



POLITECNICO DI BARI

DIPARTIMENTO DI INGEGNERIE ELETTRICA E DELL'INFORMAZIONE

GRADE IN ENGINEERING IN INDUSTRIAL TECHNOLOGIES

END OF DEGREE THESIS
IN
INTERNET OF THINGS

Characterization of TelosB and Zolertia Z1 Running OpenWSN

Docent:
Prof. Ing. Luigi Alfredo GRIECO
Correlator:
Dott. Ing. Pietro BOCCADORO

Student:
Asier JORDANA URRIZA

Academic year 2015-2016

Index

1. Introduction	4
1.1 Problem.....	4
1.2 Tools.....	5
1.3 Reach of the thesis.....	5
Chapter 2: Communication Protocol.....	6
2.1 IEEE 802.15.4.....	7
2.2 IEEE 802.15.4e.....	12
2.3 RPL.....	15
2.4 UDP	17
Chapter 3: Tools.....	18
3.1 Software: OpenWSN	18
3.1.1 OpenWSN Stack.....	19
3.1.2 Open Visualizer.....	20
3.1.3 UDPlatency	21
3.1.4 Packets	22
3.1.5 Configuration.....	23
3.2 Hardware	24
3.2.1 TelosB	24
3.2.2 Zolertia Z1.....	26
Chapter 4: Tests.....	29
4.1 Aim of the tests.....	29
4.2 Realization of the tests	30
4.3 Test 1.....	31
4.4 Test 2.....	35
4.5 Test 3.....	37
4.6 Test 4.....	39
4.7 Test 5.....	41
4.8 Test 6.....	43
4.9 Test 7.....	45
Chapter 5: Results.....	47
5.1 Desynchronizations.....	47
5.2 PLR.....	49

5.3 Latency	50
6. Conclusion	54
7. Bibliography	I
8. Appendix	III
8.1 Code A1: UDPlatency.c	III
8.2 Code A2: UDPlatency.c	VIII
8.3 Code A3: UDPlatency.c	VIII
8.4 Code A4: UDPlatency.c	VIII
8.5 Code A5: UDPlatency.c	VIII
8.6 Code B1: IEEE 802154E.h	IX
8.7 Code B2: IEEE 802154E.h	IX
8.8 Code C1: schedule.h	IX
8.9 Code C2: schedule.h	IX
8.10 Stack organization diagram of the OpenWSN protocol stack	X

1. Introduction

Prof. Ing. Alfredo Grieco professor in the Politecnico di Bari, has presented an offer of thesis related to the development on **Internet of Things**.

Asier Jordana Urriza, as an engineering student in Universidad Pública de Navarra, staying in Bari, Italy as an ERASMUS student, in the Politecnico di Bari, has taken this project related to this subject.

1.1 Problem

This project is oriented in the development of the Internet of Things. Nowadays, the Internet of Things is a very innovative field with many opportunities. The boom in this subject is thanks to the fast improvement of technology and development of the internet in the last decades that has made possible to connect to the internet almost any device, leading to many applications from smart houses that make our everyday life easier to many industry related applications such as Wireless Sensor Networks.

Due to all the applications it can have, a lot of research communities have joined the development of many tools in this field, such as motes (hardware) and operating systems (software) oriented for the Internet of Things.

Among all the available platforms, in this thesis there are characterized two of these motes running the same OS. The results obtained will be comparable.

The parameters that will be obtained are:

- Packet Lost Rate (PLR): percentual value of the number of packets lost during the transmission;
- latency: the amount of time a packet needs to reach the destination, once it has been sent.

Thanks to the results obtained, it is expected to help the development of these platforms expanding the available information about them. As this field is quite new, the amount of information is minimal and hard to find, and this information may help other researches to continue developing, showing strong and weak points of the studied tools.

1.2 Tools

The hardware that will be tested are the TelosB [13] mote by Memsic and the Z1 [14] by Zolertia. The software they will run for this thesis is OpenWSN.

6 motes of each type will be used, further details are given in chapter 3.

Other required tools:

- laptop: that will collect all the data received by the root mote, using OpenUSB on application layer. The computer runs on Ubuntu 14.04.4 due to the amount of support and tools available in this OS.
- nickel-cadmium AA batteries: to power the motes. These batteries work at 1.2 V and capacity may vary from 600 mAh to 1000 mAh. Each mote needs 2 batteries except the one connected to the computer that will be powered via USB. In total 10 batteries will be used.

1.3 Reach of the thesis

This research field is very studied nowadays by many universities and companies due to its potential and the resources available.

Its applications go from our every-day life (intelligent houses for example) to industrial environments (process monitoring, real-time control, information management, etc.).

These tests, will help to add a little step in this field, giving some performance data about two platforms (TelosB [13] and Zolertia Z1 [14], running OpenWSN [10]) working under some different conditions.

With the data obtained it is expected to see which are the strong points of this scenario and which points need further development and research.

Chapter 2: Communication Protocol

In this chapter it is shown an introduction on the standards used for the Lower layers of the communication protocol (Physical and MAC layers) [1].

To make a **Wireless Sensor Network (WSN)**, it is necessary to define a protocol that adapts to the needs of this network. Nowadays, there is a high number of wireless protocols for many different applications, so it is interesting to take a look to the different offers available.

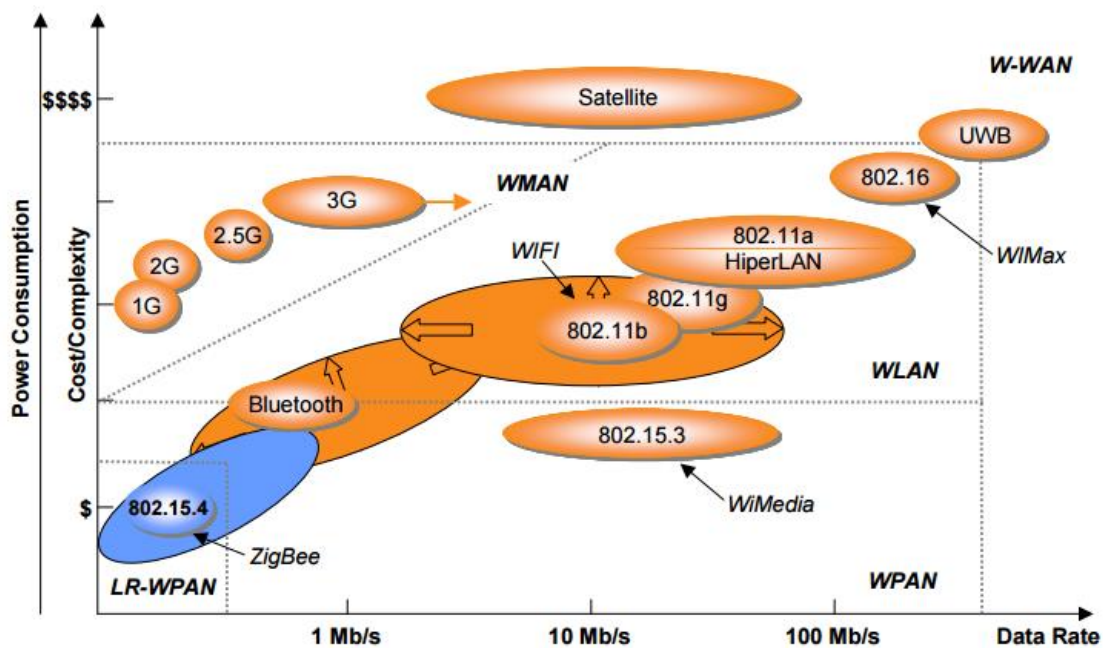


Image 1 Standard communication wireless protocols [2]

The main reasons to set up a wireless network instead of a wired one, are:

- no connectors;
- major mobility;
- ease of installation;
- more flexibility;
- improved share of resources;
- lower cost and infrastructure.

In many cases, the requirements for a WSN are:

- power consumption: low power consumption that will ensure the plant working for long time periods;
- low cost: a company won't pay for features that won't use and will always prioritize a simple-functional solution that lowers the costs of the inversion;
- robustness and stability;
- integration with the production plant.

Among all the wireless communication protocol standards available, each one focuses some applications and optimizes some features over others, for that reason, it is necessary to find the standard that focuses in the previous requirements.

2.1 IEEE 802.15.4

Among all the standards available, the IEEE 802.15.4 standard [1][2][3][4] is very suitable for a WSN. Image 2 compares IEEE 802.15.4 with Bluetooth and Wi-Fi, which are two of the most known standards.

Bluetooth and Wi-Fi are superior regarding to range and data throughput but usually there is no need of a high data transmission in this kind of applications, and the range is covered using several nodes and multihop techniques [3] that allow covering wide areas and lowering power consumption at the same time. The 802.15.4 standard is designed to get a small power consumption that grants long working time periods, a small size that makes possible a complete integration with the plant and low manufacturing costs due to the low characteristics.

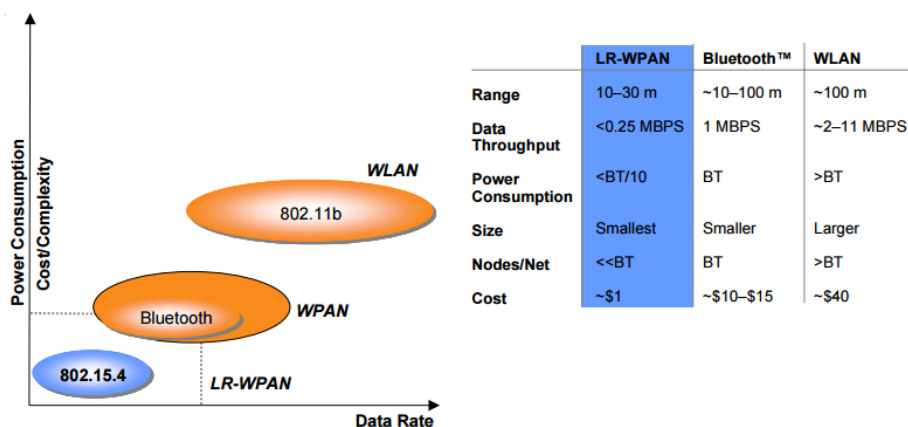


Image 2 IEEE802.15.4, Bluetooth and Wi-Fi comparison

IEEE 802.15.4 is a standard that specifies the physical layer and media access control for low-rate wireless personal area networks (LR-WPAN) [2][3]. Upper layers are not defined in this standard, and will depend on the application it is being implemented.

It implements the fundamental lower network layers, focusing on low-cost, low-speed communications between devices with little infrastructure, intending to minimize the power consumption.

The basic framework conceives:

- range: 10 m;
- transfer rate: Up to 250 kbps;
- real-time suitability;
- medium access: CSMA/CA collision avoidance;
- secure communication support;
- frequency bands: 868/915/2450 MHz.

The interaction among devices is conceived to be over a conceptually simple wireless network.

The network layers are based on the OSI model [1][3] but only the **lower layers** are defined. Interaction with upper layers is intended, accessing the MAC through a convergence sublayer, see Image 3. This implementations may rely on external devices or be embedded self-functioning devices as it is in this case.

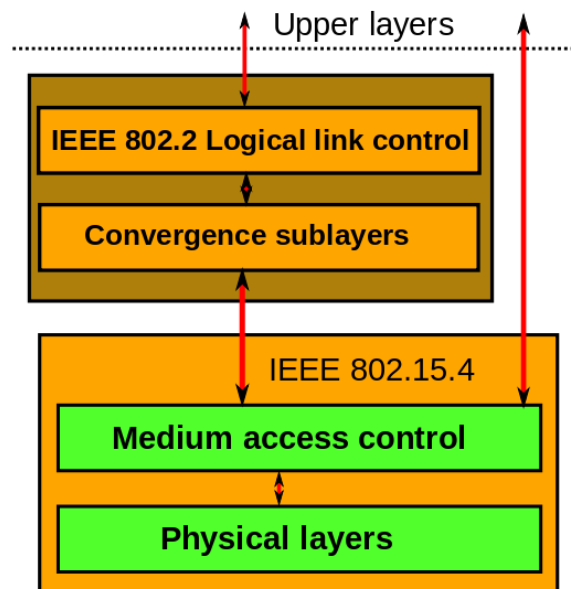


Image 3 IEEE802.15.4 protocol stack [2]

The physical layer manages the RF transceiver, performing channel selection and energy and signal management functions.

It can operate in three different frequency bands: 868.0-868.6 MHz, 902-928 MHz and 2400-2483.5 MHz. The last band is the one used in this thesis because it is the band that the radio chips in these motes use. See Image 4.




	Channel	Center Frequency (MHz)	Availability
868 MHz Band	0	868.3	 <i>Europe</i>
915 MHz Band	1	906	 <i>Americas</i>
	2	908	
	3	910	
	4	912	
	5	914	
	6	916	
	7	918	
	8	920	
	9	922	
	10	924	
2.4 GHz Band	11	2405	 <i>World Wide</i>
	12	2410	
	13	2415	
	14	2420	
	15	2425	
	16	2430	
	17	2435	
	18	2440	
	19	2445	
	20	2450	
	21	2455	
	22	2460	
	23	2465	
	24	2470	
	25	2475	
	26	2480	

Image 4 IEEE802.15.4 radio bands [2]

As for the frames, the IEEE 802.15 standard does not exchange standard Ethernet frames, because most IEEE 802.15 PHYs only support frames up to 128 bytes. The physical frame-format is specified in IEEE 802.15.4.

This is how an IEEE 802.15.4 packet is structured when it reaches the Physical layer:

- Preamble: 4 bytes, used for synchronization;
- Start of Packet Delimiter: 1 byte;
- PHY Header: 1 byte, contains the length of the following data;
- PHY Service Data Unit (**PSDU**): from 0 to 127 bytes, data field.

Image 5 shows how an IEEE 802.15.4 packet is structured:

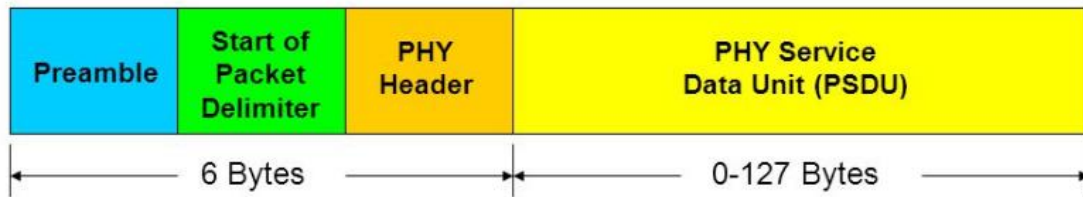


Image 5 IEEE 802.15.4 Packet Structure

The PSDU is created in the MAC layer and there are four different types of PSDU or MAC frames:

- data frame;
- beacon frame;
- acknowledgment frame: confirms the reception of a correct packet, this feature is optional;
- MAC command frame.

The MAC frame is structured as follows (Image 6):

- MAC header: contains information about source and destination addresses, sequence number and frame control. Its length is variable but always lower than 23 bytes;
- MAC Service Data Unit (MSDU): contains the payload, variable length (the whole frame never exceeds 128 bytes);
- MAC footer: frame check sequence, 2 bytes.

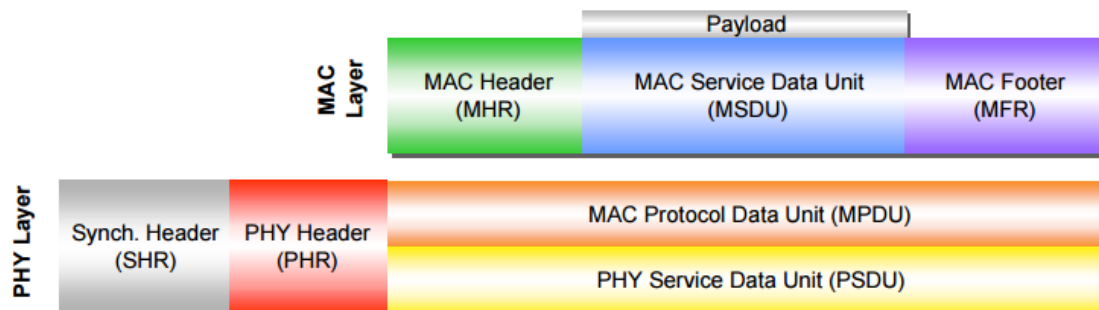


Image 6 IEEE 802.15.4 MAC frame

IEEE 802.15.4 allows also the use of a superframe structure [4]. Thanks to this structure, the coordinator can bound its channel time. A superframe is bounded by beacons and can have active and inactive portions. The coordinator will only communicate during the active period. During the inactive portion, it will enter in a low power mode, in order to save battery. This is how a superframe structure is built:

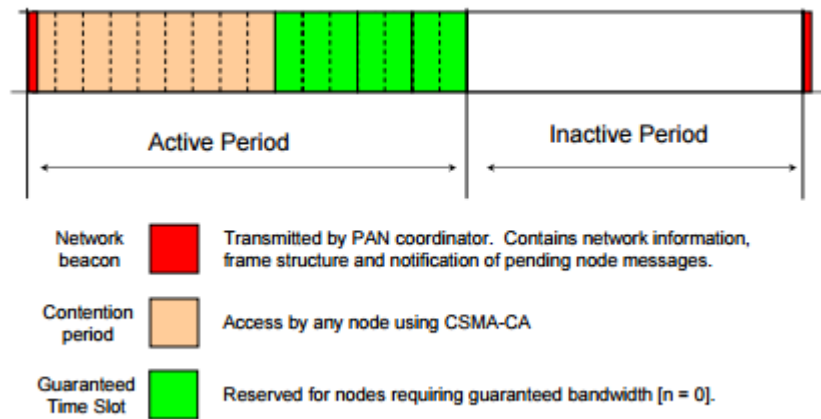


Image 7 Superframe structure [4]

- Beacons: synchronize the attached devices and identify the network and the superframe structure;
- Contention period: access by any node. Every node trying to communicate during the contention period will compete with other devices, due to the slotted CSMA/CA mechanism;
- Guaranteed Time Slots (GTS): for low latency applications or applications requiring specific data bandwidth the coordinator can dedicate portions of the active superframe to that application. These portions are called Guaranteed Time Slots (GTSs) and appear in the end of the active portion of the superframe.

All transactions have to be completed before the time of next network beacon.

As a little summary, this standard allows to use **cheap infrastructure, low energy consumption and small sized hardware**, which are the main requirements for WSNs, in exchange of more limited characteristics, comparing to other wireless standards.

However, IEEE 802.15.4 has some limitations that can make the network not to work properly under some circumstances:

- MAC unreliability and unbounded latency: this effects are increased by the amount of nodes or traffic in the network. A higher traffic will result in higher latencies. This limitation is related to CSMA/CA access method that generates long waiting periods when the channel has a high demand [4];
- no built-in frequency hopping technique: that makes the communication more sensible to noise and multipath-fading.

With this limitations in mind, Task group 4e, working for IEEE 802.15, developed from January 2008 to October 2011 a new version for this standard, called **IEEE 802.15.4e** that solved this issues.

2.2 IEEE 802.15.4e

IEEE 802.15.4e is the result given to improve the support for industrial markets. The major changes are: MAC behavior modes [5] and general functional improvements such as Low Energy (LE), Information Elements (IE), Enhanced Beacons (EB), Multipurpose Frame, MAC Performance Metrics and Fast Association (FastA) [5].

As for the MAC behavior, there are 5 new modes:

- Radio Frequency Identification Blink (BLINK): used for item or people identification, location and tracking;
- Asynchronous Multi-Channel Adaptation (AMCA): applications where large deployments are required;
- Deterministic and Synchronous Multi-Channel Extension (DSME): applications with strict timeliness and reliability requirements;
- Low Latency Deterministic Network (LLDN): for applications requiring very low latencies;
- Time Slotted Channel Hopping (TSCH): applications in industrial domains such as process automation.

The experiments in this thesis run with **TSCH** mode enabled. This mode combines time slotted access, multi-channel communication and channel hopping. To understand how TSCH mode works, it is necessary to understand how it uses time slotted access, how it executes the schedule and how channel hopping is implemented.

Time slotted Access:

Time slotted access makes latencies predictable and bounded and guarantees a bandwidth for each application. Time is divided into time slots, and each time slot have to be long enough for a MAC frame of maximum length to be sent and the acknowledgment (ACK) to be received. The duration of a time slot is not defined in the standard but for radios operating in the 2.4 GHz frequency band, takes about 4 ms to send a maximum length (128 bytes) and 1 ms for the ACK. Using a 10 ms frame slot which is a common value, there is a 5 ms period for packet processing and other operations.

This time slots are grouped into slot frames, which are continuously repeating. The slot frame length may vary, resulting in more available bandwidth and higher power consumption with shorter slot frames. This will be adjusted depending on the requirements of the application.

TSCH schedule:

The schedule tells each node what to do in each time slot: transmit, receive or sleep. The schedule indicates, for each scheduled cell (A cell in a TSCH schedule is an atomic "unit" of resource) [6], a ChannelOffset and the address of the neighbor with which to communicate.

As for the TSCH scheduling, the standard explains how the MAC layer executes a schedule but it doesn't specify how it is built. It can be centralized or distributed scheduling [6].

In centralized scheduling, a manager node is responsible for building and maintaining the network schedule. Every node has to update the manager with the list of nodes it hears and the amount of data it generates. The manager will draw the connectivity graph and will assign slots to different links based on data generation demands.

In distributed scheduling, there is no central entity and each node will decide autonomously on which links to schedule with which neighbors. This scheduling method is suitable for highly dynamic networks. For static networks, centralized schedules are known to be superior to distributed ones.

When the cell is of transmission type, the node checks the buffer and if there is a packet that matches the address of the neighbor written in the schedule information for that time slot, the packet will be transmitted (an ACK may be asked if the option is enabled, and the sending node will have to wait to receive it).

For receive cells, the node listens for possible incoming packets. If there is not packet received after a certain period, the node will shut down its radio.

If the schedule is well built, both the sending node and the receiving node, will transmit and hear in the same time (time slot) and to the same frequency (channel).

It is possible to increase the bandwidth dedicated to a certain link between two nodes scheduling multiple cells to it. The union of all this cells is called a *bundle* and it will repeat over time along with the *slotframe*.

Each transmit cell within the TSCH schedule is dedicated by default. In a dedicated cell, only a certain node can send packets to another. This standard, has the option to create shared cells that allow multiple nodes to communicate at the same time, on the same frequency. A backoff algorithm is defined to avoid contention in this cells.

Taking into account that a node can only transmit, receive or sleep (each operation has a power consumption value associated that will depend on the hardware), given the schedule for a certain node, it is easy to calculate the expected average power consumption.

Channel hopping:

Channel hopping mitigates the effects of interference and multipath fading. With channel hopping enabled, every transmission is done in a different frequency and may help avoiding noises generated by other machines (electric engines for example), especially in industrial environments, that may affect only some channels. This way, changing to another frequency increases the chances to avoid these affected channels.

To explain how channel hopping is done, first, it is necessary to introduce the *Absolut Slot Number* (ASN) [5]. The ASN initializes at 0 when a new network is created and increases by one each time slot. All synchronized nodes in a network have the same ASN.

In a scheduled cell, there is specified a SlotOffset value (used in the computation of the ASN) [5] and a ChannelOffset value [5]. There are as many ChannelOffset values as there are frequencies available (16 when using radios that are compliant with IEEE 802.15.4 at 2.4 GHz when all channels are used).

To calculate the frequency, the ASN and ChannelOffset values are used in a function that looks up on a table with all the available frequencies. The ASN and the ChannelOffset will be the same for both nodes in the scheduled cell, so the same frequency will be computed.

In the next slotframe, even with a static schedule (every scheduled cell repeats over time with constant SlotOffset and ChannelOffset values), the ASN value will change, giving as a result a different frequency for the same scheduled cell.

Channel hopping is a technique known to efficiently combat multi-path fading and external interferences.

2.3 RPL

RPL [7][8] stands for *IPv6 Routing Protocol for Low powered and Lossy Networks*. Which is a **routing protocol** that has been conceived to overcome routing issues in Low Power and Lossy Networks (LLN) as Wireless Sensor Networks (WSN).

The objective of an RPL is to create a topology that minimizes the power consumption of the nodes, avoiding cyclic paths, and finding the most power efficient path. For this objective, a *Destination-Oriented Direct Acyclic Graph* (DODAG) [8] is built. A DODAG is a directed graph, in which all edges are oriented in a way that no cycles exist, and end in a sink called DODAG root (In a DAG, there can be more than one sink but in a DODAG there is only one).

In a DODAG, there are two opposite “directions”:

- “Up”: direction in which information goes from edge nodes to the root;
- “Down”: direction from the root to the edge nodes.

The RPL assigns ranks [8] to the nodes in the DODAG. The rank of a node is its position relative to other nodes with respect to the root. The rank will always decrease in the Down direction and increase in the Up direction. The way of calculate the Rank may vary. It may just consider a simple topological distance, may be calculated as a function of link metrics or may consider other properties such as constraints.

We can define 3 types of traffic in the network:

- Multipoint to point traffic (MP2P): usually the dominant traffic flow in LLNs. The information goes in up direction. It is used to collect data from the network into a sink, the DODAG root commonly;
- Point to multipoint traffic (P2MP): flows in down direction, mostly used when the DODAG root has to inform the rest of the nodes some parameters about the network or needs to ask all the nodes for some data;
- Point to point (P2P): this traffic can be in both directions, it is used when specifically two nodes need to interact (for example, a controller and an actuator). If the node has not routing tables stored, the packet will flow upwards, until it reaches a node that is able to route the packet. As final instance, it will reach the DODAG root.

With this rank methodology a message sent from an edge node to the root, will go always to a lower rank node until it reaches its destination. The route will not have cycles and the power consumption will be minimized, due to the rank gradient created towards the DODAG root.

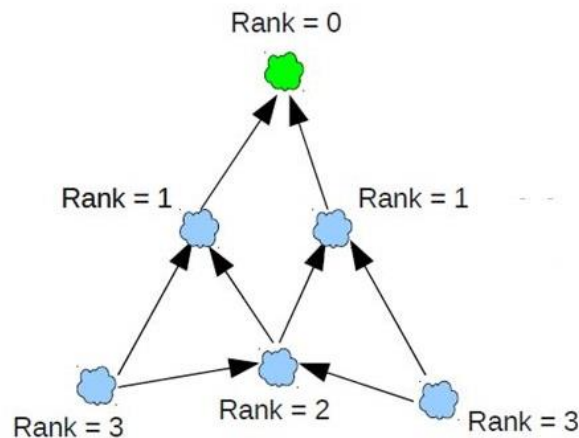


Image 8: RPL network diagram [7]

The RPL defines four types of messages for information exchange and topology maintenance:

- DODAG information exchange (DIO): contains information about the current rank of a node, the current RPL instance, IPv6 address of a node, etc;
- Destination advertisement object (DAO): enables support for down traffic to propagate destination information upwards along the DODAG;
- DODAG information solicitation (DIS): is used to require DIO messages from a reachable neighbor;
- DAO-ACK: is sent by DAO recipient in response to a DAO message.

In order to save battery in LLNs, the RPL, adapts the sending rate of DIO messages according to the stability of the network. In a network with stable links, the control messages will be rare.

2.4 UDP

User Datagram Protocol (UDP) [9] is a **transport level protocol**, based on exchange of datagrams in the environment of an interconnected set of computer networks. This protocol assumes that the Internet Protocol (IP) [9] is used as the underlying protocol. Allows to send packets in a network, without requiring a previously established connection, because the header has the necessary addressing information.

Image 9 shows how a UDP header is built:

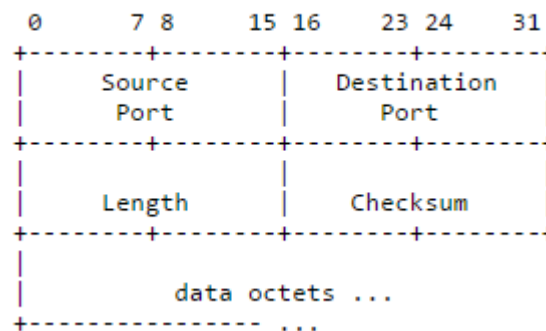


Image 9 User datagram protocol header format

The header has the following fields:

- Source Port: optional field that indicates the port of the sending process. It is the port where a reply will be addressed in the absence of other information. If the field is not used a 0 is inserted;
- Destination Port: it references the port of a particular internet destination address;
- Length: the length in bytes of the user datagram, including the header and the data;
- Checksum: is the 16 bit one's complement of the sum of pseudo header of information from the IP header, the UDP header and the data. This information is used to protect against misrouted datagrams.

The UDP does not have confirmation or flow control and there is no guarantee of delivery, ordering or duplicate protection.

UDP is used when **error checking is not necessary or is done by the application**, avoiding the load of that processing at the network interface level. Usually, it is used in time-sensitive applications because dropping packets is preferable to waiting to delayed packets. Applications requiring ordered reliable delivery of streams of data should use the Transmission Control Protocol (TCP) [9].

Chapter 3: Tools

In this chapter, a deeper description of the tools presented in the introduction is given. On one hand, there is an introduction to the basics of the software and later, the notes that will be tested are described.

3.1 Software: OpenWSN

OpenWSN [10] is an **open-source** implementation of a **standards-based protocol stack** rooted in IEEE 802.15.4e Time Slotted Channel Hopping (TSCH) standard.

It enables ultra-low power and reliable networks that are fully integrated into the internet.

Image 10 shows the protocol stack implemented in OpenWSN. This protocol stack is based entirely on Internet of Things standards.

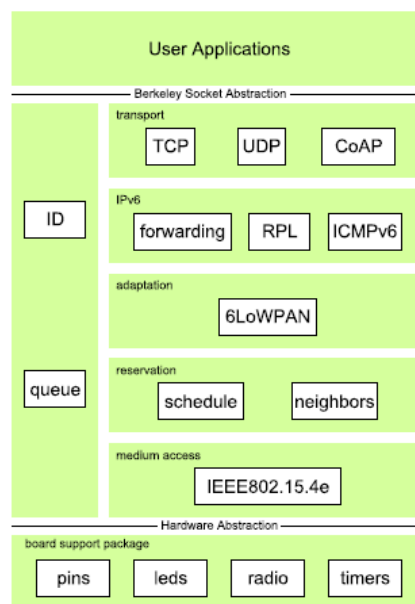


Image 10 The OpenWSN protocol stack [10]

The OpenWSN stack utilizes abstraction at two levels. The Berkeley Socket Abstraction considers that the communication of two applications on two different hosts is through a socket, identified by the IP addresses of the hosts and the two ports corresponding to each application.

The OpenWSN stack respects this abstraction, and developing an application on it is comparable to developing an application on a regular internet host.

The Hardware Abstraction groups all functions concerning the hardware into a group of files called the ‘board support package’ (BSP). Thanks to this, a great part of the **code can be shared among all supported platforms**.

This code portability made possible the use of a similar application written for this experiment for two different platforms, TelosB and Zolertia Z1 that are introduced in the next section of this chapter.

3.1.1 OpenWSN Stack

Protocol layers are wired together to form a stack. Send functions are used to push packets down the stack; Receive functions to pull them up. The bytes of a packet live in the OpenQueue [11] component. A packet is a variable of type OpenQueueEntry_t [11] and is defined in openwsn.h; components pass a pointer to an OpenQueueEntry_t variables in the Send and Receive functions (See “8.10 Stack organization diagram of the OpenWSN protocol stack” on the appendix for a more detailed diagram).

An *OpenQueueEntry_t* is the actual packet that is going to be sent, along with some metadata created by some upper layers that lower layers will need to send the packet down the stack. For example, the MAC layer sets a parameter that informs the driver about which channel it need to transmit the packet, or RPL layer specifies to MAC layer about which neighbor the packet is headed.

To see how it works, here is an example of what each layer does and how the packet flows from one layer to other:

- the application has to get a free *OpenQueueEntry_t* and creates the packet with the payload that is going to be sent. As for the metadata, it will take ownership and creatorship of the packet, it will include source and destination port that will be used by layer 4 (TCP or UDP) and destination address that will be used in layer 3 by RPL. Finally it sends the packet to layer 4;
- layer 4 (UDP in this thesis), will take the ownership of the packet and it will write where the payload starts and how long it is. Then it will add UDP header with the source and destination addresses written by the application layer. Finally, sends the packet to RPL;
- RPL in layer 3, takes ownership of the packet and fills the metadata with data for the next hop. It forwards the packet to IPHC;
- IPHC (Internet Protocol Header Compression) [11] belongs to the adaptation layer needed to compress the internet header into a smaller header that IEEE 802.15.4 standard can handle (6LoWPAN header). After adding this header, it sends the packet to layer 2;
- in layer 2, the MAC protocol (IEEE 802.15.4e) takes ownership of the packet. It fills the metadata with the amount of retries left, the power for the radio antenna and the channel (obtained from *schedule.c* [11]). It will add the MAC

header with the next hop information and frame type information and the packet is sent to the drivers;

- the drivers or physical layer will configure the radio power and the channel. It won't take ownership of the packet, so if something goes wrong, MAC will be able to retransmit the packet.

3.1.2 Open Visualizer

OpenVisualizer [12] is the primary tool for plugging OpenWSN network into the Internet.

These are the main features:

- connects OpenWSN network to the Internet over a virtual interface (both Windows and Linux);
- portable across popular operating systems;
- shows the internal state (neighbor table, scheduling table, queue, etc.) of each node physically connected to the OpenVisualizer;
- displays errors reported by motes;
- can run with either physical motes, or emulated motes.

The software is based on publish-subscribe messaging between components in a Python process. The diagram in Image 11 provides a high-level view. The Event Bus provides the messaging framework. Specific components implement services like connection of wireless motes via serial connection, and external or internal Python based applications can be used (The application used for this thesis, UDPlatency is an internal Python process). Also, notice that several motes may be connected simultaneously.

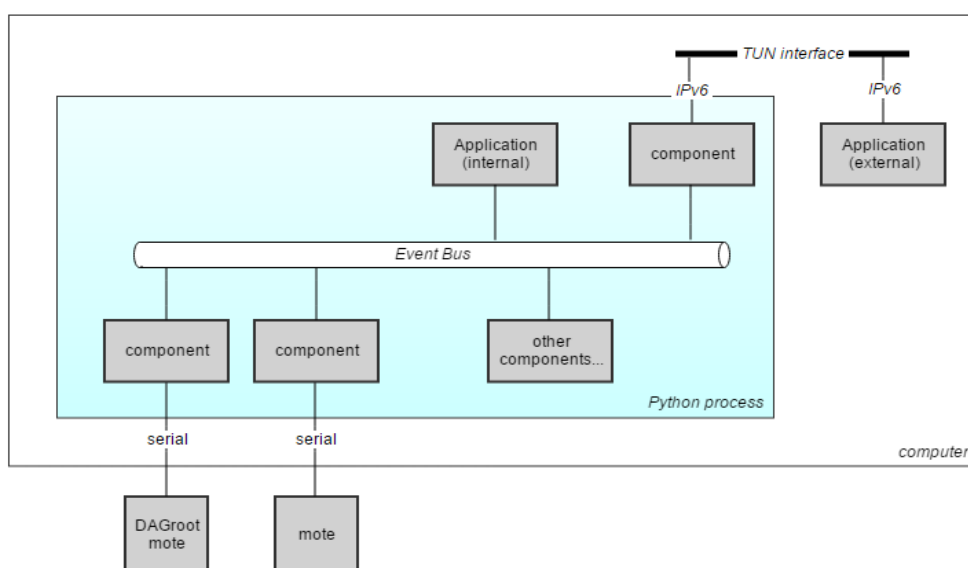


Image 11 Diagram of OpenVisualizer's architecture [12]

This is how OpenVisualizer has worked in the tests:

- the root of the network received the packets and they were captured by the component *moteConnector* [12] via USB;
- *moteConnector* extracts the data from the received packet and it is published in the event bus;
- the *UDPlatency* script captures this data and updates the parameters with the new values (new latencies, sequence number, PLR, etc.) and saves them in a text file.

After the text file is created the process ends and from this point the user can use the data generated freely.

3.1.3 UDPlatency

UDPlatency is an application [Code A1] written by students of *Politecnico di Bari*. UDPlatency is an application that consists of two parts. The first one, an application written in the firmware that will run the mote, and the other part that will run with OpenVisualizer in the computer and collects the data that receives the root mote.

The firmware part has to create the **128 byte long packets** that will be sent by every mote (On the following section is defined how packets are filled) to the network coordinator. The software application will collect the data via **USB serial communication**, then it will obtain certain parameters and finally creates a text file that saves this parameters.

This is the data that shows the text file:

- address of the sender mote;
- minimum latency;
- maximum latency;
- number of packets received;
- number of packets sent;
- average latency;
- latest latency: latency of the last received packet;
- temperature value;
- sequence number: the sequence number increases each time a new packet is sent, it is used to calculate the amount of duplicated packets;
- number of duplicated packets;
- packet lost rate (PLR): PLR without taking into account the duplicated packets. An after-work is necessary to calculate the real PLR;
- received time: Data and time when the packet was received.

In the Image 12 there is an example of the data generated.

23			
24	Mote address	0x14,0x15,0x92,0x0,0x0,0x15,0x16,0x14	
25	Min latency	285ms	
26	Max latency	4680ms	
27	Packets received		9
28	Packets sent		22
29	Avg latency	1291ms	
30	Latest latency	1320ms	
31	Temperature	[29]	
32	Sequence Number		21
33	Duplicated packets		0
34	PLR	59.0909090909%	
35	Received	2016-06-24 11:50:05.722480	
36			

Image 12 Data obtained from the script

3.1.4 Packets

To get this data, the packets were filled as follows (Code A1):

- Absolut Slot Number (ASN): it is used to calculate the latency, 5 bytes;
- ID of the mote: 64 bit address of the mote, 8 bytes;
- temperature value: 1 byte;
- 7 dummy bytes: they were originally used to send the address of the preferred parent, along with the byte used for the temperature;
- Sequence Number: it is used to calculate the amount of lost packets, 2 bytes;
- 17 dummy bytes: they are used to complete the 128 byte payload.

The rest of the data in the packet is occupied by the header and metadata added by other components in other layers:

- Creator: 1 byte;
- Owner: 1 byte;
- Sequence Number: 2 bytes;
- layer 4, protocol: 1 byte;
- layer 4, source port: 2 bytes;
- layer 4, destination port: 2 bytes;
- layer 4, payload: 1 byte;
- layer 4, length: 1 byte;
- layer 3, destination address: 16 bytes;
- layer 3, source address: 16 bytes;
- layer 2, next hop: 8 bytes;
- layer 2, previous hop: 8 bytes;
- layer 2, frame type: 1 byte;
- layer 2, dsn [11]: 1 byte;
- layer 2, retries left: 1 byte;
- layer 2, number of tx attempts: 1 byte;

- layer 2, ASN: 5 bytes;
- layer 2, cell objects: 1 byte;
- layer 2, number of cells: 1 byte;
- layer 2, frame ID: 1 byte;
- layer 2, join priority: 1 byte;
- layer 2, security: 1 byte;
- layer 2, security level: 1 byte;
- layer 2, key ID mode: 1 byte;
- layer 2, key source: 8 bytes;
- layer 1, tx power [11]: 1 byte;
- layer 1, rssi [11]: 1 byte;
- layer 1, lqi [11]: 1 byte;
- layer 1, crc [11]: 1 byte;
- layer 1, rx time stamp [11]: 1 byte.

3.1.5 Configuration

OpenWSN gives the opportunity to modify some parameters in order to adapt the communications to the desired situation. In this thesis it was used the following configuration (Codes B1 and C1):

- super-frame length: 101 slots. It is done to get a low duty cycle (<1%) that will ensure a low power consumption;
- Channel Hopping: enabled;
- synchronization channel: 23, it was changed from the default (channel 20) in order to avoid interferences with other motes in the laboratory;
- power of the radio: 0 db;
- ADV timeout: ADV sent every 5 seconds;
- maximum keep-alive period: 7,5 seconds.

Other parameters as number of retries and the number of shared channels are going to be modified during the tests.

As a summary, it would be interesting to see that OpenWSN is a very suitable OS for this tests. It is a software designed for WSNs that has adopted IEEE 802.15.4e in its core. This standard suits perfectly for LLNs more specially in industrial and noisy environments. An interesting application could be monitoring a production process in an industrial plant.

Besides, it is created to favor portability between different platforms, which makes the OS very interesting to test different motes as it is in this case. It allows to implement different applications in a simple way and many communication parameters are easy to modify, giving a huge freedom that allows to make tests for many purposes and applications, making OpenWSN a very useful tool for researchers.

3.2 Hardware

In this project 2 different motes are characterized. The TelosB by Memsic and the Z1 by Zolertia.

3.2.1 TelosB

MEMSIC's TelosB Mote [13] is an open-source platform designed to enable cutting-edge experimentation for the research community.

The TelosB bundles all the essentials for lab studies into a single platform including: USB programming capability, an IEEE 802.15.4 radio with integrated antenna, a low-power MCU with extended memory and an optional sensor suite. TelosB offers many features, including:

- RF transceiver: IEEE 802.15.4 compliant;
- ISM band: 2.4 to 2.4835 GHz;
- data rate: 250 kbps;
- antenna: integrated onboard;
- microcontroller: 8 MHz TI MSP430;
- RAM: 10 kB;
- low current consumption;
- external flash: 1 MB;
- programming and data collection via USB;
- sensor suite: including integrated light, temperature and humidity sensors.

The TelosB platform was developed and published to the research community by UC Berkeley. This platform delivers low power consumption allowing for **long battery life** as well as fast wakeup from sleep state.

It is powered by two AA batteries in serial configuration that supply the mote with 3 V. If the TelosB is plugged into the USB port for programming or communication, power is provided from the host computer.

In Image 13, there is a block diagram that shows the main modules integrated in TelosB.

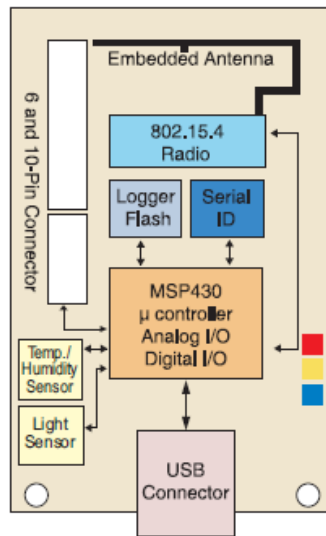


Image 13 TelosB block diagram

Characteristics of the TelosB:

<i>Specifications</i>	
<i>Processor Performance</i>	16 bit RISC
<i>Processor clock</i>	8 MHz
<i>Program Flash Memory</i>	48 kB
<i>External Flash</i>	1024 kB
<i>RAM</i>	10 kB
<i>Configuration EEPROM</i>	16 kB
<i>Serial Communications</i>	UART
<i>Analog to Digital Converter</i>	12 bit ADC
<i>Digital to Analog Converter</i>	12 bit DAC
<i>Expansion connector</i>	16 Pins
<i>Processor current draw</i>	1.8 mA (Active) 5.1 μ A (Sleep)
<i>Frequency band</i>	2400 MHz to 2483.5 MHz
<i>Transmit (TX) data rate</i>	250 kbps
<i>RF power</i>	-24 dBm to 0 dBm
<i>Transceiver current draw</i>	23 mA (active) 1 μ A (Sleep)
<i>Temperature sensor range</i>	-40 $^{\circ}$ C to 123.8 $^{\circ}$ C
<i>Resolution</i>	0.01 $^{\circ}$ C
<i>Accuracy</i>	\pm 0.5 $^{\circ}$ C
<i>Battery</i>	2x AA batteries
<i>User interface</i>	USB

Processor and memory specifications are very low compared with other applications nowadays such as smartphones. However, as mentioned in previous sections, these WSNs don't seek high performance but low power consumption and cheap hardware.

Besides, TelosB works in the 2.4 GHz frequency band which is one of the bands that uses the IEEE 802.15 standard. It has an industrial temperature range compatible temperature sensor from Texas Instruments and very low current consumptions that are the features we are looking for.

Finally, its USB interface makes TelosB easy to program and extract the data from the tests, which makes it really interesting for research purposes.

3.2.2 Zolertia Z1

Z1 by Zolertia [14] (Image 14) is a low-power wireless sensor network (WSN) module that serves as a general purpose development platform for WSN developers and researchers.

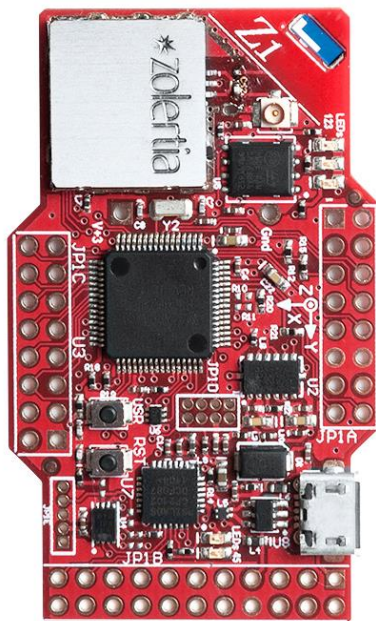


Image 14 Zolertia Z1 [14]

The Z1 is a low power wireless module compliant with IEEE 802.15.4 and Zigbee protocols intended to help WSN developers to test and deploy their own applications and prototypes with the best tradeoff between time of development and hardware flexibility.

Its core architecture is based upon the MSP430+CC2420 family of microcontrollers and radio transceivers by Texas Instruments, which makes it compatible with motes based on this same architecture. These are the main features:

- expansion connector: 52 pin;
- microcontroller: 16 MHz 2nd generation MSP430™;
- ISM band: 2.4 GHz;
- temperature sensor: TI ZIG001 industrial range digital temperature and humidity sensor;
- optional external antenna: U.FL connector;
- Micro-USB connector: for power and debugging.

The Z1 WSN Module is specified to be used in the industrial range of temperatures. Nominally, it should be powered at 3V, although it may work partially or totally since 1.8 V (without radio) or 2.1 V (with radio).

Image 15 shows a block diagram with the main modules of the Z1.

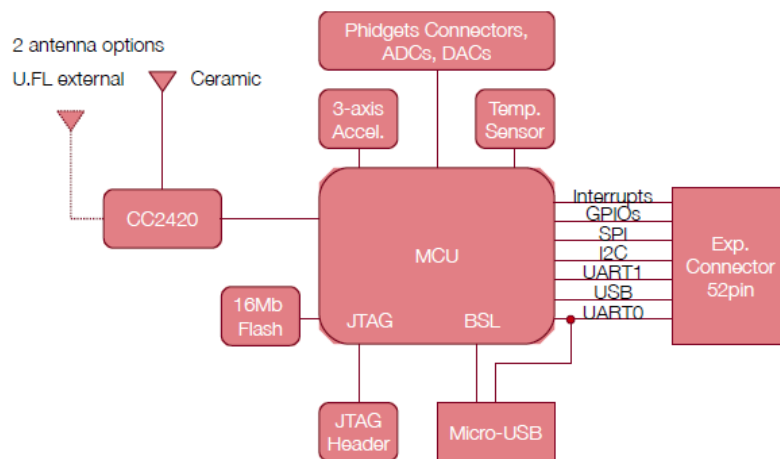


Image 15 Zolertia Z1 block diagram

Zolertia Z1 characteristics:

<i>Specifications</i>	
<i>Processor Performance</i>	16 bit MCU
<i>Processor clock</i>	16 MHz
<i>Program Flash Memory</i>	92 kB
<i>External Flash</i>	2048 kB
<i>RAM</i>	10 kB
<i>Serial Communications</i>	UART
<i>Analog to Digital Converter</i>	8 bit ADC
<i>Digital to Analog Converter</i>	2 bit DAC
<i>Expansion connector</i>	52 Pins
<i>Processor current draw</i>	<1 mA (Active) 0.5 μ A (Sleep)
<i>Frequency band</i>	2400 MHz to 2483.5 MHz
<i>Transmit (TX) data rate</i>	250 kbps
<i>RF power</i>	-24 dBm to 0 dBm
<i>Transceiver current draw</i>	18.8 mA (active) 1 μ A (Sleep)
<i>Temperature sensor range</i>	-25 $^{\circ}$ C to 85 $^{\circ}$ C
<i>Resolution</i>	0.1 $^{\circ}$ C
<i>Accuracy</i>	\pm 0.5 $^{\circ}$ C
<i>Battery</i>	2x AA batteries
<i>User interface</i>	Micro USB

As for the Zolertia Z1, it has similar characteristics that the TelosB, with some differences:

- 16 MHz clock speed (vs 8 MHz);
- built-in clock factory calibration;
- 92 kB flash (vs 48 kB);
- lower power consumption (Half);
- 10 kB RAM (vs 8 kB);
- 2 MB external flash memory (vs 1 MB);
- most of the MSP ports are visible (all the ADC and DAC);
- USB pins are visible to be able to use another kind of USB connector;
- calibration tables are not deleted during programming.

Zolertia Z1, with improved characteristics that the TelosB, is a very interesting choice for this tests in order to compare a more powerful board using the same OS, so it is visible how a better hardware affects to the overall performance.

Chapter 4: Tests

In this chapter, all the tests done for the thesis are given, explaining how the experiments have been conducted and giving the results in tables and graphics with a commentary of the obtained results.

4.1 Aim of the tests

The main objective of the thesis is to characterize the two motes running OpenWSN. Focused on the latency and the packet lost rate (PLR).

All the results are given for each mote, and the average values for the network, making possible to cross-compare both platforms.

For the latencies, it is given the **maximum, minimum and average latency**.

To calculate the PLR, the amount of packets sent, received and duplicated is needed, using the following expression:

$$PLR (\%) = \frac{Packets\ sent - Unrepeated\ received\ packets}{Packets\ sent} \times 100 = \frac{Packets\ lost}{Packets\ sent} \times 100$$

Equation 1: PLR

It is also necessary to take into account the number and duration of desynchronizations, because when one mote desynchronizes, the amount of traffic in the network will change and the results won't be comparable.

Both, latency and PLR are obtained at application layer. The latencies are calculated as it follows: from the moment a message is sent from application layer, until it reaches the application layer of the receiving mote. Due to this condition, retransmission on MAC layer are not taken into account, so very high latencies may appear if a packet is retransmitted many times.

Retransmissions on MAC layer, are not taken into account for the calculation of the PLR, and only those sent from the application layer of the sending mote and received by the application layer of the receiving mote are considered.

Finally, it is shown the impact that the firmware has in the memory of the motes, to see how each change affects to the mote.

4.2 Realization of the tests

To make this results comparable, all tests must be done under the same conditions.

The tests consist in:

- number of motes: 5 motes plus one as the DAG root of the network;
- duration: 30 minutes, divided in 10 minutes of warm up, and 20 minutes of test;
- topology: star topology.

Apart from these common aspects, there are some configurations that may vary from one test to another, in order to obtain the desired results. These are the parameters that will be modified:

- number of retransmissions;
- amount of shared channels;
- period of transmission of the messages;
- the payload.

All the tests where done twice, one for the TelosB and the other for the Z1.

4.3 Test 1

The aim of the first test is to obtain some comparable **latencies** and **PLR** values for both notes.

Codes used in this tests are shown in the appendix A1, B1 and C1. For the Z1 see also code A2.

Some initial conditions are chosen:

Number of retransmissions:	3
Amount of shared channels:	6
Period of transmission (s):	4
Payload (bytes):	127

Table 1: test 1, configuration

TelosB:

Memory impact:

ROM (bytes)	44184
RAM (bytes)	6902

Table 2: test 1, memory impact for TelosB

Data obtained from test 1 for TelosB:

<i>N</i>	<i>Mote</i>	<i>Pack. Sent</i>	<i>Pack. Rec.</i>	<i>Pack. Dupl.</i>	<i>Rec.-Dupl.</i>	<i>PLR (%)</i>
1	1d	256	140	15	125	51,17
2	a2	287	91	10	81	71,78
3	7	297	137	15	122	58,92
4	14	293	83	4	79	73,04
5	b7	292	112	14	98	66,44
<i>Network</i>		1425	563	58	505	64,56

<i>N</i>	<i>Desynch.</i>	<i>Time desynch. (s)</i>	<i>Avg latency (ms)</i>	<i>Max latency (ms)</i>	<i>Min latency (ms)</i>
1	0	0	10582,71	137085	285
2	2	85	6507,86	58890	390
3	0	0	9538,91	262065	315
4	2	95	5390,42	71895	285
5	1	45	11097,59	240585	315
<i>Network</i>			8927,75	262065	285

Table 3: test 1, obtained data for TelosB

Z1:

Memory impact:

ROM (bytes)	37926
RAM (bytes)	6966

Table 4: test 1, memory impact for Z1

Data obtained from test 1 for Z1:

<i>N</i>	<i>Mote</i>	<i>Pack. Sent</i>	<i>Pack. Rec.</i>	<i>Pack. Dupl.</i>	<i>Rec.-Dupl.</i>	<i>PLR (%)</i>
1	10	297	127	15	112	62,29
2	20	295	124	8	116	60,68
3	30	295	73	5	68	76,95
4	40	298	122	10	112	62,42
5	50	290	70	4	66	77,24
<i>Network</i>		1475	516	42	474	67,86

<i>N</i>	<i>Desynch.</i>	<i>Time desynch. (s)</i>	<i>Avg latency (ms)</i>	<i>Max latency (ms)</i>	<i>Min latency (ms)</i>
1	0	0	7881,05	118575	45
2	0	0	13028,59	436155	45
3	1	60	5314,73	91425	105
4	0	0	10784,88	284190	30
5	1	70	7838,36	81600	135
<i>Network</i>			9357,5	436155	30

Table 5: test 1, obtained data for Z1

These results show very high latencies and inadmissible PLR values, more than 60 % of the packets sent never reach their destination.

The PLR increments linearly along with the time desynchronized, resulting in a higher amount of packets lost when a mote loses synchronization.

On the other hand, many desynchronizations happened in different moments during the test, so is not possible to compare the results among both platforms, because the amount of motes transmitting data varies.

To solve this problem, it is shown a time window without desynchronizations.

This are the results obtained in the 4 minute window of the test 1:

TelosB:

<i>N</i>	<i>Mote</i>	<i>Pack. Sent</i>	<i>Pack. Rec.</i>	<i>Pack. Dupl.</i>	<i>Rec.-Dupl.</i>	<i>PLR (%)</i>
1	1d	40	22	4	18	55,00
2	a2	40	13	1	12	70,00
3	7	47	12	0	12	74,47
4	14	39	13	1	12	69,23
5	b7	40	13	2	11	72,50
<i>Network</i>		206	73	8	65	68,45

<i>N</i>	<i>Desynch.</i>	<i>Time desynch. (s)</i>	<i>Avg latency (ms)</i>	<i>Max latency (ms)</i>	<i>Min latency (ms)</i>
1	0	0	3182,61	7830	390
2	0	0	3264,64	8340	570
3	0	0	4987,50	27615	525
4	0	0	3178,93	15660	840
5	0	0	3536,25	10545	315
<i>Network</i>			3542,60	27615	315

Table 6: test 1 (window without desynchronizations), obtained data for TelosB

Z1:

<i>N</i>	<i>Mote</i>	<i>Pack. Sent</i>	<i>Pack. Rec.</i>	<i>Pack. Dupl.</i>	<i>Rec.-Dupl.</i>	<i>PLR (%)</i>
1	10	46	22	4	18	60,87
2	20	46	23	3	20	56,52
3	30	37	18	2	16	56,76
4	40	50	13	0	13	74,00
5	50	47	10	0	10	78,72
<i>Network</i>		226	86	9	77	65,93

<i>N</i>	<i>Desynch.</i>	<i>Time desynch. (s)</i>	<i>Avg latency (ms)</i>	<i>Max latency (ms)</i>	<i>Min latency (ms)</i>
1	0	0	3182,61	15525	450
2	0	0	2429,25	8205	45
3	0	0	2280,00	6150	180
4	0	0	3653,18	12990	180
5	0	0	2467,50	4440	270
<i>Network</i>			2770,66	15525	45

Table 7: test 1 (Window without desynchronizations), obtained data for Z1

The average latency obtained is 3 times lower and a reduction of the maximum latency of one order of magnitude.

On the other hand, the values for the PLR does not change significantly.

These results show a better performance of the Z1 in these conditions:

- average latency is 27.86 % lower for the Z1;
- PLR is 2.52 % lower for the Z1.

Anyway, the packet lost rate is unacceptable, (in this thesis, a PLR around 10 % will be considered as acceptable) further tests are needed.

4.4 Test 2

In this second test the retransmissions are deleted with the aim of lowering the data traffic at MAC layer level.

The changes made are shown in code B2.

Number of retransmissions:	0
Amount of shared channels:	6
Period of transmission (s):	4
Payload (bytes):	127

Table 8: test 2, configuration

TelosB:

Memory impact:

ROM (bytes)	44166
RAM (bytes)	6902

Table 9: test 2, memory impact for TelosB

Data obtained from test 2 for TelosB:

<i>N</i>	<i>Mote</i>	<i>Pack. Sent</i>	<i>Pack. Rec.</i>	<i>Pack. Dupl.</i>	<i>Rec.-Dupl.</i>	<i>PLR (%)</i>
1	1d	295	79	0	79	73,22
2	a2	296	17	0	17	94,26
3	7	292	67	0	67	77,05
4	14	289	59	0	59	79,58
5	b7	288	29	0	29	89,93
<i>Network</i>		1460	251	0	251	82,81

<i>N</i>	<i>Desynch.</i>	<i>Time desynch. (s)</i>	<i>Avg latency (ms)</i>	<i>Max latency (ms)</i>	<i>Min latency (ms)</i>
1	0	0	722,81	2610	285
2	0	0	776,67	1740	270
3	0	0	818,82	1650	270
4	0	0	970,25	2040	285
5	0	0	863,50	1680	300
<i>Network</i>			826,58	2610	270

Table 10: test 2, obtained data for TelosB

Z1:

Memory impact:

ROM (bytes)	37908
RAM (bytes)	6966

Table 11: test 2, memory impact for Z1

Data obtained from test 2 for Z1:

<i>N</i>	<i>Mote</i>	<i>Pack. Sent</i>	<i>Pack. Rec.</i>	<i>Pack. Dupl.</i>	<i>Rec.-Dupl.</i>	<i>PLR (%)</i>
1	10	311	137	0	137	55,95
2	20	295	42	0	42	85,76
3	30	270	14	0	14	94,81
4	40	283	50	0	50	82,33
5	50	313	92	0	92	70,61
Network		1472	335	0	335	77,24

<i>N</i>	<i>Desynch.</i>	<i>Time desynch. (s)</i>	<i>Avg latency (ms)</i>	<i>Max latency (ms)</i>	<i>Min latency (ms)</i>
1	0	0	721,41	1485	45
2	0	0	659,65	2940	45
3	0	0	337,00	1455	30
4	0	0	493,82	1410	30
5	0	0	520,97	2025	30
Network			607,68	2940	30

Table 12: test 2, obtained data for Z1

By lowering the traffic, the latencies decreased by one order of magnitude. This might be because we lower the amount of collisions and the motes are less busy transmitting and receiving data.

On the other hand and as expected, the PLR has worsen. There is a gain of 18.25 % of PLR for the TelosB and 9.38 % for the Z1. With no retransmissions, each message has one try to reach their destination.

It is also important to notice that reducing the traffic, has made the network more stable, avoiding desynchronizations.

4.5 Test 3

For the next test, the number of available channels is increased, trying to lower the latency.

The changes made are in code C2.

Number of retransmissions:	3
Amount of shared channels:	12
Period of transmission (s):	4
Payload (bytes):	127

Table 13: test 3, configuration

TelosB:

Memory impact:

ROM (bytes)	44196
RAM (bytes)	7106

Table 14: test 3, memory impact for TelosB

Data obtained from test 3 for TelosB:

<i>N</i>	<i>Mote</i>	<i>Pack. Sent</i>	<i>Pack. Rec.</i>	<i>Pack. Dupl.</i>	<i>Rec.-Dupl.</i>	<i>PLR (%)</i>
1	1d	267	98	9	89	66,67
2	a2	290	94	10	84	71,03
3	7	297	144	20	124	58,25
4	14	159	41	0	41	74,21
5	b7	296	133	8	125	57,77
<i>Network</i>		1309	510	47	463	64,63

<i>N</i>	<i>Desynch.</i>	<i>Time desynch. (s)</i>	<i>Avg latency (ms)</i>	<i>Max latency (ms)</i>	<i>Min latency (ms)</i>
1	1	5	3858,52	45015	330
2	1	60	4199,67	45150	270
3	1	5	3228,21	38310	375
4	2	90	5204,63	55080	315
5	0	0	4439,22	59685	315
<i>Network</i>			4001,65	59685	270

Table 15: test 3, obtained data for TelosB

Z1:

Memory impact:

ROM (bytes)	37938
RAM (bytes)	7170

Table 16: test 3, memory impact for TelosB

Data obtained from test 3 for Z1:

<i>N</i>	<i>Mote</i>	<i>Pack. Sent</i>	<i>Pack. Rec.</i>	<i>Pack. Dupl.</i>	<i>Rec.-Dupl.</i>	<i>PLR (%)</i>
1	10	306	146	17	129	57,84
2	20	298	115	13	102	65,77
3	30	285	52	6	46	83,86
4	40	306	107	16	91	70,26
5	50	284	65	4	61	78,52
<i>Network</i>		1479	485	56	429	70,99

<i>N</i>	<i>Desynch.</i>	<i>Time desynch. (s)</i>	<i>Avg latency (ms)</i>	<i>Max latency (ms)</i>	<i>Min latency (ms)</i>
1	0	0	3239,69	30810	30
2	2	40	4857,35	40110	45
3	3	90	3647,02	27255	45
4	1	30	3629,58	59460	30
5	3	60	4104,84	32025	30
<i>Network</i>			3866,91	59460	30

Table 17: test 3, obtained data for Z1

As expected, the latency has dropped to half regarding to the first test but the value is still quite high.

As for the PLR, there are similar results, having a small increase in packets lost.

Comparing with the first test, the PLR also increases linearly along with the desynchronization time, with a similar slope, being this higher for the Z1.

4.6 Test 4

Increasing the amount of available channels hasn't solved the problem, so a decrease in the amount of traffic is needed. Taking into account that retransmissions can't be disabled because the PLR goes up, the traffic in application layer is going to be reduced, increasing the period of the transmission of the packets.

The changes made are in code A3.

Number of retransmissions:	3
Amount of shared channels:	6
Period of transmission (s):	10
Payload (bytes):	127

Table 18: Test 4, configuration

TelosB

Memory impact:

ROM (bytes)	44184
RAM (bytes)	6902

Table 19: test 4, memory impact for TelosB

Data obtained from test 4 for TelosB:

<i>N</i>	<i>Mote</i>	<i>Pack. Sent</i>	<i>Pack. Rec.</i>	<i>Pack. Dupl.</i>	<i>Rec.-Dupl.</i>	<i>PLR (%)</i>
1	1d	117	55	3	52	55,56
2	a2	116	82	11	71	38,79
3	7	121	96	16	80	33,88
4	14	112	53	5	48	57,14
5	b7	118	70	8	62	47,46
<i>Network</i>		584	356	43	313	46,40

<i>N</i>	<i>Desynch.</i>	<i>Time desynch. (s)</i>	<i>Avg latency (ms)</i>	<i>Max latency (ms)</i>	<i>Min latency (ms)</i>
1	0	0	2246,79	6720	390
2	0	0	2023,92	6465	270
3	0	0	2307,99	9990	270
4	0	0	2265,56	9165	75
5	0	0	2460,42	7395	300
<i>Network</i>			2256,81	9990	75

Table 20: test 4, obtained data for TelosB

Z1:

Memory impact:

ROM (bytes)	37926
RAM (bytes)	6966

Table 21: test 4, memory impact for Z1

Data obtained from test 4 for Z1:

<i>N</i>	<i>Mote</i>	<i>Pack. Sent</i>	<i>Pack. Rec.</i>	<i>Pack. Dupl.</i>	<i>Rec.-Dupl.</i>	<i>PLR (%)</i>
1	10	116	74	8	66	43,10
2	20	114	75	13	62	45,61
3	30	119	58	5	53	55,46
4	40	113	38	2	36	68,14
5	50	116	58	7	51	56,03
<i>Network</i>		578	303	35	268	53,63

<i>N</i>	<i>Desynch.</i>	<i>Time desynch. (s)</i>	<i>Avg latency (ms)</i>	<i>Max latency (ms)</i>	<i>Min latency (ms)</i>
1	0	0	1937,20	10995	30
2	0	0	6995,07	43710	45
3	0	0	1668,56	7470	30
4	0	0	1556,07	5625	30
5	0	0	1414,32	4530	75
<i>Network</i>			2905,83	43710	30

Table 22: test 4, obtained data for Z1

Reducing the traffic at the application layer has reduced both the latency and the PLR. The traffic congestion has been reduced and retransmissions are still enabled to ensure that the PLR doesn't increase a lot.

There is a decrease of 18.16 % of the PLR for the TelosB and 14.23 % for the Z1 regarding the first test and four times lower latencies.

Apart from that, the problem of the desynchronizations is gone, the same as in the test 2, due to the lower traffic.

Anyway, the PLR must be lower for this set up to be useful.

4.7 Test 5

In the following test, the period of 10 seconds is maintained and the data payload is lowered taking away 17 bytes (13.28 % of the payload) that were included in the packet just to complete the 127 bytes.

The changes made are in code A3.

Number of retransmissions:	3
Amount of shared channels:	6
Period of transmission (s):	10
Payload (bytes):	110

Table 23: test 5, configuration

TelosB:

Memory impact:

ROM (bytes)	44154
RAM (bytes)	6902

Table 24: test 5, memory impact for TelosB

Data obtained from test 5 for TelosB:

<i>N</i>	<i>Mote</i>	<i>Pack. Sent</i>	<i>Pack. Rec.</i>	<i>Pack. Dupl.</i>	<i>Rec.-Dupl.</i>	<i>PLR (%)</i>
1	1d	118	75	9	66	44,07
2	a2	115	67	8	59	48,70
3	7	118	46	4	42	64,41
4	14	111	62	3	59	46,85
5	b7	119	82	12	70	41,18
<i>Network</i>		581	332	36	296	49,05

<i>N</i>	<i>Desynch.</i>	<i>Time desynch. (s)</i>	<i>Avg latency (ms)</i>	<i>Max latency (ms)</i>	<i>Min latency (ms)</i>
1	0	0	2358,16	10005	300
2	0	0	2331,84	8475	210
3	0	0	2231,49	7080	270
4	0	0	2255,71	6825	285
5	0	0	2058,89	5535	300
<i>Network</i>			2243,42	10005	210

Table 25: test 5, obtained data for TelosB

Z1:

Memory impact:

ROM (bytes)	37900
RAM (bytes)	6966

Table 26: test 5, memory impact for Z1

Data obtained from test 5 for Z1:

<i>N</i>	<i>Mote</i>	<i>Pack. Sent</i>	<i>Pack. Rec.</i>	<i>Pack. Dupl.</i>	<i>Rec.-Dupl.</i>	<i>PLR (%)</i>
1	10	116	92	13	79	31,90
2	20	118	56	5	51	56,78
3	30	114	72	8	64	43,86
4	40	115	55	3	52	54,78
5	50	113	55	6	49	56,64
<i>Network</i>		576	330	35	295	48,78

<i>N</i>	<i>Desynch.</i>	<i>Time desynch. (s)</i>	<i>Avg latency (ms)</i>	<i>Max latency (ms)</i>	<i>Min latency (ms)</i>
1	0	0	1387,74	5535	45
2	0	0	1811,05	4560	30
3	0	0	1497,53	6255	30
4	0	0	1356,96	6930	30
5	0	0	1743,75	4440	30
<i>Network</i>			1538,06	6930	30

Table 27: test 5, obtained data for Z1

The results are comparable to the previous test. There is a minor improvement in the latency but the PLR has worsen.

Lowering the payload has not brought any significant changes so the full payload is kept.

4.8 Test 6

Looking at the results obtained in the previous tests, the most effective way to lower the PLR is to increase the period. The period is increased to 30 s.

The changes made are in code A4.

Number of retransmissions:	3
Amount of shared channels:	6
Period of transmission (s):	30
Payload (bytes):	127

Table 28: test 6, configuration

TelosB:

Memory impact:

ROM (bytes)	44212
RAM (bytes)	6904

Table 29: test 6, memory impact for TelosB

Data obtained from test 6 for TelosB:

<i>N</i>	<i>Mote</i>	<i>Pack. Sent</i>	<i>Pack. Rec.</i>	<i>Pack. Dupl.</i>	<i>Rec.-Dupl.</i>	<i>PLR (%)</i>
1	1d	35	34	8	26	25,71
2	a2	38	33	9	24	36,84
3	7	37	24	4	20	45,95
4	14	39	23	2	21	46,15
5	b7	39	42	14	28	28,21
<i>Network</i>		188	156	37	119	36,70

<i>N</i>	<i>Desynch.</i>	<i>Time desynch. (s)</i>	<i>Avg latency (ms)</i>	<i>Max latency (ms)</i>	<i>Min latency (ms)</i>
1	0	0	2561,14	12510	300
2	0	0	2923,24	8490	360
3	0	0	2671,20	6885	315
4	0	0	2961,88	10665	315
5	0	0	2494,53	8640	345
<i>Network</i>			2696,65	12510	300

Table 30: test 6, obtained data for TelosB

Z1:

Memory impact:

ROM (bytes)	37952
RAM (bytes)	6968

Table 31: test 6, memory impact for Z1

Data obtained from test 6 for Z1:

<i>N</i>	<i>Mote</i>	<i>Pack. Sent</i>	<i>Pack. Rec.</i>	<i>Pack. Dupl.</i>	<i>Rec.-Dupl.</i>	<i>PLR (%)</i>
1	10	38	29	5	24	36,84
2	20	39	40	13	27	30,77
3	30	37	26	5	21	43,24
4	40	38	26	5	21	44,74
5	50	37	31	7	24	35,14
<i>Network</i>		189	152	35	117	38,10

<i>N</i>	<i>Desynch.</i>	<i>Time desynch. (s)</i>	<i>Avg latency (ms)</i>	<i>Max latency (ms)</i>	<i>Min latency (ms)</i>
1	0	0	2625,50	6435	30
2	0	0	2564,63	7020	30
3	0	0	2570,00	7170	30
4	0	0	2033,89	6765	105
5	0	0	2580,94	8685	225
<i>Network</i>			2489,24	8685	30

Table 32: test 6, obtained data for Z1

The PLR has dropped under 40 %, which is a proof that decreasing the traffic at application layer is causing the network to get better results.

On the other hand, the latency seems to be stabilized at around 2500 ms, and doesn't seem to improve increasing the period.

4.9 Test 7

Increase of the period to 90 s, trying to reduce the PLR more.

The changes made are in code A5.

Number of retransmissions:	3
Amount of shared channels:	6
Period of transmission (s):	90
Payload (bytes):	127

Table 33: test 7, configuration

TelosB:

Memory impact:

ROM (bytes)	44212
RAM (bytes)	6904

Table 34: test 7, memory impact for TelosB

Data obtained from test 7 for TelosB:

<i>N</i>	<i>Mote</i>	<i>Pack. Sent</i>	<i>Pack. Rec.</i>	<i>Pack. Dupl.</i>	<i>Rec.-Dupl.</i>	<i>PLR (%)</i>
1	1d	11	15	4	11	0,00
2	a2	12	10	1	9	25,00
3	7	11	12	2	10	9,09
4	14	11	10	1	9	18,18
5	b7	12	12	1	11	8,33
<i>Network</i>		57	59	9	50	12,28

<i>N</i>	<i>Desynch.</i>	<i>Time desynch. (s)</i>	<i>Avg latency (ms)</i>	<i>Max latency (ms)</i>	<i>Min latency (ms)</i>
1	0	0	2138,57	6540	600
2	0	0	2133,33	4740	555
3	0	0	2216,25	6555	465
4	0	0	3201,67	6825	405
5	0	0	2091,25	5415	450
<i>Network</i>			2456,69	7755	405

Table 35: test 7, obtained data for TelosB

Z1:

Memory impact:

ROM (bytes)	37952
RAM (bytes)	6968

Table 36: test 7, memory impact for Z1

Data obtained from test 7 for Z1:

<i>N</i>	<i>Mote</i>	<i>Pack. Sent</i>	<i>Pack. Rec.</i>	<i>Pack. Dupl.</i>	<i>Rec.-Dupl.</i>	<i>PLR (%)</i>
1	10	12	14	3	11	8,33
2	20	10	13	3	10	0,00
3	30	14	18	4	14	0,00
4	40	10	10	1	9	10,00
5	50	12	11	2	9	25,00
<i>Network</i>		58	66	13	53	8,62

<i>N</i>	<i>Desynch.</i>	<i>Time desynch. (s)</i>	<i>Avg latency (ms)</i>	<i>Max latency (ms)</i>	<i>Min latency (ms)</i>
1	0	0	2010,00	5775	75
2	0	0	3536,54	10830	75
3	0	0	3018,53	10740	30
4	0	0	1976,25	3975	735
5	0	0	2422,50	5880	510
<i>Network</i>			2679,59	10830	30

Table 37: test 7, obtained data for Z1

With a period of 90 seconds, the PLR has reached acceptable values, whereas the latency hasn't changed and has kept stable around 2500 ms.

Chapter 5: Results

In this chapter the data from the previous chapter is analyzed, giving the results of the tests.

5.1 Desynchronizations

In tests number 1 and 3 some desynchronizations happened during the realization of the tests. These are the only tests with both, 4 second period and retransmissions enabled.

After lowering the traffic at application or MAC layer, desynchronizations have disappeared, which means that with a high amount of traffic, a lot of collisions happened, in a way that ADV messages cannot arrive the mote, resulting in a desynchronization.

It is also interesting to see how the desynchronizations create an increase of the PLR, this effect being this higher in the motes with longer desynchronization periods, as shown in Figure 1, Figure 2, Figure 3 and Figure 4.

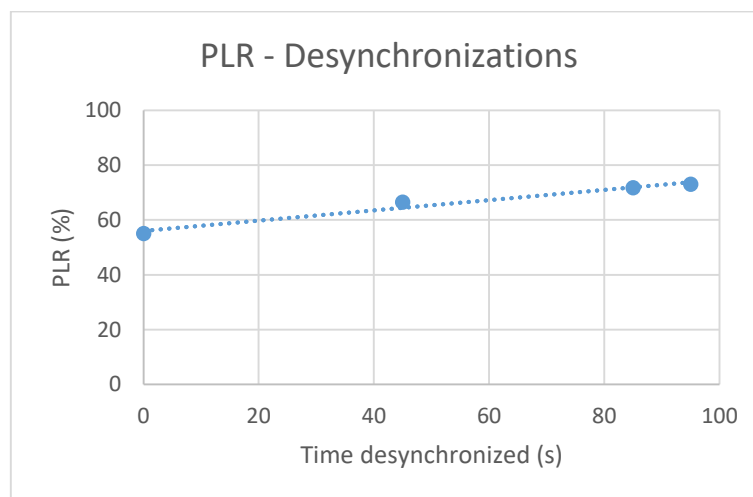


Figure 1 - Test 1: PLR-Desynchronizations with TelosB

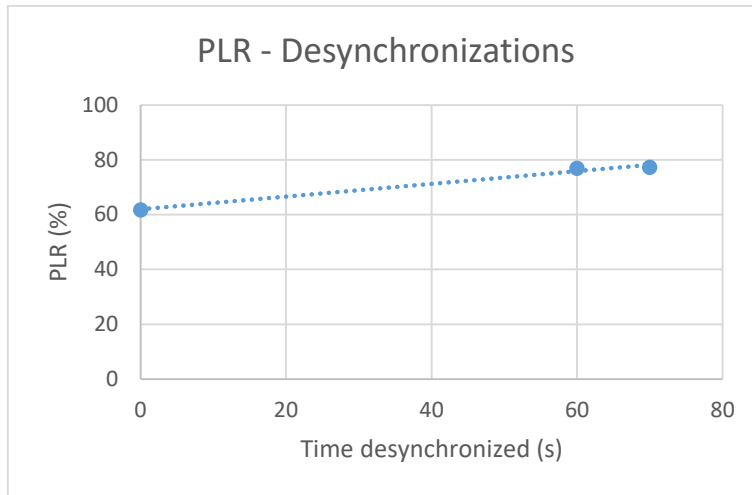


Figure 2 - Test 1: PLR-Desynchronizations with Zolertia Z1

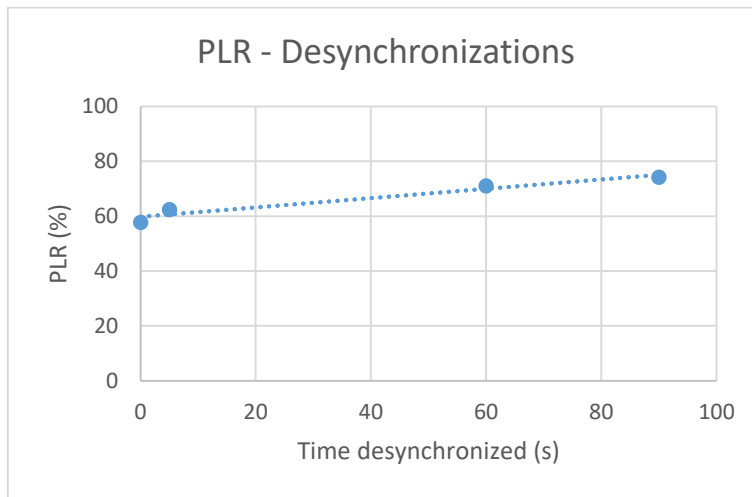


Figure 3 - Test 3: PLR-Desynchronizations with TelosB

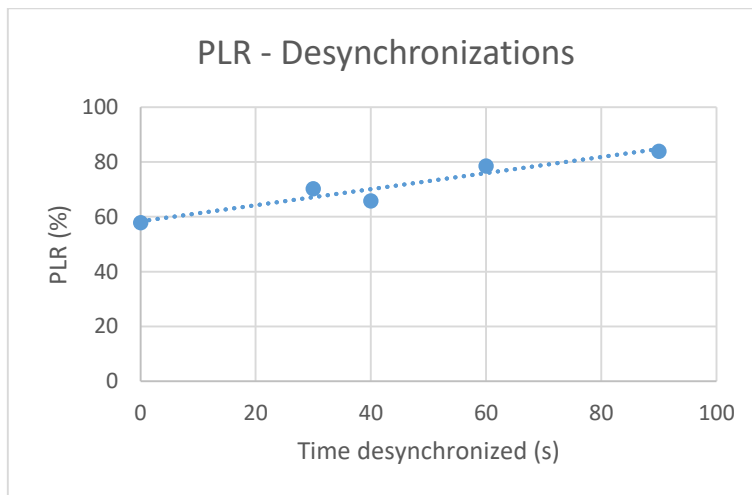


Figure 4 - Test 3: PLR-Desynchronizations with Zolertia Z1

Figure 1 and Figure 2 show the effect of the amount of time desynchronized in the PLR for both motes in test 1 and Figure 3 and Figure 4 for test 3.

The slopes in the linear function vary from 0,16 to 0,29, being this effect greater on the Z1 motes, in Table 38 it is shown a comparison of these slopes (Increase of PLR (%) per desynchronized time unit (s)).

	TelosB	Z1
Test 1	0,19	0,23
Test 3	0,16	0,29

Table 38 Increase of PLR for each second desynchronized

5.2 PLR

The PLR has got very bad results in most tests, with values over 50 % in 3 tests and over 20 % in 6 out of the 7 tests.

It is important to see that **both motes have had a very similar behavior** in all tests as Figure 5 shows.

The PLR has been specially affected in test 2, in which there weren't retransmissions, causing a lot of packets not reaching the DAGROOT.

In test 3 the results are similar for those of test 1 so it is possible to guess that the amount of available channels does not affect to the PLR in a significant way.

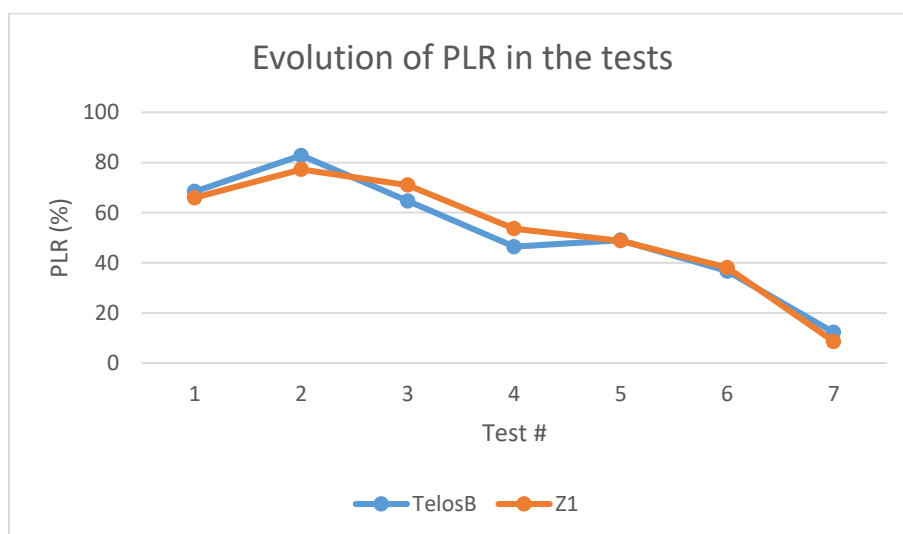


Figure 5 Evolution of PLR in the tests

The tests have shown that **increasing the period of the data transmission has improved the PLR** reaching to 12.28 % for TelosB and 8.62 % for Z1 with 90 s period. Figure 6 shows how the PLR has changed with different periods. For this graphic tests 1, 4, 6 and 7 have been taken into account. All of them have 6 channels, full payload and retries enabled and the only difference is the transmission period.

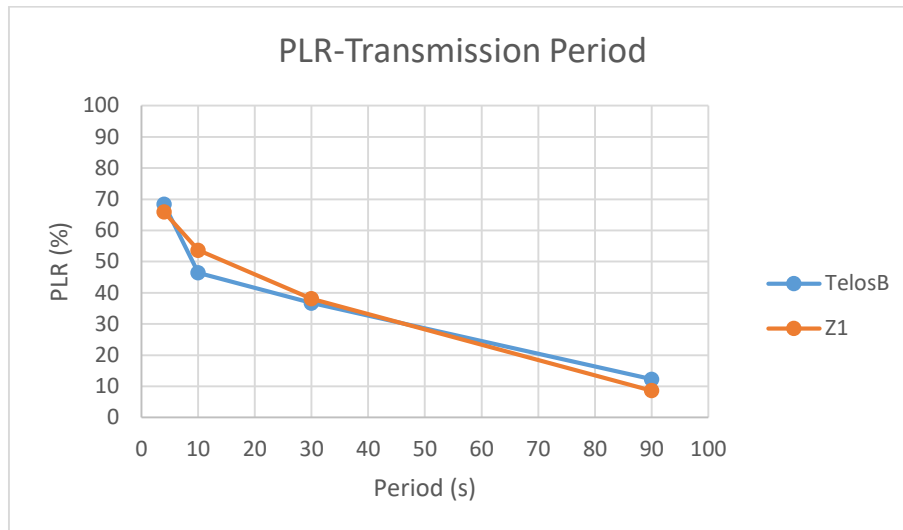


Figure 6 Evolution of PLR with the period

To finalize with the PLR, the Z1 has got the best results in the last test with a 3.66% of difference with respect to the TelosB but in other tests the TelosB has got better results, so we cannot say that one of them is clearly better than the other regarding this parameter.

5.3 Latency

Latencies have been quite high in all the tests. Figure 7 shows a comparison of the average latencies of both motes in all the tests.

Once again, both motes have had a very **similar behavior**, resulting in two similar curves in the graph.

In the first test, the average latency was close to 10 s, which is a very high value for this type of networks. This configuration had 3 retries and 6 channels the same as the last test but the 4 seconds period added to the desynchronizations, have given this bad results.

Taking a look at the window of 4 minutes without desynchronizations, the average latency drops down, making obvious that the **desynchronizations generate high**

latencies, due to the messages that wait to be retransmitted until the motes synchronize again.

In test 3, the available channels were duplicated with the aim of reducing latencies. The test was successful, getting much lower latencies than in the first test but the effect of the desynchronizations is still there, having higher latencies than in the rest of the tests with only 6 channels but no desynchronizations.

The best result for latencies are in the second test, in which there were **not retransmissions**. This is due to the fact that all the messages that arrived successfully were received in the first try but as the PLR has shown, most of the packets were lost.

In the rest of the tests with no desynchronizations and lower data traffic, the average latencies have been more or less **stable around 2.5 seconds**.

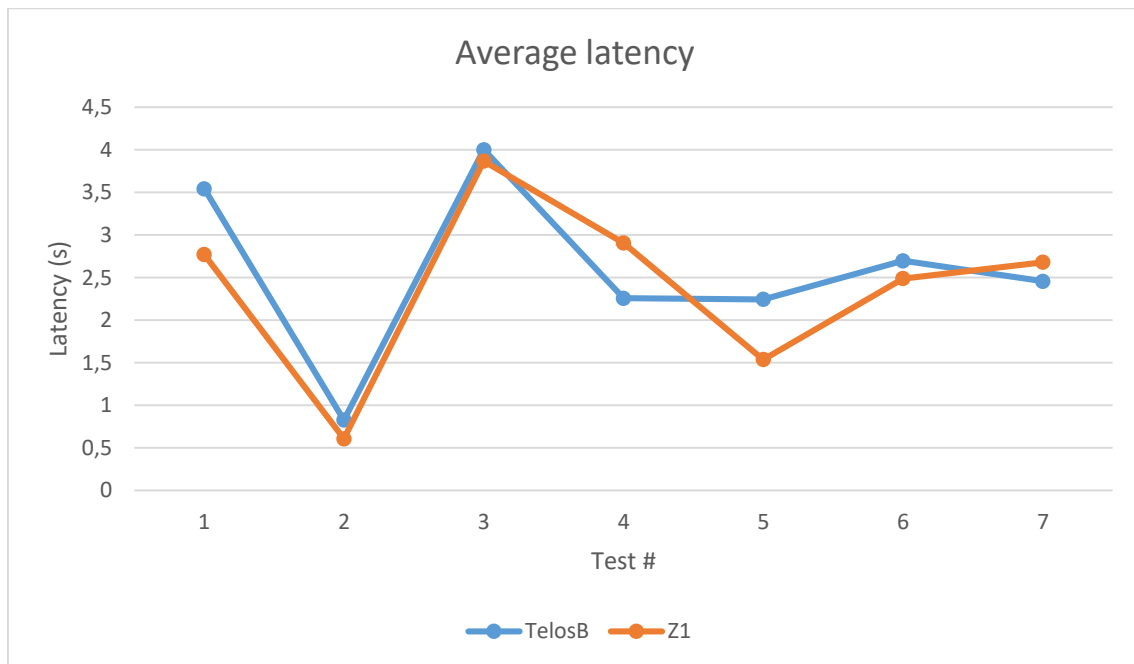


Figure 7 Average latency in the tests

As for the maximum latencies (Figure 8), the first and third tests have got the worse results showing the effects of the desynchronizations. The maximum latencies are much lower in the rest of the tests.

In test 4, the Z1 has got a very high maximum latency. Taking a look in the tables of the previous chapter, it is caused by a mote that got very high latencies, increasing the average and giving this unexpected value, probably caused by some error during the test, because it does not seem to repeat this behavior in other tests.

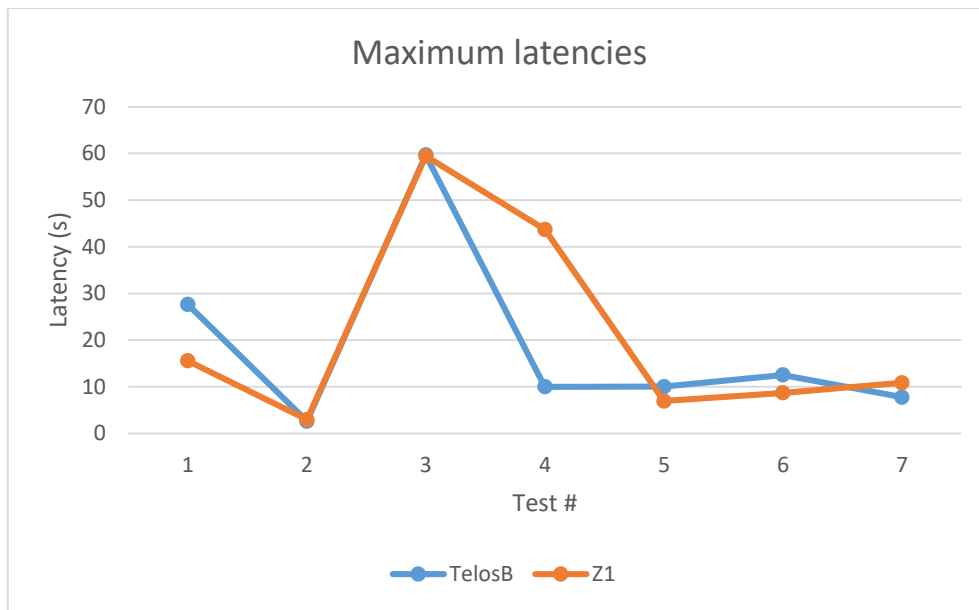


Figure 8 Maximum latency in the tests

Finally, as for the minimum latency, Figure 9 shows the minimum latency obtained in each test.

This results show that the Zolertia Z1 has been able to get the lower latencies, obtaining a minimum latency of 30 ms in most tests, whereas the TelosB has got minimum latencies over 200 ms. Here is the first result in which **the processor of the Z1 has shown its better specifications**, being able to get in all the tests some packets whit 30 ms latency. Even in the last tests, in which the total amount of packets received by each mote was low (around 10 packets) due to the high period of transmission, the Z1 was able to get this low value. The TelosB on the other hand, had an increase of the minimum latency showing that getting low latency values is rarer, so it experienced an increase in minimum latency.

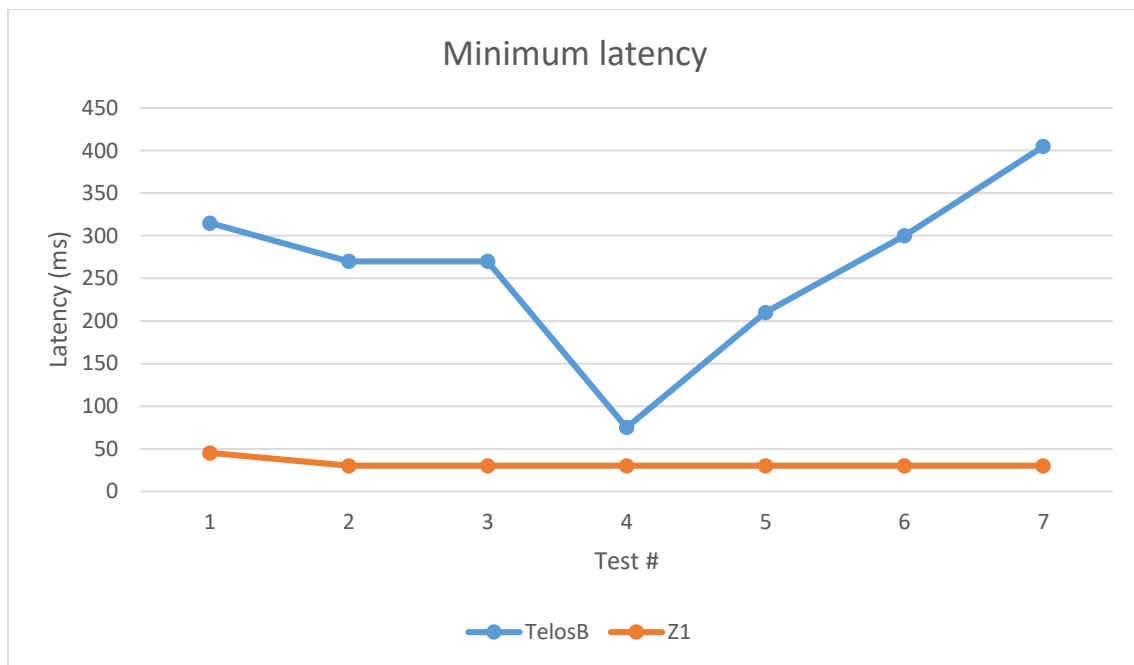


Figure 9 Minimum latencies in the tests

6. Conclusion

The objective of this thesis was to characterize Wireless Sensor Networks (WSNs) in terms of performances. For that purpose, **latencies** and **packet lost rate** numbers of two known motes (TelosB and Zolertia Z1 running OpenWSN) have been compared.

The results have shown how they work under different conditions (amount of channels, retransmissions, transmission period and payload length) and which conditions are needed to get admissible results.

Furthermore, all the tests have been done equally for both motes, which means that results are cross-comparable.

After looking at the results of the tests, we can conclude that the **PLR is strongly influenced by the amount of traffic in the network** for both platforms.

The only way to get an acceptable PLR (around 10%) has been with a 90 second period of data transmission, getting higher values for other values, reaching over 60 % of PLR with 4 seconds period, or 80 % without retransmissions.

On the other hand, the latencies are better with higher periods too, getting an average latencies above 8 seconds for periods of 4 seconds. Anyway, latency settles down faster than the PLR, getting an average latency around 2,5 seconds for periods higher than 10 seconds.

The better results for the latencies have been obtained without retransmissions, being under 1 second, but the increase of the PLR, doesn't allow us to use this configuration because of the high amount of packets lost.

This results make clear that this scenario **cannot be used for any real time application**, due to the high period required and the high latencies obtained, it is impossible to control anything that needs a fast response and high frequency control actions.

Anyway, it is possible to find some applications that do not require high speeds in which the variables under control cannot change fast in the time, such as temperature, pressure, humidity, etc.

If we compare both motes, we cannot say that there has been a huge difference between them. In some tests the TelosB has gotten better results for latency or PLR numbers whereas in others the Z1 has gotten better results, never having a significant difference.

In every test both motes have given the expected results, increasing or lowering the latencies and PLR values with similar tendencies, for example, both of them have lowered the PLR when we increased the period of transmission or both of them have lowered the latency and increased the PLR when there weren't retransmissions.

As for the final test, being the one with the most acceptable results, the Z1 has got an 8.62 % of PLR against the 12.28 % gotten by TelosB. Resulting in a difference of 3.66 %.

On the other side, the latencies have been better for the TelosB mote, with an average latency of 2456,69 ms in contrast with the 2679,59 ms of the Z1. Resulting on a 9,07 % faster results for the TelosB.

This results are surprising for the fact that the microcontroller of the Z1 has 16 MHz clock frequency that doubles the 8 MHz of the TelosB, expecting to obtain better results for the first one. **The superior clock speed has only been evident in the values for minimum latencies**, even if they have not made any difference in the average latency. This issue needs to be studied, a higher performance is expected from a superior, more expensive hardware, and *OpenWSN* has not been able to take advantage of this features.

It is also important to consider that the experiments have been done in a **noisy environment**, influenced by Wi-Fi networks that operate in the **same radio frequency** (2.4 GHz) and other motes and devices.

Currently, The Internet Of Things is a subject with a lot of repercussion and demand, that brings many opportunities for a close future. Nowadays, the exchange of information is a fundament in our society and making a wider and safer internet in which millions of devices can communicate, makes our life more comfortable, easier and safer, giving us the possibility to access many information and controlling remotely all the connected devices that are part of our environment.

This thesis gives a little start to the characterization of two motes running a new firmware under development but the work ahead from this point is full of possibilities that will need the help of many universities around the world and the research community.

To continue the characterization, there are many possibilities that would allow to know better this subject:

- tests with different amount of motes;
- different topologies;
- expansion to more devices: *Openmote* for example;
- expansion to other firmwares: *Contiki* or *Riot*;
- longer tests: to see how the motes work on long term working periods;
- addition of more sensor data: humidity;
- different configurations: to improve performance, trying to reduce latency, adapting for real-time applications, etc;
- tests in different environments;
- experiments on the communication security.

7. Bibliography

- [1] P. Kinney - IEEE 802.15 WPAN Task Group 4e (TG4e) - IEEE standard association - www.ieee802.org/15/pub/TG4e.html - 2003 - Consulted: 09/07/2016.
- [2] J. A. Gutierrez - IEEE std. 802.15.4 Enabling Pervasive Wireless Sensor Networks - EATON – 2005.
- [3] F. Zhao, L Guibas - Wireless Sensor Networks – 2004 - 1st edition.
- [4] Wireless & Mobile Network Lab - IEEE 802.15.4 MAC - <http://wireless.cs.tku.edu.tw/~kpsih/course/wireless96/IEEE%20802.15.4%20MAC%20Layer.pdf> - 2005 - Consulted: 03/08/2016.
- [5] G. Anastasi - From IEEE 802.15.4 to IEEE 802.15.4e Another Step towards the Internet of (Important) Things - Pervasive Computing & Networking Lab (PerLab) Dept. of Information Engineering, University of Pisa - 2014.
- [6] T. Watteyne, M. Palattella, L. Grieco - Using IEEE 802.15.4e Time-Slotted Channel Hopping (TSCH) in the Internet of Things (IoT): Problem Statement - Internet Engineering Task Force (IETF) - 2015.
- [7] T. Tsvetkov - RPL: IPv6 Routing Protocol for Low Power and Lossy Networks - Fakultät für informatik, Technische Universität München - 2011.
- [8] T. Winter - RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks - Internet Engineering Task Force (IETF) - 2012.
- [9] J. Postel - User Datagram Protocol - Internet Engineering Task Force (IETF) - 1980.
- [10] F. Chraim - OpenWSN: a Standards-Based Low-Power Wireless Development Environment - Transactions on Emerging Telecommunications Technologies - 2012.

[11] T. Watteyne - Stack - OpenWSN-atlassian - <https://openwsn.atlassian.net/wiki/display/OW/Stack> - 2014 - Consulted: 03/08/2016.

[12] T. Watteyne - Architecture - OpenWSN-atlassian - <https://openwsn.atlassian.net/wiki/display/OW/Architecture> - 2013 - Consulted: 03/08/2016.

[13] Memsic inc. - TelosB datasheet - TelosB mote platform - 2003 - 6020-0094-03 Rev A.

[14] Zolertia - Zolertia Z1 data sheet - 2010 - v1.1 Rev. C.

8. Appendix

8.1 Code A1: UDPlatency.c

```
#include "opendefs.h"
#include "udplatency.h"
#include "openudp.h"
#include "openqueue.h"
#include "openserial.h"
#include "packetfunctions.h"
#include "opentimers.h"
#include "openrandom.h"
#include "opencoap.h"
#include "scheduler.h"
#include "IEEE802154E.h"
#include "idmanager.h"
#include "neighbors.h"
#include "sixtop.h"
#include "scheduler.h"
#include "sht11.h"

//===== defines =====

#define UDPLATENCYPERIOD 10000
#define NUMPKTTEST 300

//===== variables =====

udplatency_vars_t udplatency_vars;
//===== prototypes =====

void udplatency_timer(void);
void udplatency_pushTimer(void);
void udplatency_PushTask(void);
void trigger_forward(void);

//===== public =====

void udplatency_init(void) {
    udplatency_vars.seqNum_my          = 0;
    udplatency_vars.seqNum_global      = 0;
    udplatency_vars.udplatency_security = 0;
    udplatency_vars.triggerReceived    = 0;

    udplatency_vars.UDPLATENCYPERIOD = 4000;
    udplatency_vars.NUMPKTTEST = 600;

    udplatency_vars.timerId = opentimers_start(udplatency_vars.UDPLATENCYPERIOD,
        TIMER_PERIODIC, TIME_MS,
        udplatency_timer);
}

void udplatency_task() {
    OpenQueueEntry_t* pkt;
    open_addr_t * p;
    open_addr_t q;
    unsigned int    avg          = 0;
```

```

// don't run if not synch
if (ieee154e_isSynch() == FALSE) return;

// don't run on dagroot
if (idmanager_getIsDAGroot()) {
    opentimers_stop(udplateny_vars.timerId);
    return;
}

// prepare packet
pkt = openqueue_getFreePacketBuffer(COMPONENT_UDPLATENCY);
if (pkt==NULL) {
//      openserial_printError(COMPONENT_UDPLATENCY,ERR_NO_FREE_PACKET_BUFFER,
//      (errorparameter_t)0,
//      (errorparameter_t)0);
//      increment seqNum, PLR stats on OV
    udplateny_vars.seqNum_my++;
    udplateny_vars.seqNum_global++;
    return;
}

pkt->creator      = COMPONENT_UDPLATENCY;
pkt->owner        = COMPONENT_UDPLATENCY;
pkt->l4_protocol  = IANA_UDP;
pkt->l4_sourcePortORicmpv6Type = WKP_UDP_LATENCY;
pkt->l4_destination_port = WKP_UDP_LATENCY;
pkt->l3_destinationAdd.type = ADDR_128B;
memcpy(&pkt->l3_destinationAdd.addr_128b[0],&ipAddr_motedata,16);

// the payload contains the 64bit address of the sender + the ASN

packetfunctions_reserveHeaderSize(pkt, sizeof(asn_t));
ieee154e_getAsn(pkt->payload); //gets asn from mac layer.

packetfunctions_reserveHeaderSize(pkt,8);
p=idmanager_getMyID(ADDR_64B);
pkt->payload[0] = p->addr_64b[0];
pkt->payload[1] = p->addr_64b[1];
pkt->payload[2] = p->addr_64b[2];
pkt->payload[3] = p->addr_64b[3];
pkt->payload[4] = p->addr_64b[4];
pkt->payload[5] = p->addr_64b[5];
pkt->payload[6] = p->addr_64b[6];
pkt->payload[7] = p->addr_64b[7];

packetfunctions_reserveHeaderSize(pkt,8);
avg =sht11_cal_temp();

pkt->payload[0] = avg;
pkt->payload[1] = 0;
pkt->payload[2] = 0;
pkt->payload[3] = 0;
pkt->payload[4] = 0;
pkt->payload[5] = 0;
pkt->payload[6] = 0;
pkt->payload[7] = 0;

// insert Sequence Number
packetfunctions_reserveHeaderSize(pkt,sizeof(udplateny_vars.seqNum_global));
pkt->payload[0] = (udplateny_vars.seqNum_global >> 8) & 0xff;
pkt->payload[1] = udplateny_vars.seqNum_global & 0xff;

```

```

pkt->FIFO_seqNum = udplacency_vars.seqNum_my;

openserial_printInfo(COMPONENT_UDPLATENCY,155,
                    (errorparameter_t)pkt->FIFO_seqNum,
                    (errorparameter_t)100);

//17 bytes payload
uint8_t i;
for (i = 0; i < 16; i++){
    packetfunctions_reserveHeaderSize(pkt,1);
    pkt->payload[0] = i;
}

// send packet
if ((openudp_send(pkt)) == E_FAIL) {
    openqueue_freePacketBuffer(pkt);
}

// increment seqNum
udplacency_vars.seqNum_my++;
udplacency_vars.seqNum_global++;

// close timer when test finish
if (udplacency_vars.seqNum_my > udplacency_vars.NUMPKTTEST) {
    udplacency_vars.triggerReceived = FALSE;
    udplacency_vars.seqNum_my = 0;
    udplacency_vars.seqNum_global = 0;
    opentimers_stop(udplacency_vars.timerId);
}
}

void udplacency_timer(void) {
    scheduler_push_task(udplacency_task,TASKPRIO_COAP);
}

void udplacency_pushTimer(void){
    scheduler_push_task(udplacency_PushTask,TASKPRIO_SIXTOP);
}

void udplacency_forwardTimer(void){
    scheduler_push_task(trigger_forward,TASKPRIO_SIXTOP);
}

void udplacency_PushTask(void){
    udplacency_vars.timerId = opentimers_start(udplacency_vars.UDPLATENCYPERIOD,
        TIMER_PERIODIC,TIME_MS,
        udplacency_timer);
}

void udplacency_sendDone(OpenQueueEntry_t* msg, oerror_t error) {
    msg->owner = COMPONENT_UDPLATENCY;
    if (msg->creator!=COMPONENT_UDPLATENCY) {
        //openserial_printError(COMPONENT_UDPLATENCY,ERR_UNEXPECTED_SENDDONE,
        //    (errorparameter_t)0,
        //    (errorparameter_t)0);
    }
}

```

```

    openqueue_freePacketBuffer(msg);
}

void udplacency_receive(OpenQueueEntry_t* msg) {
    openqueue_freePacketBuffer(msg);
}

void trigger_receive(OpenQueueEntry_t* msg){

    //TODO
    uint16_t receivedRate;
    uint16_t numberOfPackets;
    uint8_t securityFlag;
    uint16_t timeToWaitReceived;
    if(idmanager_getIsDAGroot()){
        udplacency_vars.triggerReceived = TRUE;
    }

    //if I have received the desync message previously, simply discard it.
    if (udplacency_vars.triggerReceived == TRUE){
        //free the RAM
        openqueue_freePacketBuffer(msg);

        return;
    } else {
        udplacency_vars.triggerReceived = TRUE;
    }
}

openserial_printError(COMPONENT_UDPLATENCY,ERR_INVALIDSERIALFRAME,
    (errorparameter_t)0,
    (errorparameter_t)500);

//toss the protocol header
packetfunctions_tossHeader(msg,1);

//retrieve values
//get the rate
receivedRate = msg->payload[0] + 256 * msg->payload[1];
packetfunctions_tossHeader(msg,2);

//get the number of packets to generate
numberOfPackets = msg->payload[0] + 256 * msg->payload[1];
packetfunctions_tossHeader(msg,2);

//get the security flag
securityFlag = msg->payload[0];
packetfunctions_tossHeader(msg,1);

//get the time to wait
timeToWaitReceived = msg->payload[0] + 256 * msg->payload[1];
packetfunctions_tossHeader(msg,2);

//free up the RAM
openqueue_freePacketBuffer(msg);

//save variables
udplacency_vars.UDPLATENCYPERIOD = receivedRate;
udplacency_vars.NUMPKTTEST = numberOfPackets;
udplacency_vars.udplacency_security = securityFlag;
udplacency_vars.timeToWaitReceived = timeToWaitReceived;

//schedule the timer for the start of the UDPLatency task
udplacency_vars.globaltimerId = opentimers_start(timeToWaitReceived,

```

```

        TIMER_ONESHOT,TIME_MS,

        udplatency_pushTimer);
}

uint16_t udplatency_getSeqNum(void){
    uint16_t value;
    value = udplatency_vars.seqNum_global;
    udplatency_vars.seqNum_global++;
    return value;
}
uint8_t udplatency_getTimerId(void){
    return udplatency_vars.timerId;
}

void udplatency_setSecurity(bool value){
    udplatency_vars.udplatency_security = value;
}

bool udplatency_getSecurity(void){
    return udplatency_vars.udplatency_security;
}

void udplatency_setPeriod(uint16_t value){
    udplatency_vars.UDPLATENCYPERIOD = value;
}

//Forward the trigger message down in the tree

void trigger_forward(void){

    OpenQueueEntry_t* pkt;

    //generate a broadcast MAC message with received parameters
    pkt = openqueue_getFreePacketBuffer(COMPONENT_OPENBRIDGE);
    if (pkt==NULL) {
        return;
    }

    openserial_printError(COMPONENT_UDPLATENCY,ERR_INVALIDSERIALFRAME,
        (errorparameter_t)0,
        (errorparameter_t)501);

    //admin
    pkt->creator = COMPONENT_SIXTOP;
    pkt->owner = COMPONENT_UDPLATENCY;

    // some I2 information about this packet
    pkt->I2_frameType = IEEE154_TYPE_DATA;
    pkt->I2_nextORpreviousHop.type = ADDR_16B;
    pkt->I2_nextORpreviousHop.addr_16b[0] = 0xff;
    pkt->I2_nextORpreviousHop.addr_16b[1] = 0xff;
    pkt->isBroadcastIE = TRUE;
    // pkt->FIFO_sn = 0; //maximum priority

    //payload
    //amount of time we have to wait for the start of sending packets
    packetfunctions_reserveHeaderSize(pkt, sizeof(uint16_t));
    pkt->payload[0] = (uint8_t) udplatency_vars.timeToWaitReceived;
    pkt->payload[1] = (uint8_t) (udplatency_vars.timeToWaitReceived >> 8);
}

```

```

//security flag
packetfunctions_reserveHeaderSize(pkt, sizeof(uint8_t));
pkt->payload[0] = udplatency_vars.udplatency_security; //list-termination

//number of packets
packetfunctions_reserveHeaderSize(pkt, sizeof(uint16_t));
pkt->payload[0] = (uint8_t) udplatency_vars.NUMPKTTEST;
pkt->payload[1] = (uint8_t) (udplatency_vars.NUMPKTTEST >> 8);

//rate
packetfunctions_reserveHeaderSize(pkt, sizeof(uint16_t));
pkt->payload[0] = (uint8_t) udplatency_vars.UDPLATENCYPERIOD;
pkt->payload[1] = (uint8_t) (udplatency_vars.UDPLATENCYPERIOD >> 8);

//add id for the protocol
packetfunctions_reserveHeaderSize(pkt, sizeof(uint8_t));
pkt->payload[0] = 0xAB;

// put in queue for MAC to handle
sixtop_send(pkt);

return;
}

```

8.2 Code A2: UDPlatency.c

```

packetfunctions_reserveHeaderSize(pkt,8);
avg =tmp102_read_temp_simple();

pkt->payload[0] = avg;
pkt->payload[1] = 0;
pkt->payload[2] = 0;
pkt->payload[3] = 0;
pkt->payload[4] = 0;
pkt->payload[5] = 0;
pkt->payload[6] = 0;
pkt->payload[7] = 0;

```

8.3 Code A3: UDPlatency.c

```

udplatency_vars.UDPLATENCYPERIOD = 10000;
udplatency_vars.NUMPKTTEST = 600;

```

8.4 Code A4: UDPlatency.c

```

udplatency_vars.UDPLATENCYPERIOD = 30000;
udplatency_vars.NUMPKTTEST = 600;

```

8.5 Code A5: UDPlatency.c

```

udplatency_vars.UDPLATENCYPERIOD = 90000;
udplatency_vars.NUMPKTTEST = 600;

```

8.6 Code B1: IEEE 802154E.h

```
#define SYNCHRONIZING_CHANNEL    23 // channel the mote listens on to synchronize
#define TXRETRIES                3 // number of MAC retries before declaring failed
#define TX_POWER                 31 // 1=-25dBm, 31=0dBm (max value)
#define RESYNCHRONIZATIONGUARD  5 // in 32kHz ticks. min distance to the end of the slot to successfully
                                //synchronize
#define US_PER_TICK              30 // number of us per 32kHz clock tick
#define ADVTIMEOUT               5 // in seconds: sending ADV every 30 seconds
#define MAXKAPERIOD              500 // in slots: @15ms per slot -> ~30 seconds. Max value used by adaptive
synchronization.
#define DESYNCTIMEOUT            4000 //in slots: @15ms per slot -> ~35 seconds. A larger DESYNCTIMEOUT is
//needed if using a larger KATIMEOUT.
#define LIMITLARGETIMECORRECTION 5 // threshold number of ticks to declare a timeCorrection "large"
#define LENGTH_IEEE154_MAX       128 // max length of a valid radio packet
#define DUTY_CYCLE_WINDOW_LIMIT (0xFFFFFFFF>>1) // limit of the dutycycle window
```

8.7 Code B2: IEEE 802154E.h

```
#define SYNCHRONIZING_CHANNEL    23 // channel the mote listens on to synchronize
#define TXRETRIES                0 // number of MAC retries before declaring failed
#define TX_POWER                 31 // 1=-25dBm, 31=0dBm (max value)
#define RESYNCHRONIZATIONGUARD  5 // in 32kHz ticks. min distance to the end of the slot to successfully
                                //synchronize
#define US_PER_TICK              30 // number of us per 32kHz clock tick
#define ADVTIMEOUT               5 // in seconds: sending ADV every 30 seconds
#define MAXKAPERIOD              500 // in slots: @15ms per slot -> ~30 seconds. Max value used by adaptive
synchronization.
#define DESYNCTIMEOUT            4000 //in slots: @15ms per slot -> ~35 seconds. A larger DESYNCTIMEOUT is
//needed if using a larger KATIMEOUT.
#define LIMITLARGETIMECORRECTION 5 // threshold number of ticks to declare a timeCorrection "large"
#define LENGTH_IEEE154_MAX       128 // max length of a valid radio packet
#define DUTY_CYCLE_WINDOW_LIMIT (0xFFFFFFFF>>1) // limit of the dutycycle window
```

8.8 Code C1: schedule.h

```
#define SUPERFRAME_LENGTH        101 //should be 101

#define NUMADVSLOTS              1
#define NUMSHAREDTXRX            6
//#define NUMDEDICATEDTX         14
//#define NUMDEDICATEDRX         14
#define NUMSERIALRX              1
```

8.9 Code C2: schedule.h

```
#define SUPERFRAME_LENGTH        101 //should be 101

#define NUMADVSLOTS              1
#define NUMSHAREDTXRX            12
//#define NUMDEDICATEDTX         14
//#define NUMDEDICATEDRX         14
#define NUMSERIALRX              1
```


8.10 Stack organization diagram of the OpenWSN protocol stack

