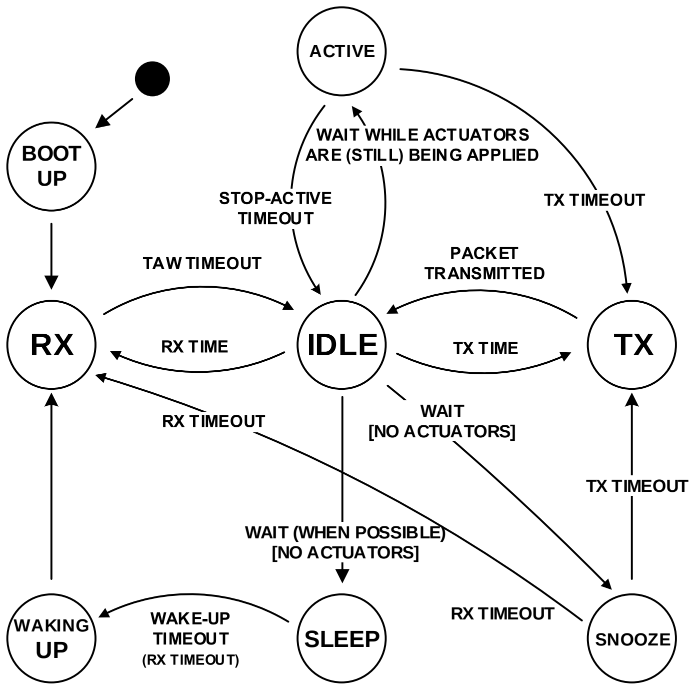


Figure 4.8 – Collar's state-machine

In an optimistic approach, the sheep's behavior would always be perfectly fine and no actuators would be needed to apply. Thus, after the TAW's ending, only two options could be considered: wait for the C2B time slot to send its data (during a type 2 µC) or wait for the next SW (during a non-type 2 µC). These situations were already addressed in the energy efficiency awareness section of Chapter 3 (Figure 3.12 and Figure 3.13).

The Texas Instruments' CC1110 offers a low power consumption mode that can be programmed to switch back to the fully operational mode after a certain amount of time. At a first glance, this seems the perfect choice to set up our collar when dealing with those low power waiting times. Yet, the first pitfall appears. This C1110's low power consumption mode requires the waking timer to use the Low Power RC Oscillator[4] whose accuracy isn't suitable with our TDMA scheme[5]. However, this SoC offers a second power consumption mode whose waking timer can be any of the system's timers clocked by the preferable High-

[4] The CC1110 SoC also permits the usage of a second crystal oscillator for this purpose, but its start-up time is around 400 ms [60], thus it was not considered as an option to clock the waking timer (Sleep Timer).

[5] CC1110's datasheet states an accuracy of ± 1% [60]. Experimentation tests revealed a maximum drift of around 65 ms for sleeping periods of around 6 seconds.

Speed Oscillator[6]. Even though it is not as good as the first option in terms of power consumption, this "*semi*" low power mode can be set up to reduce the energy consumption while waiting for the C2B time slot without jeopardizing its timing accuracy. Figure 4.9 depicts the collar's duty-cycle in type 2 µC's with the introduction of the semi low power consumption mode.

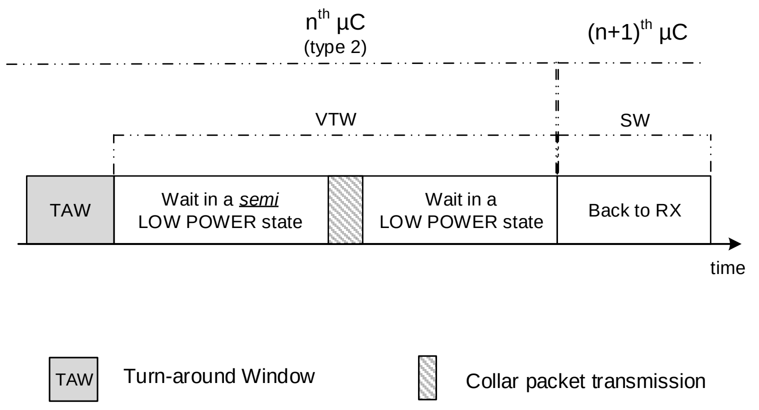

nth µC
(type 2)
(n+1)th µC
VTW
SW
TAW
Wait in a semi LOW POWER state
Wait in a LOW POWER state
Back to RX
time
TAW Turn-around Window
Collar packet transmission

Figure 4.9 – Remake of Figure 3.12 with the introduction of the "semi" Low Power mode

Meanwhile, the "true" Low Power (LP) mode can still be used while waiting for the next SW because we can compensate the low accuracy of the waking timer by shortening the *sleeping time*, ensuring that the collar will switch back to the RX State a little before the SW starts.

Table 4.3 compares the power consumption of CC1110's power modes that SheepIT's protocol makes use of.

Power Consumption modes addressed by SheepIT

	Consumption
Active power mode (with low CPU activity)	5.0 mA
Low Power (LP) mode (as PM2 in datasheet)	0.5 µA
"Semi" LP mode (as PM0 in datasheet)	4.3 mA

Table 4.3 – Power consumption modes [60]

A reduction of at least 14 % of energy consumption is achievable with the introduction

[6] The High-Speed Crystal Oscillator has an accuracy of ± 40 ppm [60].

of the “semi” LP without compromising the TDMA schedule.

As the reader is probably be guessing by now, these two power-saving sates are the **SLEEP** (“true” LP mode) and **SNOOZE** (“semi” LP mode), depicted in the state-machine of Figure 4.8 (page 54). Because the SLEEP State uses a different clock source than the rest of the states, we need to re-configure the default High-Speed Oscillator just as soon as the waking timer is triggered. This is done under the **WAKING-UP** State that automatically transits to RX after restoring the clock source.

Another hardware limitation can restrict even more the usage of the SLEEP State. The waking timer has a minimum timeout value (11.72 ms) [60] that unfortunately is in the order of a few C2B time slots. This means that the collars who own a time slot near the edge between the VTW and SW windows will not have enough time to SLEEP after their time slots. In situations like these, a collar will never enter the SLEEP State during a type 2 µC and the waiting time between their slot and the beginning of the next SW will also be elapsed in SNOOZE. Hence, besides preceding the transition to TX, the SNOOZE State can also toggle to RX to address this exceptional case.

In a more realistic approach the state-machine must be prepared to deal with the application of actuators that will preclude or limit the usage of the power saving modes. The actuators application decision is made right after the sensors reading (during TAW) and they can last beyond the TAW’s ending. Because it might be reasonable to have full control of the actuators while they are being applied, the collar is prevented from entering a low power mode until the actuators application is finished. Therefore, the **ACTIVE** State was implemented. It simply halts the program’s execution after TAW’s ending in an infinite loop for a desired time duration with the SoC running in active power mode. Withal, the loop can be broken earlier if a **TX TIMEOUT** occurs in the meantime. When it happens, after the packet transmission, the IDLE State returns the execution back to ACTIVE so the collar keeps fully awake during the remaining time for the **STOP-ACTIVE TIMEOUT**. After this timeout, the collar can rest in SLEEP / SNOOZE depending on how much time it remains for the next SW.

If some state transitions still look confusing, we hope to clarify them in the following section with some more detailed timing diagrams as the synchronization procedure is also disclosed.

4.4.2 State transitions

This section focuses on the state transition events and the implementation of the synchronization algorithm.

4.4.2.1 Synchronization procedure

The chosen timers to execute the state transitions are laid in Table 4.4. As in the beacon's firmware implementation, T1 was also used as a "master" timer due to its 16-bit operation mode. It is used to synchronize the TAW_{time}, to program the transitions from SNOOZE to RX (when the waiting time is not enough to SLEEP) and from ACTIVE to IDLE. TMAC is used to count multiples of time slots, in this case C2B slots, and trigger an interrupt for TX TIMEOUT (*C2B-tx*$_{time}$). TSLEEP (timer T2) is the only timer able to wake-up the system from the LP consumption mode and thus it is only used for that purpose. Despite the potential usage of T4 for other purposes related with the posture control algorithm (as mentioned in Table 4.1, page 46), during the SW no other task besides the synchronization should be executed and so T4 is available to perform the required time stamps to apply the drift compensation algorithm.

Timers' usage in synchronization and state transitioning events

	T1	**TMAC**	**TSLEEP**	**T4**
TAW TIMEOUT (TAW$_{time}$)	●	-	-	-
TX TIMEOUT: C2B-tx$_{time}$	-	●	-	-
RX TIMEOUT				
- Wake-Up	-	-	●	-
- After Snooze	●	-	-	-
Stop Active	●	-	-	-
Timestamp / Drift Comp.	-	-	-	●

Table 4.4 – Timer's usage in synchronization and state transitioning events (Collars)

By examining the state machine, we can see that both TAW TIMEOUT and STOP-ACTIVE TIMEOUT redirect the program's execution to the IDLE State and they use the same timer to trigger the transition. To keep things clear, a variable is used (*T1_used_as*) to identify the T1's timer purpose while in the IDLE State to make a decision according to it. This variable should be set to one of the values laid down in Table 4.5.

T1 usage discrimination
(when switching to IDLE)

T1's timeout purpose	**T1_used_as (tag)**
TAW TIMEOUT	TAW_TIMEOUT
STOP-ACTIVE	STOP_ACTIVE

Table 4.5 – Definition of T1_used_as possible values when switching to IDLE state

Now that the timers' usage is introduced, we'll look to the synchronization procedure and the related state transitions in the following section.

4.4.2.2 State transition examples – timing diagrams

To avoid a tedious section describing every transition for every example, only some of them will be analyzed. The rest may already be intuitive enough, after the introductory state-machine description. The following examples use a color scheme to identify the collar's state through time. The state's colors scheme can be found in Figure 4.10.

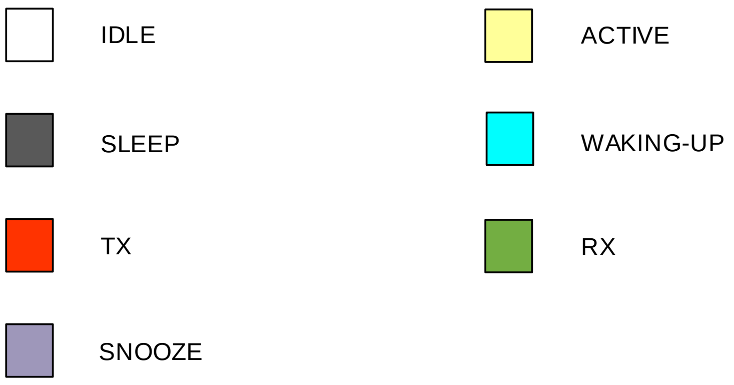


Figure 4.10 – Collar's states color scheme

Type 2 micro-cycles

- **Example 1:** **Collar sleeps after the C2B slot**

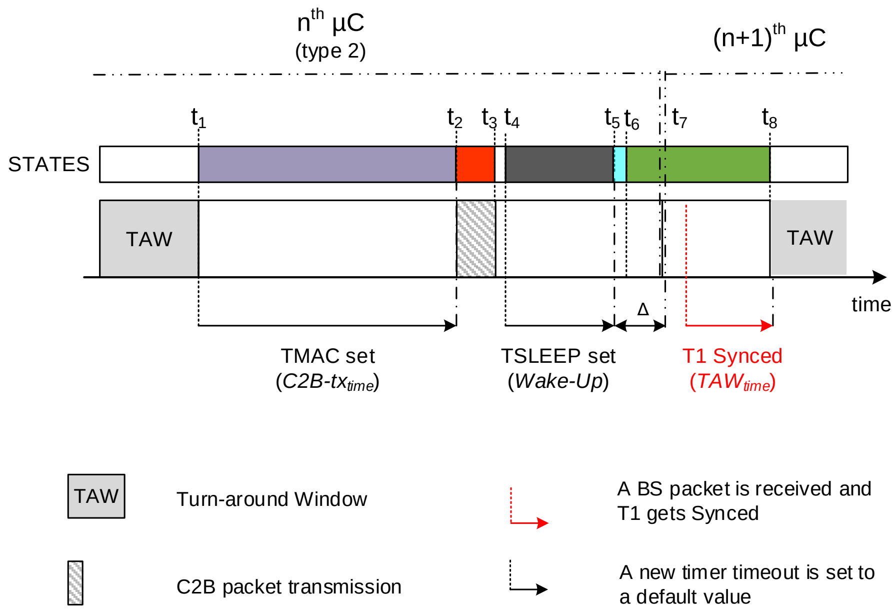


Figure 4.11 – Example 1: Collar sleeps after the C2B slot (type 2 µC)

At t_3 the packet is finished being sent and IDLE State is entered where TSLEEP is configured. At t_5 the collar awakes and switches to RX at t_6. Note that the collar awakes Δ units of time[7] earlier than the SW starts (t_7) to accommodate clock drifts due to TSLEEP's accuracy and low resolution[8].

- **Example 2:** **Not enough time to sleep – Collar snoozes**

The figure's example is in the next page. This time, after returning to IDLE (t_3) the collar goes to SNOOZE (t_4). The waking time (t_5) can be much more close to the SW beginning time (t_6) because T1's accuracy is much higher[9].

---

[7] $\Delta = 2 \times MAX\_OBSERVED_{TSLEEP\ drift} \Leftrightarrow \Delta = 130$ ms
($MAX\_OBSERVED_{TSLEEP\ drift}$ also addresses the execution time of the WAKING-UP State)

[8] As it was configured, TSLEEP has a resolution of ≈ 0.92 ms (a lot worse than T1's resolution ≈ 0.079 ms)

[9] T1 will only be used for this purpose when the *sleeping time* is below 11.72 ms (minimum TSLEEP timeout value). With an accuracy of ± 40 ppm, the maximum drift for a rounded *snooze time* of 12 ms will be 0.00048 ms. Nevertheless, T1's resolution is less than that: 0.079 ms.
Therefore, a conservative value for Δ was used when waking from Snooze: Δ = 1 ms

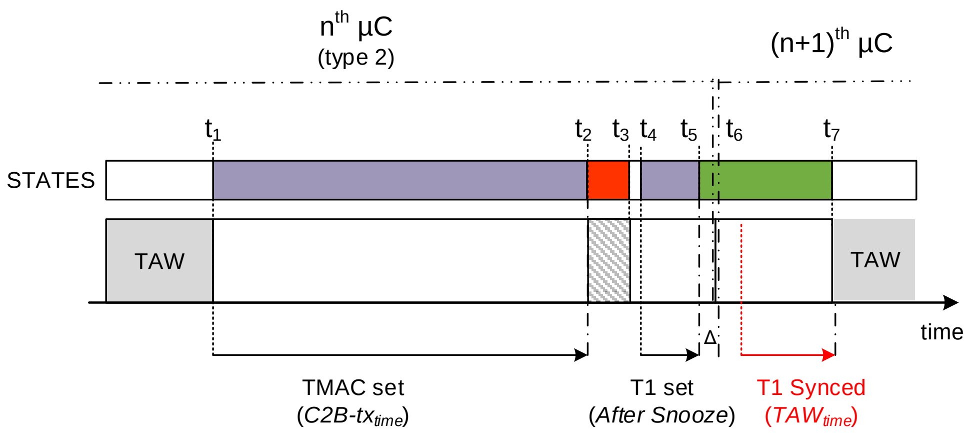


Figure 4.12 – Example 2: Not enough time to sleep – Collar snoozes (type 2 µC)

- **Example 3:** **Actuators applied before the C2B slot**

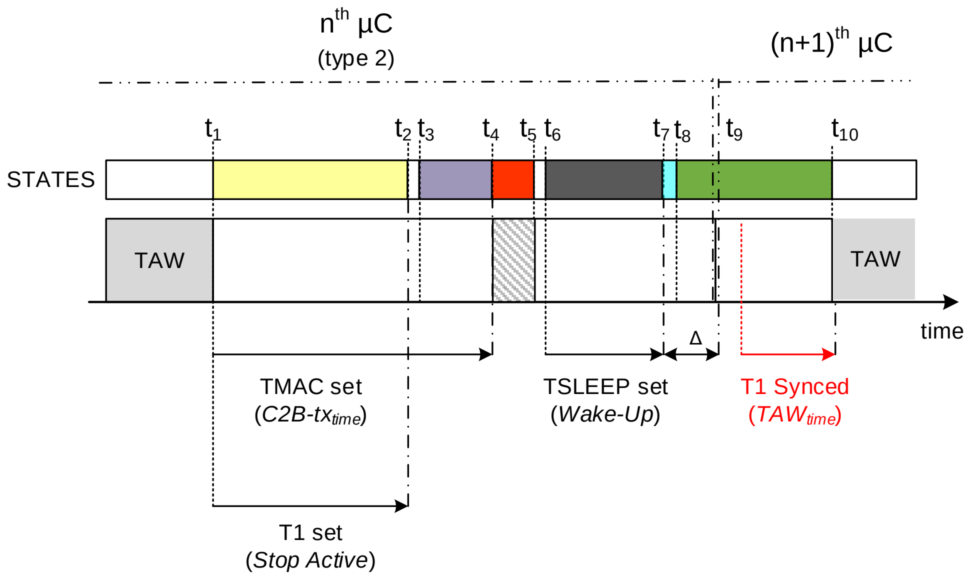


Figure 4.13 – Example 3: Actuators applied before the C2B slot (type 2 µC)

Because the actuators have started to be applied inside the TAW, after its ending the CPU continues in active mode as it switches to the ACTIVE State. Before this transition, at t_1, timer T1 is configured with the duration of the actuators application while TMAC is configured with the TX TIMEOUT as usual. Since the actuators cease to be applied at t_2 there's a transition to IDLE. As there were no signs of a TMAC interrupt, the IDLE State sends the execution to SNOOZE (t_3) so the collar can enter the "semi" LP mode as it waits for TMAC's ISR to switch to TX (t_4).

- **Example 4:** **Actuators application ceases while in TX**

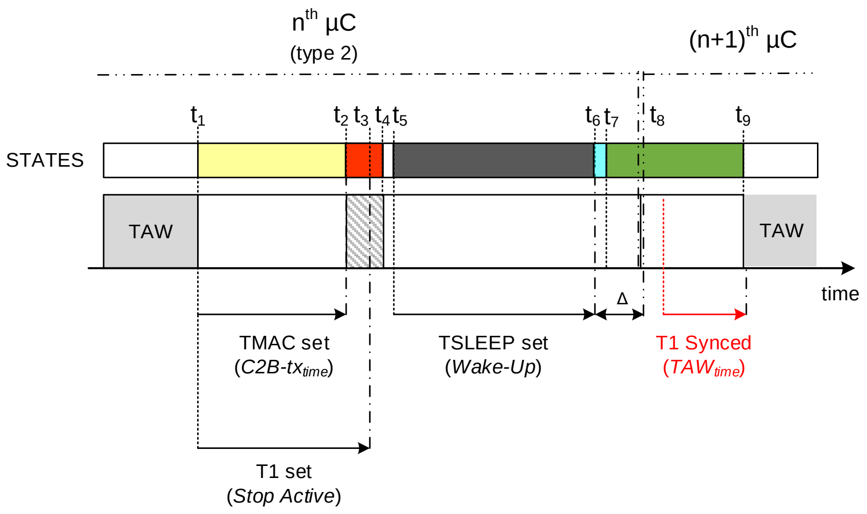


Figure 4.14 – Example 4: Stimulus application ceases while in TX (type 2 µC)

T1 is set to switch to IDLE at t_3 to stop the actuators application. However, at this time, the collar is in the middle of a packet transmission. To avoid a packet transmission abortion T1's interrupts must be disabled inside the TX State, so it can not break this state's normal functioning. Consequently, the IDLE State must check T1's *timeout-flag* each time the program's execution comes from TX. By doing it at t_4, it knows the actuators application should already be ceased and so the SLEEP state can now be called (t_5).

- **Example 5:** **TX interrupts the ACTIVE state**

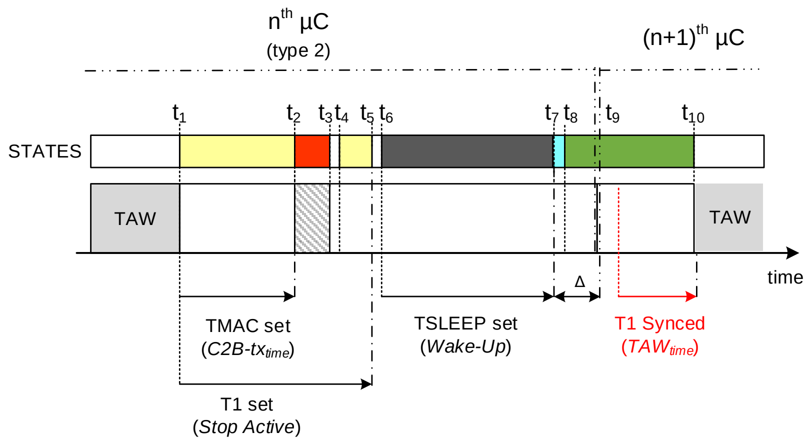


Figure 4.15 – Example 5: TX interrupts the ACTIVE state (type 2 µC)

At t_3, (Figure 4.15) the IDLE State knows T1's timeout-flag is still reset, meaning that not enough time has elapsed to cease the actuators application. So, ACTIVE State is called as the CPU continues to be in full active mode until the timeout to cease the actuators application is triggered (at t_5).

- **Example 6: Collar owns the first C2B slot**

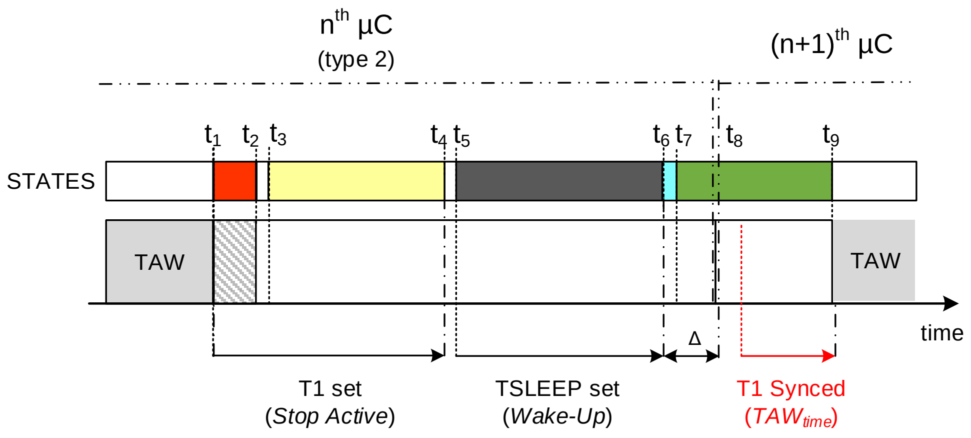


Figure 4.16 – Example 6: Collar owns the first C2B slot (type 2 µC)

After the TAW's execution, the IDLE State switches directly to TX.

In this example the collar has started to apply an actuator during the TAW and so, before the transition to TX, T1 is configured with the STOP ACTIVE TIMEOUT value.

After the packet transmission (t_2), the IDLE State knows that the collar should not SLEEP because STOP ACTIVE TIMEOUT wasn't triggered yet, and so it toggles to ACTIVE State to keep the CPU in full active mode.

If no actuators were applied, at t_2 the collar would have switched to SLEEP state instead.

Non-type 2 micro-cycles

- **Example 7:** **Collar sleeps after TAW's ending**

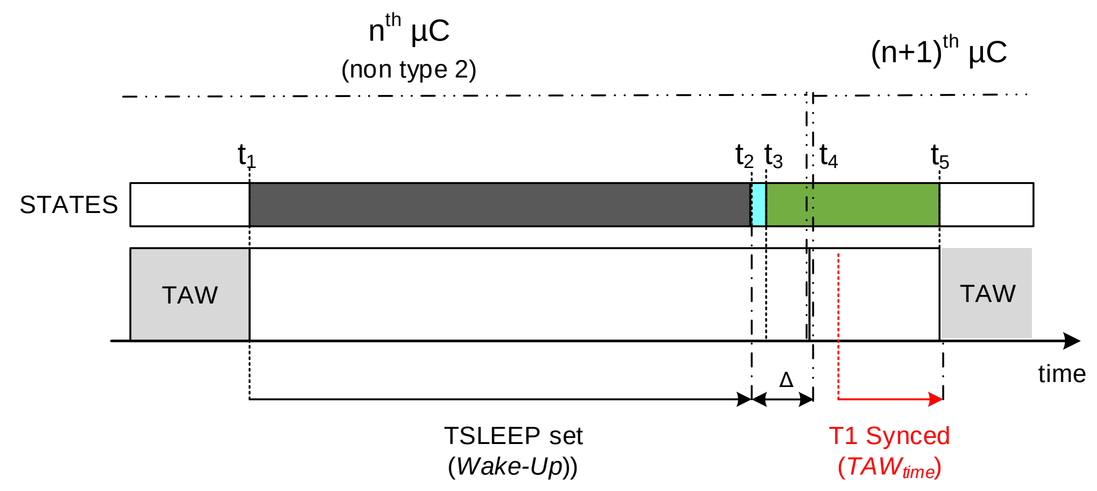


Figure 4.17 – Example 7: Collar sleeps after TAW's ending (non- type 2 µC)

Example 8: **Collar sleeps after the actuators application**

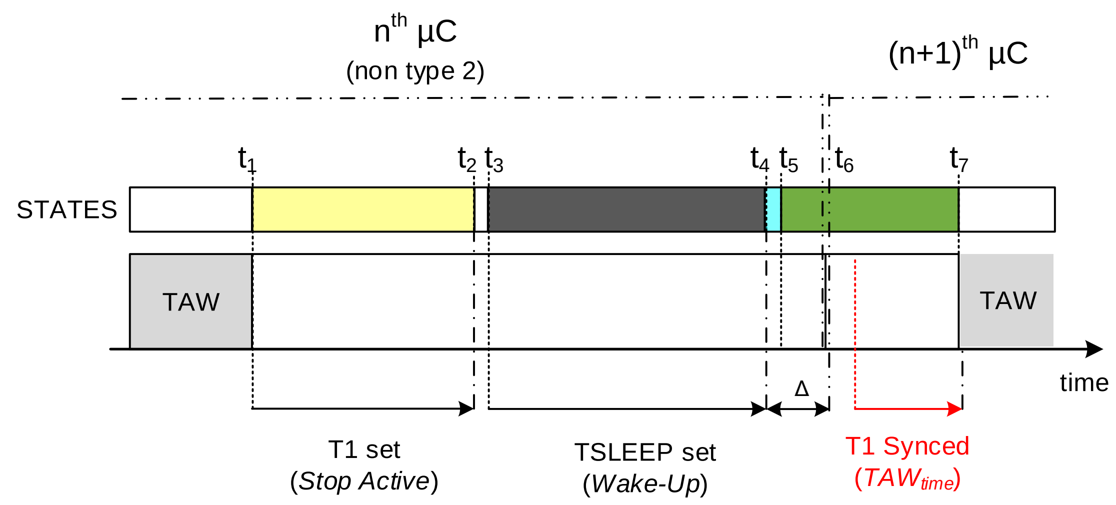


Figure 4.18 – Example 8: Collar sleeps after the stimulus application

4.4.3 Collar's states – Flowcharts

Because the Boot-Up, RX and TX states share a lot with their beacon's counterparts, we will skip most of its analysis, referencing the beacon's chapter anytime it's applicable. The SNOOZE, SLEEP and ACTIVE states are primarily time-consuming states, so their description is also brief. The WAKING-UP State barely does anything more besides just re-establishing the clock source, which, again, saves us some reading time. Thereupon, this chapter will be dedicated for the most part to the analysis of the IDLE State and in particular to the TAW's operations.

4.4.3.1 RX and TX states

RX State

Figure 4.20 (a) displays the flowchart of RX State. As opposed to the beacon's counterpart, the collar's version of this state only accepts BS packets. Another difference is that once it enters the state there is no timeout pre-configured to switch to another state. It must receive a Beacon Synchro to unlock the loop.

- **T1's Synchronization (executed inside *Get Packet-BS*)**

The collars' synchronization algorithm only synchronizes the beginning of the TAW's operations (TAW_{time}). A code snippet can be found in Figure 4.19.

The variable *rxID* refers to the announced beacon ID in the received BS packet whereas *thisID* is the collar's ID. The constant *MAX_BEACONS* is the total number of beacons that can be deployed which affects the SW size (equation (1)).

The drift compensation is implicitly present in the *timeTo_TAW()* function where T4 is used to perform the time stamps to compute the spanned times between BS transmissions.

```
// ---------------------------------------
/* Sync TAW */
taw_time = timeTo_TAW(rxID, thisID, MAX_BEACONS);    // remaining time to TAW
nextSMstate_after_T1Timeout(STATE_IDLE);             // Go to IDLE after T1's ISR

T1_used_as = TAW_TIMEOUT;                            // Timeout purpose when in IDLE
                                                     // next time [TABLE 4.5]

T1_count( taw_time );                                // Update T1's counter
```

Figure 4.19 – Code snippet for T1's synchronization (Collar)

TX State

Since collars can only send one message per µC, one single tx-buffer is enough to hold the packet ready to be transmitted.

In Figure 4.20 (b), we highlight the T1's interrupts disablement as we don't want the packet transmission to be interrupted by a possible STOP ACTIVE TIMEOUT (example 4, Figure 4.14 of page 61).

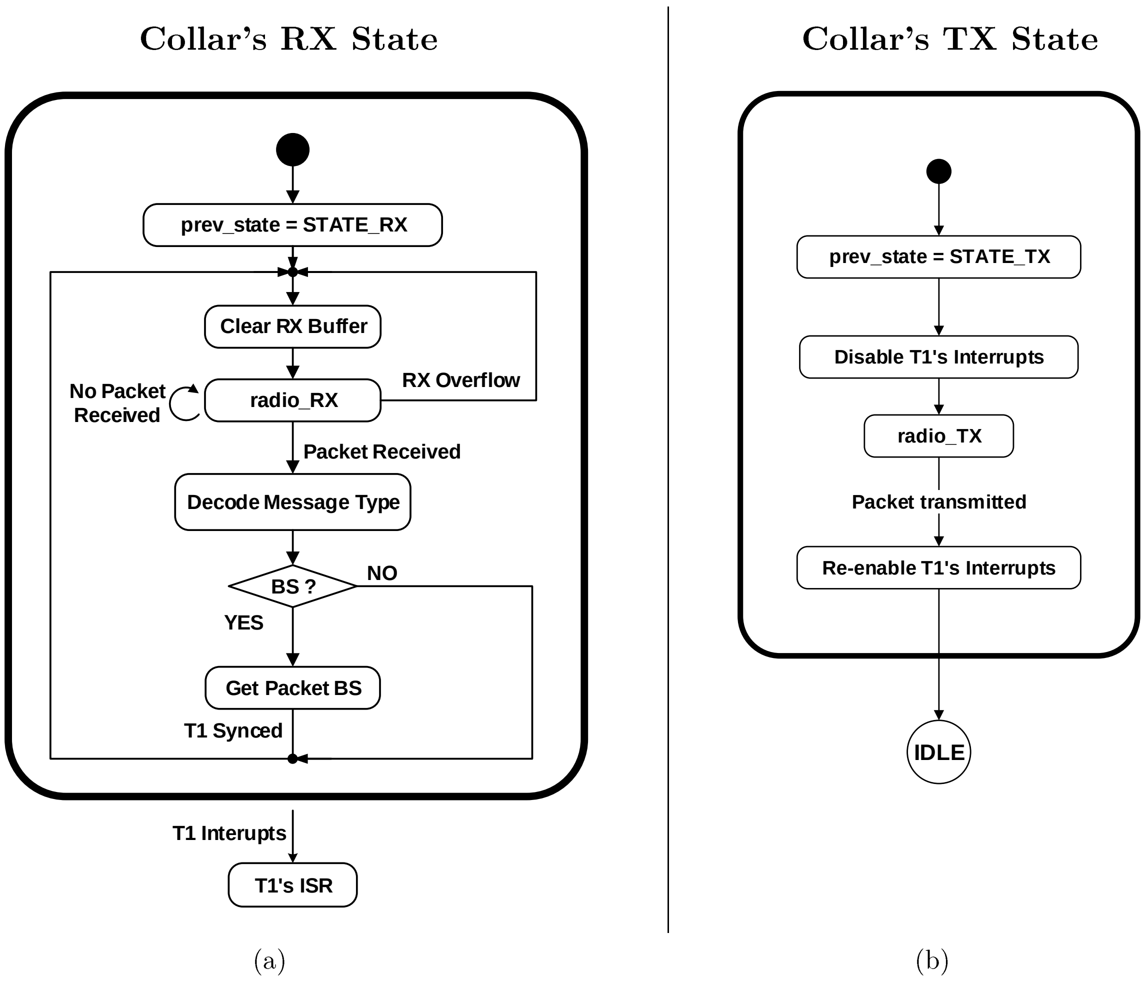


Figure 4.20 – Collar's RX State (a) and TX State (b) flowchart

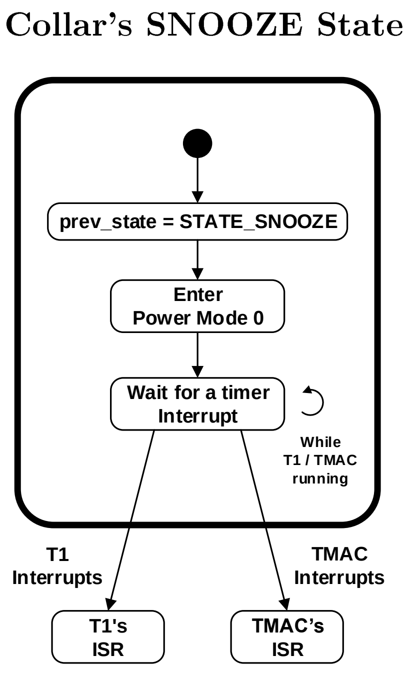
4.4.3.2 SNOOZE, SLEEP, WAKING-UP and ACTIVE States

Snooze State

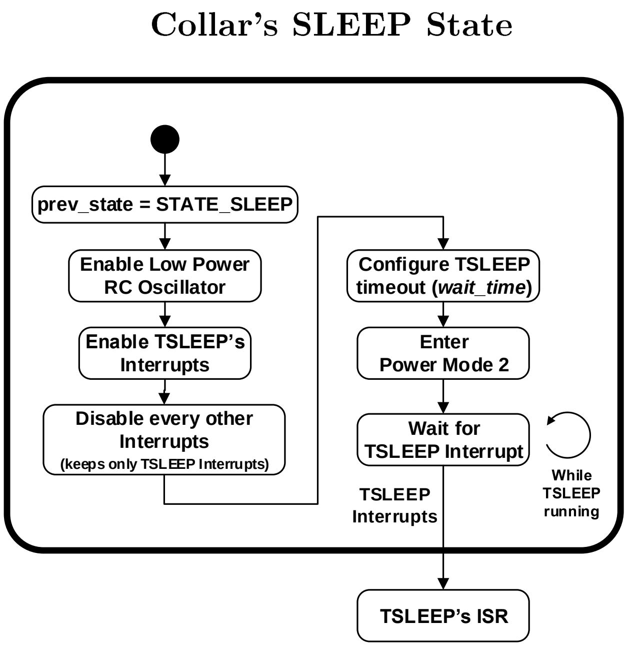
When snoozing, Figure 4.21 (a) of next page, the collar enters the CC1110's Power Mode 0 (as stated in its datasheet [60]). Even though the CPU is actually "frozen", we can view this power mode state as an infinite loop broken by one of the timers' interrupts: T1 (RX TIMEOUT) or TMAC (TX TIMEOUT).

Sleep State

SLEEP State, Figure 4.21 (b), starts with the set-up of the Low Power RC Oscillator (shuts off the default High Speed Osc.). Prior to configure the TSLEEP timer, other interrupts are disabled, as recommended by SoC's datasheet. The timeout configuration uses a global variable (*wait_time*) to state the sleep-time duration. Once the CPU enters the Power Mode 2 (datasheet nomenclature), TSLEEP timer is activated. This state halts the CPU and it can be viewed as an infinite loop that can only be broken by a TSLEEP interrupt. Once it does, TLEEP's ISR will call the WAKING-UP State.



(a)



Collar's SLEEP State
prev_state = STATE_SLEEP
Enable Low Power RC Oscillator
Enable TSLEEP's Interrupts
Disable every other Interrupts
(keeps only TSLEEP Interrupts)
Configure TSLEEP timeout (wait_time)
Enter Power Mode 2
Wait for TSLEEP Interrupt
While TSLEEP running
TSLEEP Interrupts
TSLEEP's ISR

(b)

Figure 4.21 – Collar's SNOOZE State (a) and SLEEP State (b)

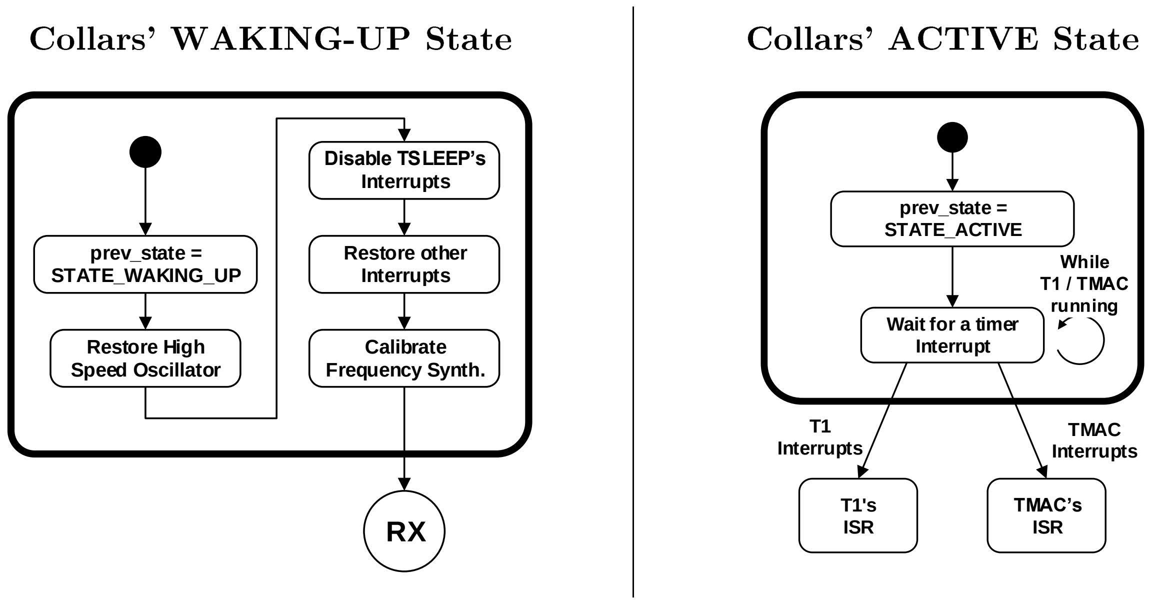
Waking-Up State

The WAKING-UP State, Figure 4.22 (a) of next page, restores the default High Speed Oscillator. The interrupts disabled during the SLEEP State can now be re-enabled. The Frequency Synthesizer (FS) also needs to be calibrated. At the end, RX State is called by default.

Active State

This state, Figure 4.22 (b), consists in an infinite loop with the CPU running in active mode. Some actuators control code can be put inside this loop to monitor its functioning.

Two timers can interrupt the loop: T1 (to stop the execution of this state and return to IDLE) and TMAC (to switch to TX).


Collars' WAKING-UP State
prev_state = STATE_WAKING_UP
Restore High Speed Oscillator
Disable TSLEEP's Interrupts
Restore other Interrupts
Calibrate Frequency Synth.
RX
Collars' ACTIVE State
prev_state = STATE_ACTIVE
Wait for a timer Interrupt
While T1 / TMAC running
T1 Interrupts
TMAC Interrupts
T1's ISR
TMAC's ISR

(a) (b)

Figure 4.22 – Collars' WAKING-UP State (a) and ACTIVE State (b)

4.4.3.3 IDLE State

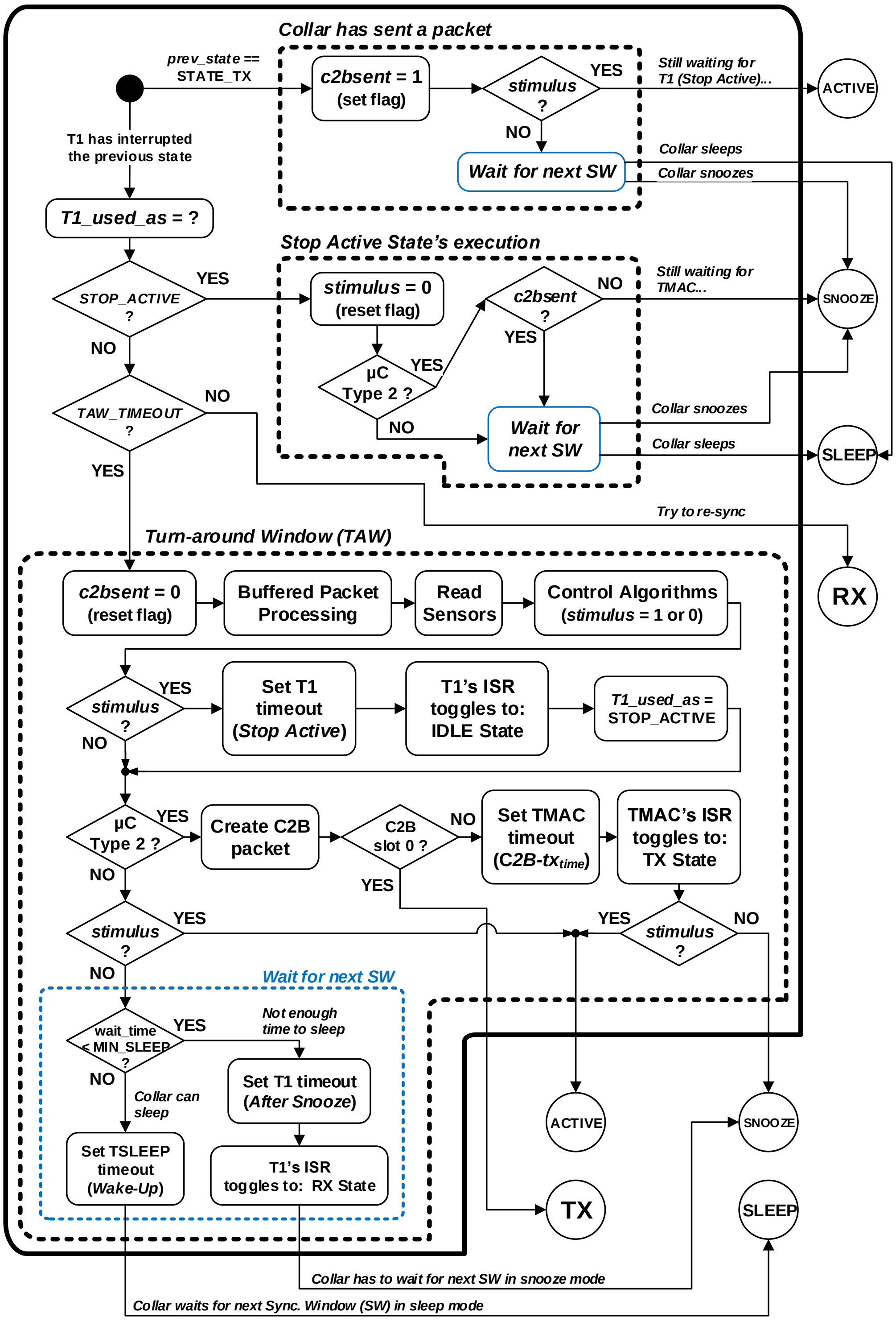
The IDLE State flowchart is portrayed in Figure 4.23, in the next page. It starts to verify the event that has triggered its execution: coming from TX State (after a C2B packet transmission) or a T1 timeout.

If it was called by T1's ISR (an interrupt occurred while executing other state), *T1_used_as* value is checked to determine which of the two possible transitions has occurred (Table 4.5, page 58).

If a TAW-TIMEOUT is acknowledged, the TAW is executed. It begins with a set of procedures that are independent of the µC type. A flag announcing the transmission of a C2B packet is reset (*c2bsent*) to keep track of a possible transition to the TX State in the subsequent window. The received BS packets are processed in order to determine the current µC type. After that, sensor reading takes place and the control algorithms are performed which include the posture control [7] and virtual fencing. These algorithms may, or not, initiate an actuator application. When it happens, *stimulus* is set to 1, otherwise it is 0. If a stimulus has started to be applied, T1 is configured with a timeout value as being the maximum time the collar should remain in the ACTIVE State after TAW's ending.

When the current µC is of type 2, a C2B packet is created to announce the collar's status. If the collar does not possess the first VTW slot, TMAC is configured with the $C2B\text{-}tx_{time}$ timeout, otherwise the collar can switch directly to TX State after TAW's ending.

Collar's IDLE State



Collar has sent a packet
prev_state == STATE_TX
c2bsent = 1 (set flag)
stimulus ?
YES
NO
Still waiting for T1 (Stop Active)...
ACTIVE
Wait for next SW
Collar sleeps
Collar snoozes
T1 has interrupted the previous state
T1_used_as = ?
Stop Active State's execution
STOP_ACTIVE ?
stimulus = 0 (reset flag)
c2bsent ?
Still waiting for TMAC...
SNOOZE
µC Type 2 ?
Wait for next SW
Collar snoozes
Collar sleeps
SLEEP
TAW_TIMEOUT ?
Try to re-sync
RX
Turn-around Window (TAW)
c2bsent = 0 (reset flag)
Buffered Packet Processing
Read Sensors
Control Algorithms (stimulus = 1 or 0)
stimulus ?
Set T1 timeout (Stop Active)
T1's ISR toggles to: IDLE State
T1_used_as = STOP_ACTIVE
µC Type 2 ?
Create C2B packet
C2B slot 0 ?
Set TMAC timeout (C2B-tx_time)
TMAC's ISR toggles to: TX State
stimulus ?
stimulus ?
Wait for next SW
wait_time < MIN_SLEEP ?
Not enough time to sleep
Set T1 timeout (After Snooze)
Collar can sleep
Set TSLEEP timeout (Wake-Up)
T1's ISR toggles to: RX State
ACTIVE
SNOOZE
TX
SLEEP
Collar has to wait for next SW in snooze mode
Collar waits for next Sync. Window (SW) in sleep mode

Figure 4.23 – Collar's IDLE State flowchart

Either when the µC is not of type 2 either when it is and the collar has to wait for its slot after TAW's ending, the stimulus application is again verified because the next state transition is depending on it. If a stimulus has started to be applied, the collar will switch to ACTIVE State at the end of the TAW. If no stimulus were applied and the collar has to wait for its C2B slot, it will do it in SNOOZE State. If no stimulus were applied and the µC is not of type 2, the subsequent VTW frame will be elapsed in a low power mode as the collar waits for the next Synchronization Window (*Wait for next SW* block). Ideally, the collar waits in the SLEEP State, unless the waiting time (*wait_time*) is less than what TSLEEP timer allows (11.72 ms), which is unlikely to happen when the waiting time is the complete VTW frame.

If the IDLE State has started its execution due to a STOP ACTIVE TIMEOUT, the code block associated with this action is executed. The flag signaling the application of an actuator (*stimulus*) is reset. For non-type 2 µC's, the *Wait for next SW* block (depicted inside the TAW) is executed in order to determine which low power mode the collar can enter to wait for the beginning of the next SW (*wait_time*). On the other hand, if the µC is of type 2 and the actuators ceased to be applied prior to the C2B packet transmission, the collar is forced to switch to SNOOZE State as it waits for TMAC to trigger a TX TIMEOUT. Notwithstanding, if the packet was already transmitted the collar may switch to SLEEP State if the *wait_time* for the next SW allows it to do.

When IDLE State is called after a packet transmission, *c2bsent* flag is raised. If an actuator was being applied when TMAC interrupted the ACTIVE State (*stimulus* is still set), the collar is forced to switch back to ACTIVE State. Otherwise, it will wait for the next SW (*wait_time*) in a low power state (*Wait for next SW* block determines which state).

One single transition is left. When T1 has interrupted the previous state and *T1_used_as* variable was not assigned with one of the two expected constants (Table 4.5 of page 58), the collar goes to RX State as an attempt of trying to re-establish its state-machine normal functioning.

Virtual Fence Algorithm

A simple Virtual Fence Algorithm was implemented during this dissertation project, being executed inside the "Control Algorithms" block of the TAW. It consists in computing the maximum of all RSSI values retrieved for the received BS packets of the previous Sync Window and comparing it with a threshold (*RSSI_THRESHOLD*). This threshold is the RSSI value of the maximum distance a collar is allowed to be away from the closest beacon and it was experimentally measured. This Virtual Fence Algorithm is portrayed in Figure 4.24 and it serves only as prototype for more advanced RSSI based location algorithms that should be addressed as future work.

```
/* ###############   VIRTUAL FENCE ALGORITHM   ############### */

rssi_aray = getLastComputedRSSI();      // RSSI values are all negative - (dBm)
maxRSSI = findMaximum(rssi_array);      // Compute the maximum (the least negative element)

/* The collar is too far away from the closest beacon (OUT OF FENCE DETECTION) */
if ( maxRSSI < RSSI_THRESHOLD ) {
    vfInfractCounter++;                 // Number of Virtual Fence infracts. in a single row

    // >> Activate an actuator (buzzer, electric stimulus) with a predefined timeout <<
}

/* The collar is inside the fence */
else {
    vfInfractCounter = 0;
}
```

Figure 4.24 – Simple virtual fence algorithm (pseudocode)

4.5 Summary

The implemented prototype for SheepIT's WSN addresses the synchronous communications between nodes during µC's of type 2 and 3, as presented in Chapter 3. Dynamic node registration and pairing were not addressed in this implementation, so the inclusion of type 1 µC's is left as future work.

Real collar data can be gathered from collar devices and relayed by beacons. For that matter, two state-machines were designed: a beacon and a collar.

Beacons can process BS, B2B and C2B packets. The state-machine is based on three states that are directly related with the transceiver mode: RX (receiving), TX (transmitting) and IDLE. While in RX, received packets get buffered and synchronization algorithm is applied, in case a BS packet is received. The TAW's operations are executed in IDLE State, which includes the received packet processing and packet creation. The beacon switches to TX when it's time to send a packet.

Collars only decode BS packets and create C2B packets using the last read values from their sensors. The state-machine includes the same states present in the beacon's counterpart plus two power saving states (SLEEP and SNOOZE), an ACTIVE State for stimulus application monitoring and a WAKING-UP State. The collar enters in SLEEP when it is not required to perform any other actions besides just waiting. Exceptions to this happen while waiting for its time slot to send a C2B packet and when the waiting time is not sufficient to enter SLEEP (minimum amount of time is required). In both exceptions SNOOZE is entered instead. In the former case, SLEEP is not suitable because this state requires the usage of a different oscillator whose accuracy is too low to feed a timer to trigger a packet transmission instant. The WAKING-UP State restores the default oscillator, an operation required after SLEEP State. The ACTIVE State halts the collar's CPU in a full active power state and it is executed when a stimulus is being applied. Additional code can be added to this state in order to monitor the stimulus application. In respect to the other three states, the main difference between them and their beacons' counterpart lies in the TAW executed in IDLE State. In the collar's version, sensors are read and posture control and localization algorithms are executed, which include the triggering of a stimulus when needed.

Chapter 5

Experimental Results and Verifications

The performance of the implemented system was evaluated through a series of experiments that test the key elements of this architecture.

Section 5.1 is dedicated to the measurement of the slot lengths as well as the packet processing time. Some data relaying experiments were carried out as a method of validating the implementation of the B2B packets. These experiments are detailed in Section 5.2. The performance of the synchronization algorithm is evaluated in Section 5.3. In Section 5.4, the experimental result for the packet loss rate is mentioned. Finally, the RSSI values as a function of distance between two nodes can be seen in Section 5.5.

Unless stated otherwise, the results presented in this chapter were obtained with the radio module parameters configured as shown in Table 5.1 and with the timers configured with a resolution and accuracy as mentioned in Table 5.2.

Radio Parameters Configuration

Parameter	Configured value
TX Power (P_{out})	10 (dBm)
# Preamble Bytes / Min detected threshold	4 / 2 (Bytes)
# SYNC Word Bytes / Min detected threshold	4 / 2 (Bytes)
Data Rate	250 (kBaud)
Base frequency	433 (MHz)
Modulation	GFSK
Frequency Deviation (GFSK)	127 (kHz)
RX Filter BW	541.7 (kHz)

Table 5.1 – Configured radio parameters

Timers' accuracy and resolution

Timer	Accuracy	Resolution
T1, T4	40 ppm	0.0788 (ms)
TMAC	40 ppm	0.0006 (ms)
TSLEEP	± 1 %	0.0289 (ms)

Table 5.2 – Timer's resolution and accuracy

5.1 Time measurements

In this section, the TAW's length based on packet processing times was estimated. The slot durations and guarding window size were computed in order to accommodate the schedule based traffic.

5.1.1 TAW's length based on packet processing time

An estimation of the time it takes to process the buffered packets was carried out. A tryout for each packet type was taken with different buffered packet loads. Because the processing time of B2B packets is dependent on the number of collars' data relayed, only its header processing time was measured. The results are shown in Figure 5.1.

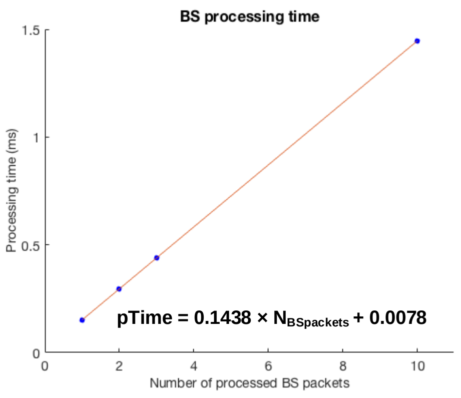
A linearization for the BS, C2B and B2B-header processing times, in ms units, is respectively given by the equations (19), (20) and (21), in which N_{BSpkt}, N_{C2Bpkt} and N_{B2Bpkt} are the number of BS, C2B and B2B packets required to be processed.

$$pTime_{BS}\left(N_{BSpkt}\right) = 0.1438 \times N_{BSpkt} + 0.0078 \ (ms) \tag{19}$$

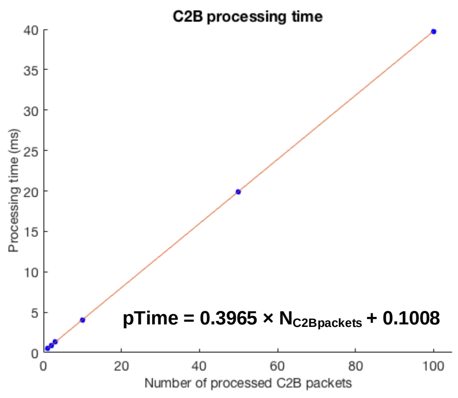
$$pTime_{C2B}\left(N_{C2Bpkt}\right) = 0.3965 \times N_{C2Bpkt} + 0.1008 \ (ms) \tag{20}$$

$$pTime_{B2B}\left(N_{B2Bpkt}\right) = 0.9978 \times N_{B2Bpkt} + 0.2063 \ (ms) \tag{21}$$

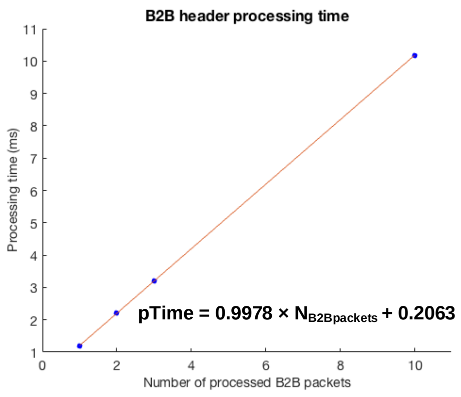
The biggest increase in the B2B packet header processing time when comparing with the BS counterpart, can be explained due to a memory allocation of 254 bytes (the maximum length of a B2B payload) executed in the processing function. The optimization of this function is left as future work.



(a)



(b)



(c)

Figure 5.1 – Processing time of BS (a) and C2B packets (b) and B2B packet headers (c)

To quicken the experimental validation of the system, it was assumed that the impact of the relayed C2B data (Table 3.8, page 39) in the processing time of B2B packets is closely related with the processing time of each C2B packet encapsulated in the relayed collar notifications. Therefore, equation (21) can be rewritten to accommodate the relayed data processing as in (22), in which N_{CN} is the number of relayed collar notifications per packet.

$$pTime_{B2B}\left(N_{B2Bpkt}, N_{CN}\right) = N_{B2Bpkt} \times \left(0.9978 + pTime_{C2B}(N_{CN})\right) + 0.2063 \ \ (ms) \qquad (22)$$

To compute the TAW_{length} for the worst-case scenario, it was considered that a beacon has to process either all BS packets and all C2B packets, or, has to process all BS packets and all B2B packets sent in the previous two windows. A reasonable number of nodes was considered as being 20 beacons and 1000 collars [2]. The number of collar notifications relayed was considered to be 6, which is the maximum that fits in a packet with 254 bytes of payload in which 36 bytes are reserved for an header (B2B header for MAX_BEACONS equal to 20). The length of the TAW only due to this packet load processing is:

$$TAW_{length_{20\,Beacons,1000\,Collars}} = \max(pTime_{BS}(20) + pTime_{C2B}(1000), pTime_{BS}(20) + pTime_{B2B}(20,6))$$

$$\Leftrightarrow TAW_{length_{20\,Beacons,1000\,Collars}} \approx \max(400, 73) = 400\ (ms)$$

This value is very conservative, even without considering the processing time of the control algorithms, since it is very unlikely to happen that a beacon has to process each and every packet sent during a µC frame.

5.1.2 Time Slot and GW lengths

Time Slot lengths

The time-to-transmit (TT_{TX}) and time-to-receive (TT_{RX}) parameters were measured with a logic analyzer as Figure 5.2 suggests. A digital port was set to '1' while the transmitter node was transmitting a packet. On the receiver side, a digital port was to set to '1' while its transceiver was in RX mode (receiving or being able to receive a packet) and set to '0' while decoding and buffering a packet. The TT_{TX} parameter is the time it took the transmitter to send a packet since its timer ISR was triggered. The TT_{RX} is the time since the instant the packet is fully transmitted until the instant the receiver is again ready to receive another packet. It encompasses both the packet decoding and buffering, due to SheepIT's protocol implementation, and a delay caused by the hardware (transceiver's and DMA delay to acknowledge a packet reception) and by the propagation time of the radio signal.

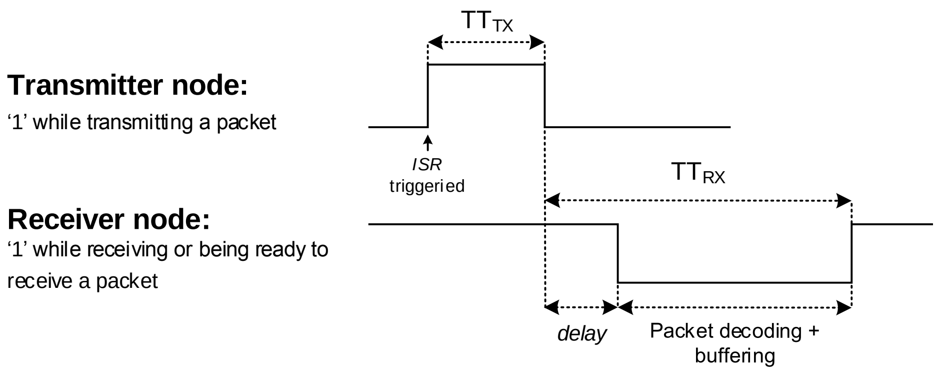

TTTX
Transmitter node:
'1' while transmitting a packet
ISR triggeried
TTRX
Receiver node:
'1' while receiving or being ready to receive a packet
delay
Packet decoding + buffering

Figure 5.2 – Slot measurement procedure

The CC1110 SoC datasheet states that the Frequency Synthesizer must be calibrated regularly. This can either be done automatically or manually. The automatic calibration can be programmed to occur every time a packet is received. A manual calibration can be issued, for instance, at the end of the TAW (prior to the next state transition) through a command strobe. The time it takes to perform a FS calibration is especially important when it is done automatically, since it must be considered as part of the time slot duration. The beacon's and collar's firmware was written in such a way that FS calibration can be easily switched between auto and manual mode, so the impact of both calibration modes on the RSSI values may be later analyzed.

Hence, the TT_{RX} and TT_{TX} parameters were measured for both manual and auto calibration. A total of 10 tryouts were performed for every packet type and every FS calibration mode, with the transceivers used in the experience swapping between transmitter and receiver actions after 5 tryouts. For B2B packets, a maximum payload was used (255 Bytes). The measured results are shown in Table 5.3 and Table 5.4.

No Frequency Synthesizer calibration

TT_{RX} (ms)

Packet	**TT_{RX}**	**Δ**
BS	1.34	0.03
C2B	1.09	0.07
B2B [255 Bytes]	2.02	0.16

(a)

TT_{TX} (ms)

Packet	**TT_{TX}**	**Δ**
BS	0.67	0.01
C2B	1.09	0.02
B2B [255 Bytes]	8.61	0.04

(b)

Table 5.3 – Measured TT_{RX} (a) and TT_{TX} parameters without FS calibration

With Frequency Synthesizer calibration

TT_{RX} (ms)

Packet	**TT_{RX}**	**Δ**
BS	2.19	0.01
C2B	1.89	0.09
B2B [255 Bytes]	2.79	0.17

(a)

TT_{TX} (ms)

Packet	**TT_{TX}**	**Δ**
BS	0.67	0.03
C2B	1.10	0.05
B2B [255 Bytes]	8.61	0.04

(b)

Table 5.4 – Measured TT_{RX} (a) and TT_{TX} parameters with FS calibration

Guarding Window length

With an oscillator accuracy of 40 ppm, the maximum time drift between two nodes as a function of the time since the last synchronization (Δt), only due to timer's T1 or TMAC accuracy, can be computed as in (23).

$$Max_{drift}(\Delta t) = 2 \times \left(\Delta t \times \frac{40}{1 \times 10^6}\right) \tag{23}$$

The GW length was computed in order to accommodate the drift due to the timers' accuracy for the maximum time between a synchronization instant (first SW slot) and a transmission instant (last VTW slot). For that purpose, a reasonable value for the µC length was firstly computed.

- **µC length estimation**

An estimation of the µC length was carried out for the same number of nodes used in TAW's length estimation. In equations (1), (2) and (4), the variables BS_{TTRX}, BS_{TTTX}, $C2B_{TTRX}$ and $C2B_{TTTX}$ were substituted by the respective values from Table 5.4 (worst-case). With a TAW_{length} of 400 ms, N_{Beac} equal to 20, N_{Col} equal to 1000 and ignoring the guarding window size for now, the µC length is estimated to be:

$$\mu C_{length} \approx 3571\ (ms)$$

Returning to the GW length estimation, the maximum drift for the previously computed µC length is

$$Max_{drift}(3571) = 0.2857 \approx 0.3 (ms)$$

Other factors may also introduce jitter, such as different CPU activity loads and timers' resolution. To also address these issues, a conservative value for the guarding window was chosen, being that 1 ms.

$$GW_{length} = 1\ ms$$

5.2 Data relaying

To test the ability of a beacon to collect data from a collar outside of its covered area, an experiment was carried out to emulate a situation like the one depicted in Figure 5.3. A macro-cycle of alternating µC types 2 and 3 was implemented.

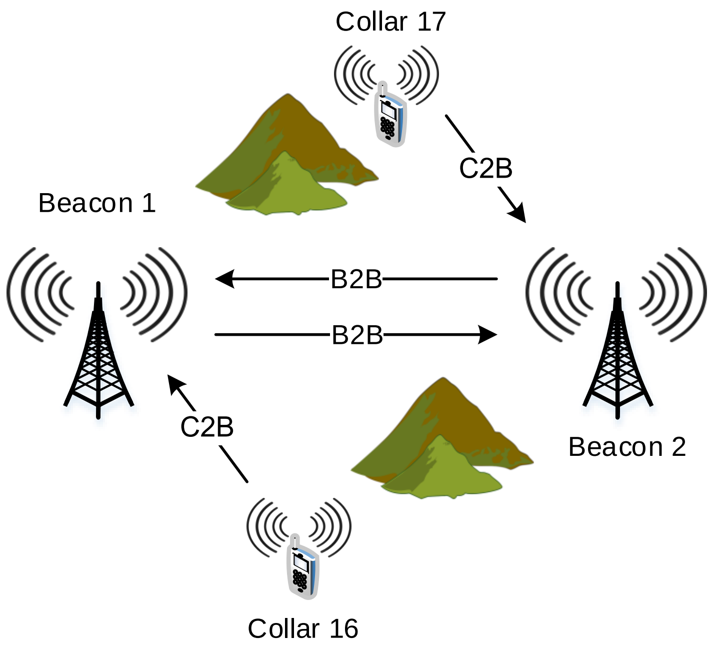
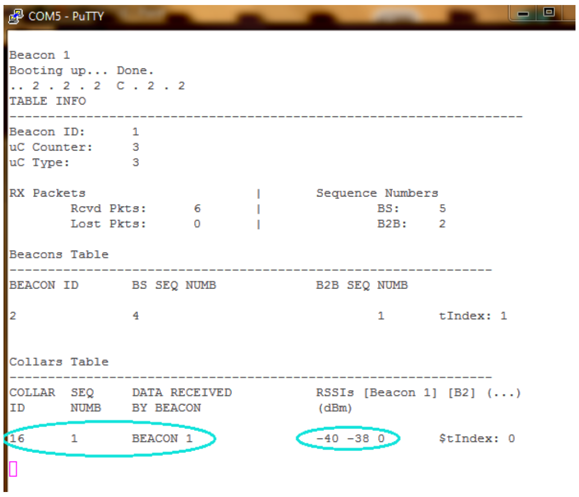


Figure 5.3 – Data relaying experiment

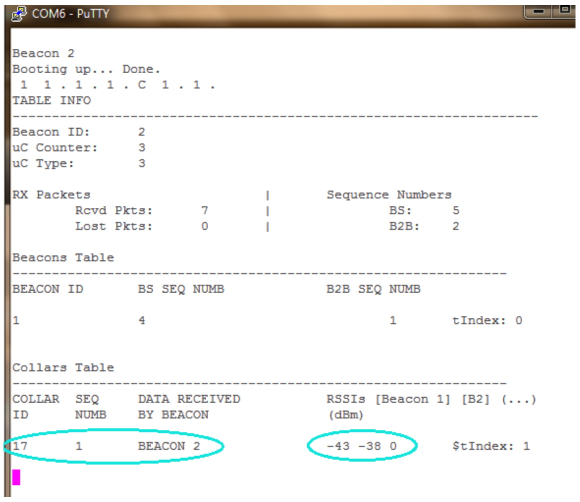
To emulate the absence of a direct link between a given collar and a beacon, beacon 1 was forced to ignore collar 17 C2B packets and beacon 2 was forced to ignore collar 16 C2B packets.

Figure 5.4 shows the table status of beacons 1 and 2 prior to the update with the relayed data. For demonstration purposes, only the collar sequence number and its last RSSI values are shown as part of the received C2B packet information.

Figure 5.5 displays the table status of beacons 1 and 2 after the update with the first data received through relay. At this time, the two collars are known to every beacon as expected.



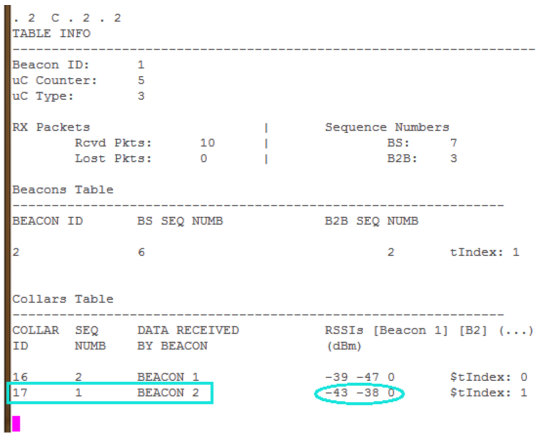
```
COM5 - PuTTY

Beacon 1
Booting up... Done.
.. 2 . 2 . 2  C . 2 . 2
TABLE INFO
-------------------------------------------------------------------
Beacon ID:      1
uC Counter:     3
uC Type:        3

RX Packets                      |       Sequence Numbers
        Rcvd Pkts:      6       |               BS:     5
        Lost Pkts:      0       |               B2B:    2

Beacons Table
---------------------------------------------------------------
BEACON ID       BS SEQ NUMB             B2B SEQ NUMB

2               4                               1       tIndex: 1


Collars Table
---------------------------------------------------------------
COLLAR  SEQ     DATA RECEIVED           RSSIs [Beacon 1] [B2] (...)
ID      NUMB    BY BEACON               (dBm)

16      1       BEACON 1                -40 -38 0       $tIndex: 0
```



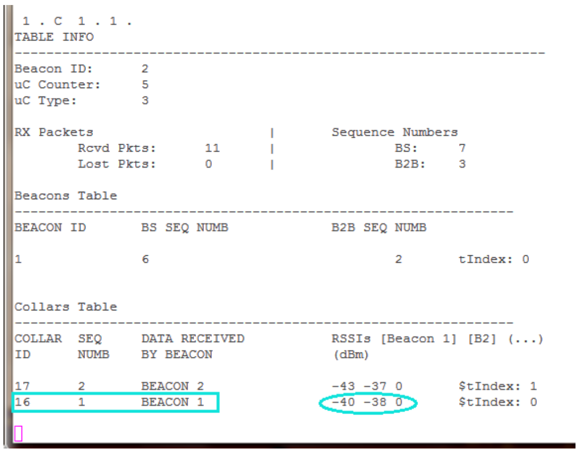
```
COM6 - PuTTY

Beacon 2
Booting up... Done.
 1  1 . 1 . 1 . C  1 . 1 .
TABLE INFO
-------------------------------------------------------------------
Beacon ID:      2
uC Counter:     3
uC Type:        3

RX Packets                      |       Sequence Numbers
        Rcvd Pkts:      7       |               BS:     5
        Lost Pkts:      0       |               B2B:    2

Beacons Table
---------------------------------------------------------------
BEACON ID       BS SEQ NUMB             B2B SEQ NUMB

1               4                               1       tIndex: 0


Collars Table
---------------------------------------------------------------
COLLAR  SEQ     DATA RECEIVED           RSSIs [Beacon 1] [B2] (...)
ID      NUMB    BY BEACON               (dBm)

17      1       BEACON 2                -43 -38 0       $tIndex: 1
```

(a) (b)

Figure 5.4 – Beacons' 1 (a) and 2 (b) table status prior to the update with the received B2B data



```
. 2  C . 2 . 2
TABLE INFO
-------------------------------------------------------------------
Beacon ID:      1
uC Counter:     5
uC Type:        3

RX Packets                      |       Sequence Numbers
        Rcvd Pkts:      10      |               BS:     7
        Lost Pkts:      0       |               B2B:    3

Beacons Table
---------------------------------------------------------------
BEACON ID       BS SEQ NUMB             B2B SEQ NUMB

2               6                               2       tIndex: 1


Collars Table
---------------------------------------------------------------
COLLAR  SEQ     DATA RECEIVED           RSSIs [Beacon 1] [B2] (...)
ID      NUMB    BY BEACON               (dBm)

16      2       BEACON 1                -39 -47 0       $tIndex: 0
17      1       BEACON 2                -43 -38 0       $tIndex: 1
```



```
 1 . C  1 . 1 .
TABLE INFO
-------------------------------------------------------------------
Beacon ID:      2
uC Counter:     5
uC Type:        3

RX Packets                      |       Sequence Numbers
        Rcvd Pkts:      11      |               BS:     7
        Lost Pkts:      0       |               B2B:    3

Beacons Table
---------------------------------------------------------------
BEACON ID       BS SEQ NUMB             B2B SEQ NUMB

1               6                               2       tIndex: 0


Collars Table
---------------------------------------------------------------
COLLAR  SEQ     DATA RECEIVED           RSSIs [Beacon 1] [B2] (...)
ID      NUMB    BY BEACON               (dBm)

17      2       BEACON 2                -43 -37 0       $tIndex: 1
16      1       BEACON 1                -40 -38 0       $tIndex: 0
```

(a) (b)

Figure 5.5 – Beacons' 1 (a) and 2 (b) table status after being updated with the previously received B2B relayed data

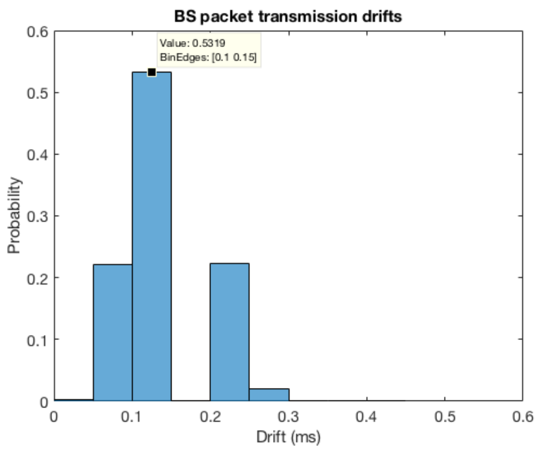
5.3 Synchronization

In this section, the synchronization algorithm is evaluated in terms of clock drifts when it comes to packet transmission instants.

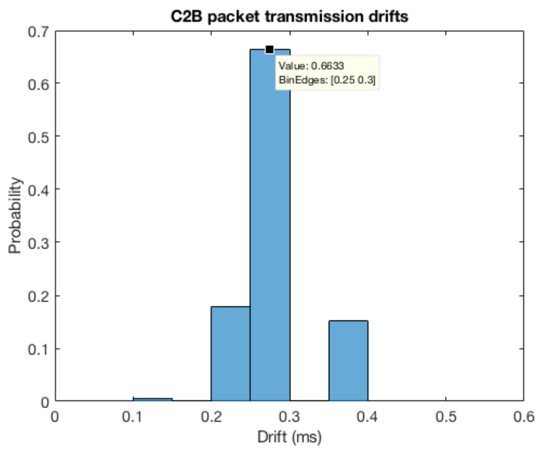
5.3.1 Packet transmission time drifts

To measure the beacons' transmission time drifts, a 3-node network was deployed with 3 beacons exchanging data over type 3 µC's. A forth CC1110 device was programmed as a *packet sniffer*, time-stamping the instants at which it received each packet. The 3 beacons were all running the same synchronization algorithm with the clock drift compensation. The *packet sniffer* measured the elapsed times between packet transmissions and computed their drifts. The VTW's traffic drift was computed against the nominal transmission time values, which in turn are relative to the beginning of the TAW (the last synchronized event).

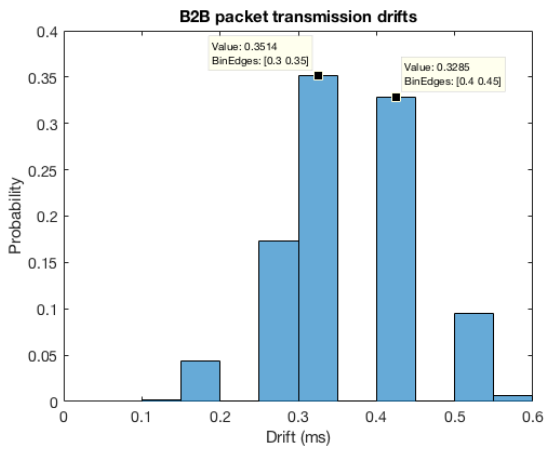
The experiment took about 15 minutes per trial. Four trials were executed and the beacons re-programmed with different ID's in order to toggle their time slots. The final results for the measured BS and B2B packet transmission drifts are plotted in a histogram in Figure 5.6 (a) and (c), respectively.



(a)



(b)

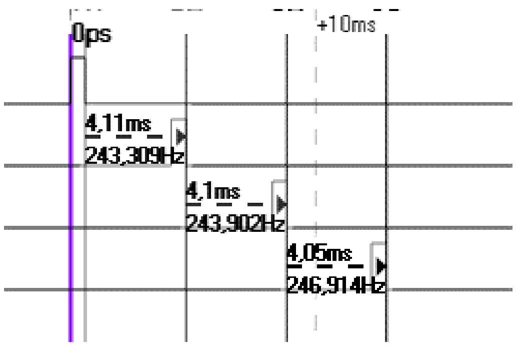


(c)

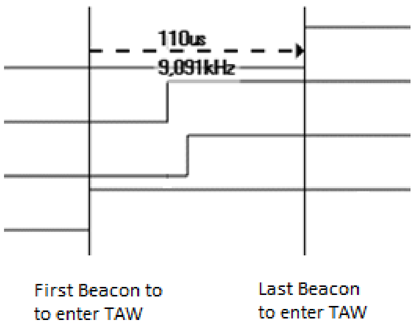
Figure 5.6 – Packet transmission drifts: BS (a), C2B (b) and B2B (c)

The collars' transmission time drifts were measured in a similar fashion using the same *packet sniffer*. This time, a network with a single beacon and two collars was deployed. As in the previous experiment, four trials of 15 minutes each were taken and the nodes swapped time slots between the trials. A histogram with the final results is plotted in Figure 5.6 (b).

The obtained results serve only as an estimation of the drifts. The *packet sniffer* also suffers from clock drift issues and so these results are also affected by its error. As a matter of validating the synchronization with a more accurate method, the packet transmission times were measured using a logic analyzer. Figure 5.7 shows the elapsed time between 4 BS packet transmissions and the maximum difference between the TAW's start of execution.



(a)



(b)

Figure 5.7 – Elapsed time between 4 BS packet transmissions (a) and maximum time difference in TAW's start of execution instant between 4 beacons (b). FS autocalibration was used in (a).

BS slot length using FS autocalibration is 2.9 ms long. Adding a guarding window of 1 ms, consecutive packet transmissions should be separated by 3.9 ms when perfectly synchronized with no drift. In the last figure, one can see that the separation is much larger (from 4.05 to 4.11 ms). This happened because the delay between the packet reception instant and the moment the synchronization algorithm is executed is being neglected. So, when a single TT_{RX} plus 1 GW should be elapsed (separating the end of one packet transmission from the beginning of the next), a TT_{RX} plus 1 GW plus the aforementioned delay is the resulting separation. This also resulted in larger values for the drift measurements in the last histograms. However, it is clear that because all devices suffer from the same issue, the synchronization performance is not affected as the TAW's start of execution event is synchronized with a difference of 110 µs, as seen in Figure 5.7 (b). The optimization of the clock drift compensation function by accounting this delay was left as future work.

5.3.2 Clock drift through time after losing synchronization

In a real scenario, one beacon may temporarily lose connectivity with all its neighbors. The absence of synchronization during this time can lead to packet collisions if the transmission instants drift too much.

The following experiment was conducted to evaluate how two packet transmissions drift from each other when the synchronization is suspended. Two beacons start to exchange BS packets (synchronously) when the second beacon stops using the synchronization algorithm (refuses to accept the other beacon's packets). A *packet sniffer* computed the drift between the two beacons. The time drift between the two nodes is plotted in Figure 5.8.

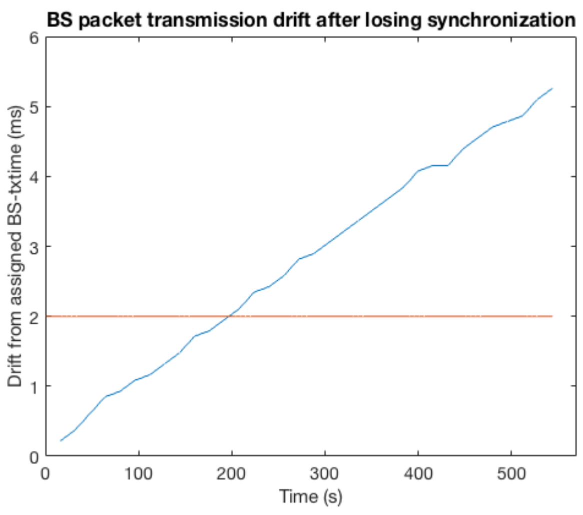


Figure 5.8 – Packet transmission drift through time after losing synchronization

Slots should be separated by 1 GW, when normally synchronized. When the drift exceeds 2 × GW, the time slots begin to overlap. To avoid packet collisions, a timeout should be triggered in the beacon that has lost the synchronization with its neighbors after the amount of time at which its transmission instant is expected to be drifted in (2 × GW) units of time from its assigned $BStx_{time}$ instant. For a guarding window of 1 ms, the results displayed in Figure 5.8 show that this timeout should be triggered at about 200 seconds after losing synchronization, preventing the desynchronized beacon from sending more packets until it is able to re-synchronize with a neighbor.

5.4 Packet loss rate

An experiment was carried out with one beacon and one collar exchanging packets over type 2 µC's. The slots were configured with the auto FS calibration TT_{TX} and TT_{RX} parameters from Table 5.4, page 77.

A total of 3085 packets were exchanged. The packet loss during this lab experiment was 9.5 %.

5.5 RSSI measurements

During this dissertation it was not possible to address a proper virtual fence algorithm. The oversimplified version (pseudocode in Figure 4.24, page 70) is known to be insufficient. Nevertheless, the relationship between the RSSI and the distance between a collar and a beacon was experimentally carried out as a reference for future work. Two pairs of radios with equal parameters were tested during this experiment.

Figure 5.9 shows how the experiment was conducted. Two beacons placed at 55 meters from each other send periodic BS messages to a collar that moves between them. The collar computes the RSSI of the two received packets and sends this information back to the beacons in a C2B message. One of the beacons is connected to a computer through a serial port to where the collar data is dumped. The collar moves in a straight line between the two beacons in steps of 5 meters. A total of 10 RSSI samples per beacon-collar pair were collected for each distance in every trial.

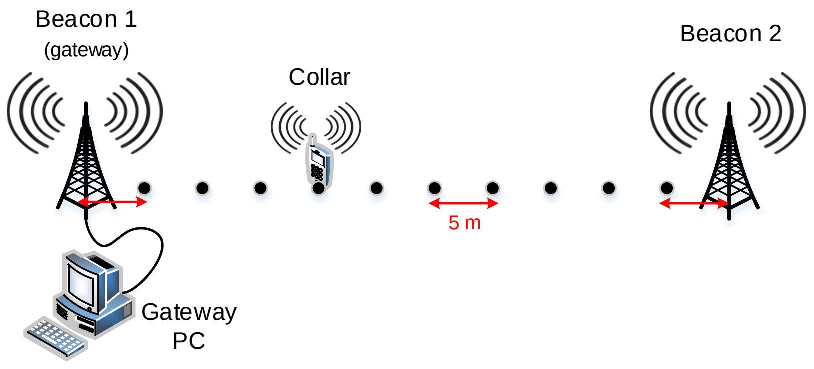
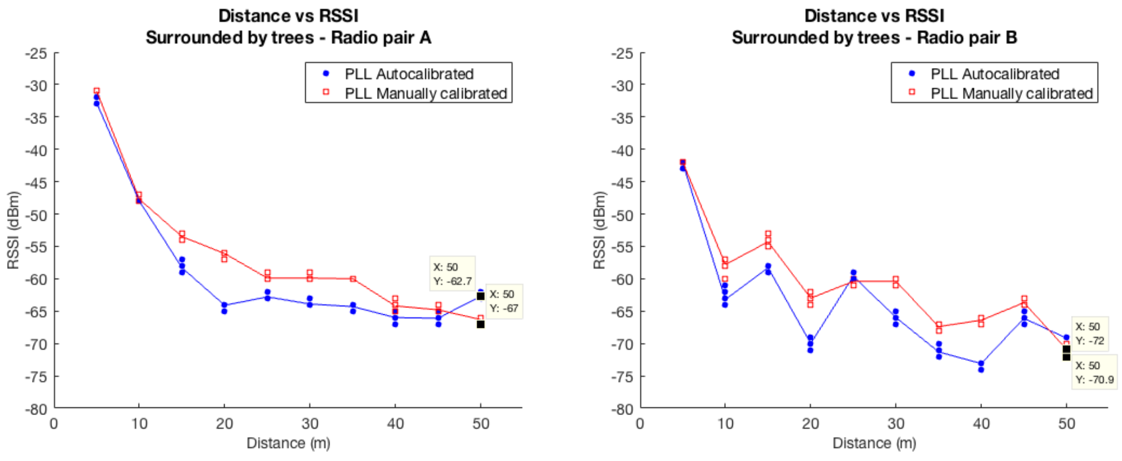


Figure 5.9 – RSSI measurement experiment

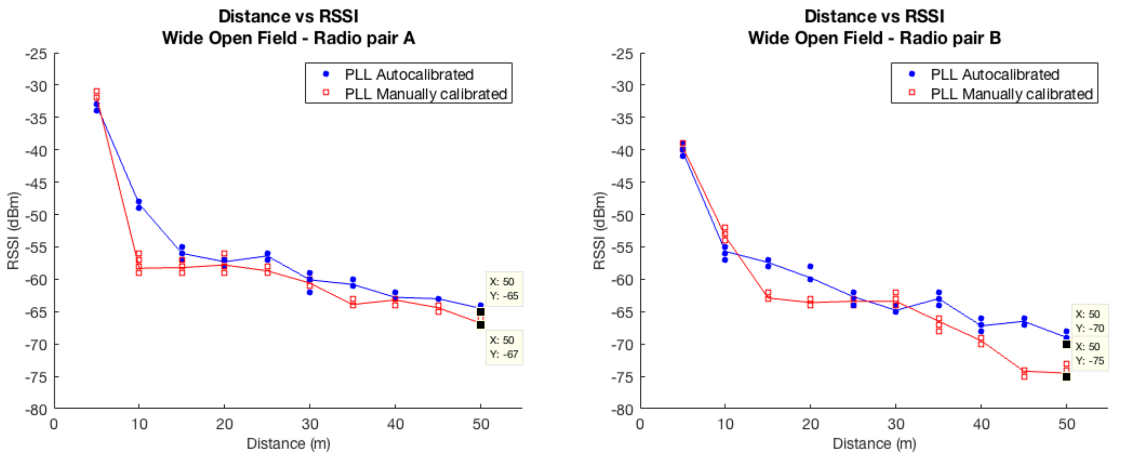
To observe if the FS calibration method impacts on the RSSI, one trial was performed with the FS auto calibration enabled and another trial with manual calibration issued at the end of the TAW.

This experiment was conducted during the same day in two different places. It was firstly tested in a smaller field with some trees and houses surrounding the area. The second test was done in a wider field with no nearby obstacles. The test results are shown in Figure 5.10 (smaller field) and Figure 5.11 (wider field). The RSSI average values are connected by the blue (FS autocalibration) and red (manual calibration) lines.



(a) (b)

Figure 5.10 – RSSI vs distance test results in a field surrounded by trees using two pairs of radios



(a) (b)

Figure 5.11 – RSSI vs distance test results in a wide-open field using two pairs of radios

From this experiment, it was not clear how the FS calibration method impacts the received signal strength on the collar. However, it shows the difficulty of predicting the distance from one collar to a beacon based only on the received strength. Therefore, a refinement of localization based on RSSI needs to be addressed in future work prior to simply rely on a single RSSI threshold.

The plots show that supposedly equal radios programmed with the same radio configuration parameters can produce different RSSI values for the same distance.

The experiment was performed in two different places as a form of validating the results, in particular the RSSI threshold for 50 m (considered a reasonable radius for a beacon covered area [2]). It is verified that the differences between the two pairs of radio were manifested in the two scenarios, being the RSSI values for the pair named "B" consistently lower for 50 m. This enforces that in future work more experiments should be performed with different pairs of radios to estimate an error for the RSSI due to differences in radio modules.

As a reference for future experiments, the antenna posts heights used in the previous experiments are laid in Table 5.5.

Antenna posts heights

Node	**h (m)**
Beacon 1, 2	1.8
Collar	0.5

Table 5.5 – *Antenna posts heights*

5.6 Summary

A very conservative value for the TAW's length was estimated solely based on packet processing times. It was assumed that during one TAW a total of 20 BS, 1000 C2B and 20 B2B packets relaying 6 CN each had to be processed, which required 470 ms. Time slots lengths were defined by experimentally measuring the TT_{TX} and TT_{RX} of each packet. The GW length was also conservatively estimated for the maximum theoretical drift in transmission instants due to oscillator's accuracy.

The synchronization algorithm was evaluated by time stamping the instants of packet transmissions. The results show a drift higher than it was initially expected. However, this has to do with an overhead that was wrongly neglected. When synchronization takes place, the node is not considering the delay between the packet reception and the start of execution of this algorithm. Because the TT_{RX} lengths were measured from the packet reception instant and not from the beginning of execution of the synchronization algorithm, nodes will always wait a bit more than they are actually required to do. This resulted in a wrongly measured drift, because this delay was not taken into account. The optimization of this algorithm implementation is left as future work.

A 9.5 % of packet loss was obtained during a lab experimentation. Results under real conditions (vineyards) are still required to be performed. Further improvements in RSSI values filtering need to be made to address a proper localization system for SheepIT.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this dissertation, an alternative solution for the weeding process in vineyards was presented. It's based on an eco-friendly solution in which sheep take the job of grazing the properties. To monitor their behavior and localization, a Wireless Sensor Network (WSN) was designed and a prototype deployed. It is composed of collars carried by sheep that monitor their activity and location. This data is transmitted to beacons, strategically distributed throughout the vineyard. The collected data is relayed by neighbor beacons until it reaches a gateway, in which all collar data should be dumped. All communications in the WSN use a common radio link, which imposed the design of a MAC layer in order to efficiently accommodate the traffic exchange.

In the State of the Art chapter, we've managed to classify the MAC layer policies as contention-based and scheduled-based, according to the literature. Some protocols targeted to WSN applications were surveyed as their advantages and drawbacks were also disclosed. None of the protocols found in literature is fully adapted to SheepIT's project constraints. Animal monitoring systems were also investigated in this chapter. Satellite based systems, particularly the GPS, are commonly used for this purpose, however, due to the high power consumption that these technologies demand, alternative solutions are being considered. Localization based on radio signals stands out due to its inexpensive implementation in WSN's that already incorporate a radio link for data exchange.

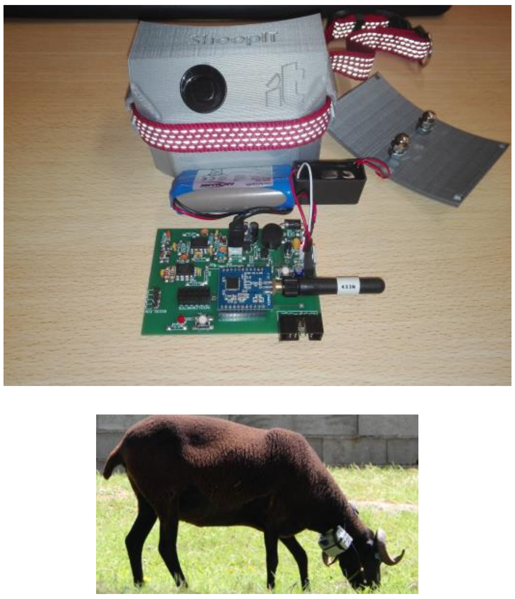
SheepIT's WSN was designed to be energy efficient. We took the duty-cycle approach to minimize the energy consumption of sheep-borne collars by letting them periodically enter low power states. The MAC layer can accommodate both schedule based and contention based traffic, but the last one is reserved for dynamic node register and pairing. The traffic is exchanged in a periodic pattern. For each type of traffic, a micro-cycle is devoted (µC) and the periodic sequence of µC's form the macro-cycle structure (MC). Based on the traffic exchanged during a µC, they were classified in three types: type 1 (node pairing using CSMA), type 2 (collar traffic using TDMA) and type 3 (inter-beacon relay using TDMA).

Two state-machines were implemented – a collar and a beacon. These state-machines obey the protocol presented in the System Architecture chapter with the exception that node register and pairing are not dynamically performed. That being said, nodes are statically assigned an ID, so µC's of type 1 are not addressed in this prototype. Moreover, despite the reservation of packet fields for that purpose, no routing schemes were implemented, so every message is being broadcasted.

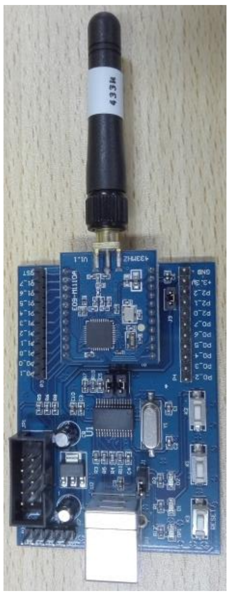
The schedule based traffic enforced the introduction of a synchronization algorithm with well-known packet transmission and reception times. Regarding this topic, a difference of 110 µs was observed between the first and the last node to acknowledge a given time instant that had to be synchronized between all nodes. A testbed consisted of 4 nodes was used for that matter.

In a laboratory experiment, a packet loss rate of 9.5 % was observed with a total of 3085 packets exchanged between two nodes. However, several parameters still need to be evaluated in a real environment, such as the influence of baud rate in packet loss as well as how the radio signal is degraded when obstacles are in between a transmitter and receiver node. Tests were performed outside of the laboratory to observe the variation of RSSI values as a function of distance between a transmitter and a receiver. The results may be used as future reference when evaluating different radio parameters and how they affect the covered area radius.

This WSN prototype was integrated with the hardware (Figure 6.1) and posture control algorithm developed under the SheepIT project. In the current development status, beacons are able to collect real data from collars and relay it between their neighbors. All communications between collars and beacons and beacons with their peers are synchronous and operate in a single channel of the 433 MHz ISM band.



(a)



(b)

Figure 6.1 – A collar device (a) and a beacon (b) of SheepIT's project

6.2 Future Work

The work presented in this dissertation is the first prototype of a WSN to be integrated in SheepIT project. Thence, several points of this work can still be improved in future developments.

The following topics summarize what this dissertation has left as future work.

- **Dynamic node register and pairing:** the existence of type 1 µC's for dynamic node register and pairing is planned in the system's architecture. Its implementation was not addressed in this dissertation.

- **Routing:** all messages are currently being broadcast. Routing mechanisms will help to minimize memory usage in beacons, since not every received inter-beacon relay packet would have to be buffered.

- **Packet segmentation:** the current version of this protocol takes no provisions against messages that exceed the maximum amount of data that hardware imposes. Packet segmentation should be addressed in future since B2B packets will surely exceed the maximum packet size when more data starts to be relayed besides the Collar Notifications.

- **Field experiments with different radio parameters:** the impact of a different configuration of the radio module was not tested. Field experiments with different baud rates or different RF equipment should be conducted to determine the right configuration for the project scenario requirements.

- **Autonomy test:** the system architecture was designed to be energy efficient. However, no experiments were conducted to evaluate the system regarding this issue.

- **Localization algorithm and virtual fence:** the hardware used in this prototype provides RSSI support. The development of a more sophisticated algorithm besides a single threshold value is still planned as future work.

- **Code optimization:** several parts of the code can be optimized. Memory allocation performed in the TAW is one example.

References

[1] "SheepIT Project." [Online]. Available: http://www.av.it.pt/sheepit/. [Accessed: 25-Sep-2017].

[2] L. Nóbrega, P. Gonçalves, P. Pedreiras, and S. Silva, "Energy efficient design of a pasture sensor network," *IEEE 5th Int. Conf. Futur. Internet Things Cloud*, 2017.

[3] A. Monteiro and I. Moreira, "Reduced rates of residual and post-emergence herbicides for weed control in vineyards," *Weed Res.*, vol. 44, no. 2, pp. 117–128, 2004.

[4] C. Carlos, "Spraying challenges in the Douro Wine Region of Portugal," 2014.

[5] Q. V. de Fornos, "Sheep grazing in a portuguese vineyard (Tagus)." [Online]. Available: http://www.quintavalefornos.com/sustainability.

[6] F. Dastgheib and C. Frampton, *Weed management practices in apple orchards and vineyards in the South Island of New Zealand*, vol. 28. 2000.

[7] R. F. Morais, "Controlo de Postura Animal," Universidade de Aveiro, 2017.

[8] H. Karl and A. Wiling, *Protocols and Architectures for Wireless Sensor Networks.* Chichester, UK: Jonh Wiley & Sons, Ltd, 2007.

[9] P. Huang, L. Xiao, S. Soltani, M. W. Mutka, and N. Xi, *The Evaluation of MAC Protocols in Wireless Sensor Networks: A Survey*, vol. 15. 2013.

[10] G. Fang and E. Dutkiewicz, "BodyMAC: Energy efficient TDMA-based MAC protocol for Wireless Body Area Networks," *2009 9th International Symposium on Communications and Information Technology.* pp. 1455–1459, 2009.

[11] D. Dzung, S.- Vasteras, C. Apneseth, N.- Billingstad, J. Endresen, and N.- Billingstad, "Design and Implementation of a Real-Time Wireless Sensor/Actuator Communication System," vol. 2, pp. 433–442, 2005.

[12] L. Kleinrock and F. Tobagi, "Packet Switching in Radio Channels: Part I - Carrier Sense Multiple-Access Modes and Their Throughput-Delay Characteristics," *IEEE Transactions on Communications*, vol. 23, no. 12. pp. 1400–1416, 1975.

[13] I. Rubin, "Access-control disciplines for multi-access communication channels: Reservation and TDMA schemes," *IEEE Transactions on Information Theory*, vol. 25, no. 5. pp. 516–536, 1979.

[14] S. Zhuo, Y. Q. Song, Z. Wang, and Z. Wang, "Queue-MAC: A queue-length aware hybrid

CSMA/TDMA MAC protocol for providing dynamic adaptation to traffic and duty-cycle variation in wireless sensor networks," *2012 9th IEEE International Workshop on Factory Communication Systems*. pp. 105–114, 2012.

[15] B. Zhao and H. Yang, "Design of Radio-Frequency Transceivers for Wireless Sensor Networks," in *Wireless Sensor Networks: Application - Centric Design*, InTech, 2010.

[16] F. Zahra Djiroun and D. Djenouri, *MAC Protocols with Wake-up Radio for Wireless Sensor Networks: A Review*, vol. PP. 2016.

[17] F. Tobagi and L. Kleinrock, "Packet Switching in Radio Channels: Part II - The Hidden Terminal Problem in Carrier Sense Multiple-Access and the Busy-Tone Solution," *IEEE Transactions on Communications*, vol. 23, no. 12. pp. 1417–1433, 1975.

[18] "IEEE Standard for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications," *IEEE Std 802.11-1997*. pp. 1–445, 1997.

[19] N. Abramson, "Development of the ALOHANET," *IEEE Transactions on Information Theory*, vol. 31, no. 2. pp. 119–123, 1985.

[20] A. El-Hoiydi, *Aloha with preamble sampling for sporadic traffic in Ad Hoc wireless sensor networks*, vol. 5. 2002.

[21] A. Klein, "Preamble-Based Medium Access in Wireless Sensor Networks," M. A. B. T.-W. S. N.-T. and P. Matin, Ed. Rijeka: InTech, 2012, p. Ch. 07.

[22] M. Buettner, G. Yee, E. Anderson, and R. Han, *X-MAC: A short preamble MAC protocol for duty-cycled wireless sensor networks*, vol. 4. 2006.

[23] A. Von Bodisco, J. Klaue, and J. Schalk, *BP-MAC: A high reliable backoff preamble MAC protocol for wireless sensor networks*, vol. 2009.

[24] W. Ye, J. Heidemann, and D. Estrin, "An energy-efficient MAC protocol for wireless sensor networks," *Proceedings.Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 3. pp. 1567–1576 vol.3, 2002.

[25] W. Ye, J. Heidemann, and D. Estrin, "Medium access control with coordinated adaptive sleeping for wireless sensor networks," *IEEE/ACM Transactions on Networking*, vol. 12, no. 3. pp. 493–506, 2004.

[26] I. Rhee, A. Warrier, M. Aia, J. Min, and M. L. Sichitiu, "Z-MAC : A Hybrid MAC for Wireless Sensor Networks," vol. 16, no. 3, pp. 511–524, 2008.

[27] I. Rhee, A. Warrier, J. Min, and L. Xu, *DRAND: Distributed randomized TDMA scheduling*

for wireless ad hoc networks, vol. 8. 2009.

[28] "Institute of Electrical and Electronics Engineers (IEEE)." [Online]. Available: https://www.ieee.org. [Accessed: 25-Sep-2017].

[29] "IEEE Standard for Information Technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks Specific Requirements Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for," *IEEE Std 802.15.4-2003*, pp. 1–670, 2003.

[30] "IEEE Standard for Low-Rate Wireless Networks - Amendment 5: Enabling/Updating the Use of Regional Sub-GHz Bands," *IEEE Std 802.15.4v-2017 (Amendment to IEEE Std 802.15.4-2015, as Amend. by IEEE Std 802.15.4n-2016, IEEE Std 802.15.4q-2016, IEEE Std 802.15.4u-2016, IEEE Std 802.15.4t-2017)*, pp. 1–35, 2017.

[31] P. Bartolomeu, M. Alam, J. Ferreira, and J. Fonseca, "Survey on low power real-time wireless MAC protocols," *J. Netw. Comput. Appl.*, vol. 75, no. Supplement C, pp. 293–316, 2016.

[32] T. Semprebom, C. Montez, and F. Vasques, "(m,k)-firm pattern spinning to improve the GTS allocation of periodic messages in IEEE 802.15.4 networks," *EURASIP J. Wirel. Commun. Netw.*, vol. 2013, no. 1, p. 222, 2013.

[33] F. Wang, D. Li, and Y. Zhao, *Analysis and Compare of Slotted and Unslotted CSMA in IEEE 802.15.4*. 2009.

[34] L.-J. Hwang, S.-T. Sheu, Y.-Y. Shih, and Y.-C. Cheng, *Grouping strategy for solving hidden node problem in IEEE 802.15.4 LR-WPAN*. 2005.

[35] T. Elshabrawy, E. Mamdouh, M. Ashour, and J. Robert, *Report Success Probability Analysis of Dense IEEE 802.15.4-Based Metering Networks with Hidden Nodes*, vol. PP. 2017.

[36] "The Zigbee Alliance." [Online]. Available: http://www.zigbee.org/. [Accessed: 04-Oct-2017].

[37] J. Song, S. Han, A. Mok, D. Chen, M. Lucas, M. Nixon, and W. Pratt, "WirelessHART: Applying Wireless Technology in Real-Time Industrial Process Control," *2008 IEEE Real-Time and Embedded Technology and Applications Symposium*. pp. 377–386, 2008.

[38] "IPv6 over Low power WPAN." [Online]. Available: https://datatracker.ietf.org/wg/6lowpan/documents/. [Accessed: 04-Oct-2017].

[39] Z. Shelby and C. Bormann, *6LoWPAN: The Wireless Embedded Internet*. 2009.

[40] S. Zhuo, Z. Wang, Y. Q. Song, Z. Wang, and L. Almeida, "A Traffic Adaptive Multi-Channel MAC Protocol with Dynamic Slot Allocation for WSNs," *IEEE Transactions on Mobile*

Computing, vol. 15, no. 7. pp. 1600–1613, 2016.

[41] W. W. Cochran and R. D. Lord, *A Radio-Tracking System for Wild Animals*, vol. 27. 1963.

[42] Telenocis, “VHF Collar.” [Online]. Available: http://www.telonics.com/products/vhfStandard/. [Accessed: 06-Oct-2017].

[43] A. Campus, “VHF tracking receiver.” [Online]. Available: http://www.african-campus.com/product/wildlife-research-internship/. [Accessed: 06-Oct-2017].

[44] S. Madry, “Doppler Satellite Positioning, Telemetry and Data Systems,” in *Global Navigation Satellite Systems and Their Applications*, New York, NY, USA: Springer.

[45] S. Rutter, N. Beresford, and G. Roberts, *Use of GPS to identify the grazing areas of hill sheep*, vol. 17. 1997.

[46] R. Nathan, O. Spiegel, S. Fortmann-Roe, R. Harel, M. Wikelski, and W. Getz, *Using tri-axial acceleration data to identify behavioral modes of free-ranging animals: General concepts and tools illustrated for griffon vultures*, vol. 215. 2012.

[47] A. Llaria, G. Terrasson, H. Arregui, and A. Hacala, “Geolocation and monitoring platform for extensive farming in mountain pastures,” pp. 2420–2425, 2015.

[48] G. Terrasson, A. Llaria, A. Marra, and S. Voaden, *Accelerometer based solution for precision livestock farming: geolocation enhancement and animal activity identification*, vol. 138. 2016.

[49] E. S. Nadimi, H. T. Søgaard, T. Bak, and F. W. Oudshoorn, “ZigBee-based wireless sensor networks for monitoring animal presence and pasture time in a strip of new grass,” *Comput. Electron. Agric.*, vol. 61, no. 2, pp. 79–87, 2008.

[50] R. P and M. Sichitiu, *Angle of Arrival Localization for Wireless Sensor Networks*, vol. 1. 2006.

[51] S. Capkun, M. Hamdi, and J. Hubaux, “GPS-free positioning in mobile Ad-Hoc networks,” vol. 0, no. c, pp. 1–10, 2001.

[52] A. Thottam Parameswaran, M. Iftekhar Husain, and S. Upadhyaya, *Is RSSI a reliable parameter in sensor localization algorithms: an experimental study.* 2009.

[53] K. Heurtefeux and F. Valois, *Is RSSI a Good Choice for Localization in Wireless Sensor Network?* 2012.

[54] K. Hsiang Kwong, T.-T. Wu, H. G. Goh, K. Sasloglou, B. Stephen, I. A. Glover, C. Shen, D. Wencai, C. Michie, and I. Andonovic, *Practical considerations for wireless sensor networks in cattle monitoring applications*, vol. 81. 2012.

[55] J. Huircan, C. Muñoz, H. Young, L. Von Dossow, J. Bustos, G. Vivallo, and M. Toneatti, *ZigBee-based wireless sensor network localization for cattle monitoring in grazing fields*, vol. 74. 2010.

[56] B. Thomas Robertson, J. D Holland, and E. Minot, *Wildlife tracking technology options and cost considerations*, vol. 38. 2012.

[57] B. Thorstensen, T. Syversen, T. Walseth, and T.-A. Bjørnvold, *Electronic Shepherd - a Low-Cost, Low-Bandwidth, Wireless Network System.* 2004.

[58] "SIGFOX." [Online]. Available: https://www.sigfox.com/en. [Accessed: 19-Nov-2017].

[59] H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*, 2nd ed. Springer Publishing Company, Incorporated, 2011.

[60] Texas Instruments, *True System-on-Chip with Low-Power RF Transceiver and 8051 MCU*, Rev. H. .