Hazard Detection With FreeRTOS based Wireless Sensor Networks

Diogo Filipe Tomás Guerra
Instituto Superior Técnico, Av. Professor Doutor Aníbal Cavaco Silva, 2744-016 Porto Salvo, Portugal
Phone: +351-914448691, e-mail: diogoguerra@tecnico.ulisboa.pt

Abstract – In this work are studied various operating systems and communication protocol stacks currently used in wireless sensor networks. The aim is to find a solution for timely delivery of information using a real-time operating system and a redundant and resilient protocol stack like the proposed IEEE 802.15.4E. Moreover, it addresses the adaptations made on the firmware to enable compatibility with the in-house already conceived platform – MoteIST. In the end the full project can compile and run on the MoteIST platform maintaining compatibility with the original project. Tests were made to assert proper communication with the work adapted.

Index terms – Timely Delivery, FreeRTOS, WSN, 802.15.4E, Hazard Detection and Emergency Response, OpenWSN.

I. INTRODUCTION

Wireless Sensor Networks are systems made up of spatial distributed nodes that measure physical quantities. This networks are suited for remote deployment where they communicate with the world via one or more base stations. This gateway relay information through a satellite constellation or other type of special communications.

Hazard Detection and Emergency Response is an application of Wireless Sensor Networks. Effective communication mechanisms are needed to manage networks with large size and the usage of Real-Time Operating Systems helps to ensure timely event response minimizing accidents and loss of life.

FreeRTOS is one of the most used Operating Systems with Real-Time characteristics ported to a wide range of microcontroller architectures. Although there are communication stacks available in FreeRTOS environment, there are none natively developed to support wireless sensor networks.

I decided to choose a recent protocol stack IEEE 802.15.4E currently under development with the project OpenWSN [1]. The OpenWSN project aims to provide a complete and free implementation of a protocol stack with a wide range of firmware and hardware platforms. The MAC implementation used is the TSCH - Time Scheduled Channel Hopping, communication schema.

The intention is to equip the FreeRTOS communications suite with a scalable, redundant and resilient protocol stack that enables timely delivery of event notifications. After this it is intended to adapt all this changes to the in-house already conceived platform – MoteIST [2] and making tests ensuring the protocol behaves as expected managing a considerable data throughput.

II. SENSOR NETWORKS

Sensor Networks (SN) have become popular in solutions for monitoring and environmental control in agriculture, industry, home automation and energy management applications [3],[4]. More recently Wireless Sensor Networks (WSN) are used in remote location monitoring environmental factors for fire prevention [5] or earthquake warning systems [6].

Nodes in a WSN monitoring fire hazards are subject to random failures due to battery depletion, as well as the antennas being reoriented in the wrong direction due to falling branches, curious animals or wind. As WSN transmit packets from node to node, the failure of several nodes relatively close to each other can partition the network into two subnets without communication [7]. This happens because one or more key nodes in the link between the subnets stops working. Also, if the node responsible for detecting the emergency event in a particular area fails to respond due to sensor element failure or node battery exhaustion a WSN failure occurs.

Taking the above into consideration one can easily say that the key factor in WSN is energy efficiency. In this hazardous applications, installation is typically carried out in a remote location, often difficult to access. For this reason, ensuring the lifetime of the network is important. This way, the cost of installing and maintaining this type of infrastructure can be reduced, making the use of WSN a more viable solution than other options.

A. Case Studies

In [8], 9 battery-powered motes are used to detect methane leaks in a factory. The selected network topology was the star topology where 1 of the nodes was configured as the base station.

Communication in the network is performed through 802.15.4[9] using ZigBee1 modules. The base station acts as a gateway with an Ethernet communication so that in case of emergency a team is notified. The lifetime of the network is greater than one year and it is mentioned that the main factor limiting it is the energy consumption of the methane sensor.

In [5] a network for the detection and monitoring of forest fires was presented. Nodes are powered by two AA batteries and controlled by an ATMega1281 microcontroller. The radio transceiver is a ZigBee
module. The network is organized in a mesh topology and a base station is used to schedule "sleep" periods. This topology was chosen because it allows network scalability. Using a duty cycle of 0.33% ensures that the lifetime of the network is greater than one year.

B. Communication Technologies in WSN

Motes have limited communication range. This factor is due not only to an inherent limitation of transmission power, in order to reduce interference in the transmission medium and to increase the lifetime of the node [10], but also thanks to power losses due to the propagation path of the electromagnetic waves and obstacles between nodes. For this reason the transmission of messages must be carried out using node-to-node routing in which the message is forwarded by intermediate nodes to the destination. These networks are referred to the mesh networks where packets can be transmitted with different schema's [11].

Network lifetime: The network is organized to be functional as long as possible. It is common that the limiting factor of the network lifetime is the energy accessible to the nodes and so routing of the packets is typically carried out so that there is the lowest energy consumption possible.

Fault Tolerance: In packet transmission the route that guarantees delivery with minimum transmission errors is chosen. Useful when communicating with mobile nodes, for example.

Response time: Packets are routed through the fastest transmission path, between the source node and the gateway. Typically this is the solution that requires less retransmissions.

The use of efficient communication protocols is an integral and fundamental part of these networks. In this way the Medium Access Control (MAC) is essential in WSN. MAC protocols must have good energy efficiency since the nodes have limited energy [4].

The MAC is responsible for the possibility of multiple nodes communicating using a shared medium, as is the case of WSN, in which the device used to transmit in the medium is the radio. Access to the medium in WSN is typically distributed, in that this access is easily implemented in networks with many nodes and therefore more easily scalable. Basic ideas of distributed access perform random access to the medium. However, it is also possible to perform scheduled access communication. In this type of access the typical considerations are below.

TDMA (Time Division Multiple Access): The transmission cycle is divided into a fixed number of slots. To each slot only a single node is assigned that can perform communication within that time without interference. The communication is coordinated by a node that signals a new transmission cycle (through a special beacon slot) allowing synchronization of the nodes in the network. This method has the advantage of ensuring throughput, fixed transmission delay and better energy efficiency. However, it suffers from limiting the response time (communication with this node is only resumed in the next scheduled slot).

TSCH (Time Scheduled Channel Hopping): This is a scheduling type that combines two different types of media access, namely Frequency Division Medium Access (similar to TDMA where slots are replaced with specific frequencies) and TDMA. In this type of scheduled transmission the transmission time continues to be divided into slots however to this is also assigned a unique transmission frequency. This frequency is typically signaled by the Hopping Sequence (HS). This method makes the possible total network throughput greater than using only one of the previous methods. It also allows relatively close nodes to communicate at the same time without interference between them, unlike TDMA.

C. Gateway Technologies

Different technologies can be used to enable communication from WSN locations with the internet. Opportunistic networks use node mobility as an advantage for packet transfer within a network. WSN's installed near a railway line can periodically exchange data with a mobile node installed in a passing train. A few seconds is the time it takes for the gateway to exchange packets stored during the period between trains with the mobile node and vice versa. This concept can also be applied with random animal movement in which routing algorithms are used according to the desired quality of service (QoS) [12].

Figure 1: Example of a network where mobile nodes carry out packet transport [12].

By means of satellite-enabled gateways message transmission is a viable option for any remote location, provided that coverage is guaranteed by the operator's satellite constellation. This service has the advantage that priority messages can be transmitted immediately with a maximum delay of a few minutes.

Figure 2: Example of a typical application of message communication in remote areas using satellites as relays.¹

Table 1: Comparison made to different popular multitasking cores to study the real-time characteristics, existence of native communication protocols and modularity for multiple and different nodes within a heterogeneous sensor network.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Contiki[15][2]</td>
<td>○</td>
<td>✓</td>
<td>&lt;30k (1)</td>
<td>13</td>
<td>✓</td>
<td>○</td>
<td></td>
</tr>
<tr>
<td>FreeRTOS[3]</td>
<td>✓</td>
<td></td>
<td>4k – 9k</td>
<td>84</td>
<td>34</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NANO-RK[16][4]</td>
<td>✓</td>
<td>X</td>
<td>&lt;18k (1)</td>
<td>12 (2)</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>RIOTOS[14][5]</td>
<td>✓</td>
<td>X</td>
<td>~5k</td>
<td>~50</td>
<td>6</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>TINYOS[15][17][6]</td>
<td>X</td>
<td>✓</td>
<td>&lt;4k</td>
<td>51</td>
<td>6</td>
<td>✓</td>
<td>○</td>
</tr>
</tbody>
</table>

Legend: ✓ - Meet the requirements; ○ – Partially meet requirements; X - Does not meet requirements;
1. Used memory includes communication protocols
2. Context switch: 45 µs on a 4 MHz processor.

D. Real-Time Operating Systems

In emergency detection and response scenarios, in addition to the need for protocols that perform timely communication, it is necessary that the OS used support real-time execution. Some OS's were evaluated and are presented in Table 1.

E. Other Communication Schemas

IEEE 802.15.4 MAC is possibly the most used protocol in WSN. The characteristics of the 802.15.4 MAC protocol are: synchronized networks access to the medium, management of guaranteed slots, validation of frames, confirmation of delivery of frames, association and disassociation to networks. The protocol can use several types of media access, depending on the network configuration used [9]:

IEEE 802.15.4E MAC is based on the IEEE 802.15.4 protocol and therefore shares many similarities. The key aspect of this new definition is the use of channel hopping (CH) that significantly increases the network's robustness against external interference and multipath fading.

In addition, unlike 802.15.4, guaranteed slot management is controlled by the hierarchically higher layers and in their design the contents of the synchronization packet is different. All other features defined by 802.15.4 remain similar.

Like its predecessor, the 802.15.4E protocol defines various types of medium access depending on the type of network to be used. In this case the focus will be on access using TSCH.

In this type of access time is defined in slots. The Absolute Slot Number (ASN) is incremented each slot and shared by all nodes in the same network. Assuming there is no propagation delay for the flag package, all nodes on the network have the same ASN. At the beginning of a slot the channel to be used is calculated using [18].

\[
CH = \text{macHoppingSequenceList} \left[ \text{macASN} + \text{channelOffset} \right] \mod \text{macHoppingSequenceLength}
\]

The channel offset is defined by the length of the list that defines the Hopping Sequence (HS). All nodes that wish to join the network expect the flag packet on a well-known and channel defined for the entire network. The flag package always contains the ASN. Figure 3 exemplifies a communications schedule for this network.

III. METHODOLOGY

Synchronized networks allow better control and knowledge about the energy consumption of a network, especially when these are made up of some nodes and the use of this network is considerable. In the examples studied [5], [8] the use of an 802.15.4 MAC protocol offers some guarantees regarding network lifetime.

2 http://www.contiki-os.org/ - December 2014
3 http://www.freertos.org/ - December 2014
5 http://riot-os.org/ – December 2014
7 https://openwsn.atlassian.net/wiki/display/OW/uRES – October 2016
Using a real-time OS to detect asynchronous and unpredictable signals that provides priority response over other tasks that are performing is an advantage. In this situation and through consultation of Table 1, FreeRTOS, Nano-RK and RiotOS are available.

The fact that 802.15.4E allows WSN to communicate in parallel in multiple channels in a frequency band allows to increase robustness to interferences in the scenario where it is used, be these interferences due to other networks or due to natural phenomena such as multipath fading.

Although OpenWSN is based on a simple OS where task is executed till completion, it include an adaptation of FreeRTOS as a task scheduler. This makes this project the most attractive option to be implemented in the MoteIST platform and therefore the one that was decided to use. To test the timely delivery hypothesis it is used:

A network topology unfavorable to the WSN under tests to evaluate the delay caused by forwarding packets between nodes and the impact of packet retransmission on the success rate of communications.

Background traffic in order to saturate the network with less priority packet traffic. This test is performed to evidence a real network where periodic data are collected in the network.

IV. ARCHITECTURE

OpenWSN is a project that provides solutions for easy development of WSN. There are three main groups in this project but implementation focuses on the firmware.

The firmware part of the OpenWSN project is developed to be transparent to the firmware, the build platform used in its development, the mote where the application is executed and it is also prepared to support different Operating Systems.

A. OpenOS

The OpenOS scheduler performs tasks with higher priority executed first. However, there is only a context switch when the current task finishes.

When a task is scheduled, the algorithm first looks for a free task buffer where it can store the information needed to execute it in the future. In case the task buffer is full the execution terminates in its entirety with an error signal and subsequent reset of the platform. If it is possible to allocate a new buffer, the algorithm searches the queue of the scheduled tasks for the position where the priority of the new task is less than the priority of the task currently under test and inserts it there.

B. OpenWSN and FreeRTOS

There is a structure with a global mutex, three processes with different priorities and 1 semaphore associated with each of these processes. Each one of these processes implement the same priority task buffer used in OpenOS. The difference is that there are three execution priorities (FreeRTOS processes) that generate sub-sets of tasks (OpenOS tasks).

As with OpenOS, tasks are performed in order of priority. The main difference is that an interrupt or instruction when scheduling a task (managed by a higher priority process) that overrides the priority of the currently running process immediately takes the processing time leaving the previous process (task) on hold. If the priority of the task to be scheduled is equal to or less than the current task, it is added to the task queue as before.

When a task finishes a new task is fetched to perform according to:

1. The priority of execution of processes (FreeRTOS).
2. The priority of the task within the sub-group managed by this process (OpenOS).

The three defined processes are:

- `xAppHandle` - is the lowest priority process responsible for managing all the user level tasks.
- `xSendDoneHandle` - Is responsible for managing tasks that handle packet errors rescheduling a new send or triggering an error notification. In addition, timer and alarm priorities fall into this category.
- `xRxHandle` - Performing at highest priority is dedicated to packet-receiving jobs scheduled by the protocol stack machine (802.15.4e scheduler). These tasks perform the interpretation and handling of packets that target increasingly higher layers of the protocol stack.

V. DEVELOPMENT

Firstly will be the hardware adaptation to OpenWSN's well-defined default function calls. The platform that we are going to use consists of a main board that integrates the TI microcontroller MSP430F5438A and the sub-modules of the platform that are the radio with a TI CC2420 and the UART-USB converter with a TI TUSB3410. An example of the constituents of the disassembled MoteIST platform can be visible below in Figure 4.

![Figure 4: The components that make up the MoteIST platform. Main PCB (TOP), Radio PCB (LEFT) and SERIAL (RIGHT)](image)
A. Adaptation of the MoteIST platform

One of the components used in MoteIST that is already developed in the stack used by OpenWSN is the software library for the radio, assuming that it is already operational.

In the case of the microcontroller there are projects in the OpenWSN with similar family as the microcontroller used in MoteIST which is the F5 series and family. This is the case of motes identified in the bsp folder as wsn430v13b, wsn430v14 and telosb that use the MSP430F1611 microcontroller.

First a folder for the platform is created at bsp-> boards-> MoteISTv5. This folder hosts all the files that perform the hardware abstraction of the upper layers.

After this the abstraction files are adapted to MoteIST according to their logical hierarchy. These files are adapted one by one and at each step is tested the correct operation of these through projects, common among the various motes, available for this purpose.

The compilation files used by SCons, the build tool used by OpenWSN, are now defined in preparation for the next steps. The adapted files are:

- **board_info.h** – Macros, interrupts and other definitions.
- **board.c** – Peripherals configuration and platform initialization.
- **leds.c and debugpins.c**
- **spi.c** – Interface with the radio.
- **uart.c** – Interface with the computer for debug and gateway packets provider.
- **bsp_timer.c** – Generic timer.
- **radiotimer.c** – Communications state machine timer.
- **eui64.c** – Mote unique identification (for network address purpose)

Having the integration successfully completed the project is tested for the use of OpenOS and finally integration with the protocol stack and some demo applications.

The first problem arise. There is an error linking the project because there is not enough memory! The error displayed by the console can be seen below.

```
/home/msp430-toolchain/v3_04_05_00/lib/gcc/msp430-elf/4.9.1/../../../../msp430-elf/bin/ld: region `ROM' overflowed by 16276 bytes
[...]
collectors: error: ld returned 1 exit status
scons: *** [building] Error
scons: building terminated because of errors.
Figure 5: Error displayed when Compiling the OpenWSN project.
```

This lack of space can be justified by the occupied ROM by the protocol stack and the applications present in the project. Although we can configure for the stack to only serve one of the transport protocols available and to disable compilation of the applications, the future introduction of FreeRTOS is expected to make available space for future applications minimal or nonexistent. As so, to use the full potential of the microcontroller configurations are made to enable extended compilation in order to run bigger programs.

B. Compilation in extended mode

The MSP430 series 5 (microcontroller of the MoteIST) has two zones of ROM memory. Namely the microcontroller (msp430f5438a) has 256 kB of available ROM.

It turns out that ROM is split between LOW ROM and HIGH ROM that is used depending on the build mode used. So far the compiler was just using LOW ROM because in compiling with mspgcc (compiler used) it compiles to the small memory model (small). The memory model for which the code is compiled varies the type of addressing performed by the microcontroller and to use HIGH ROM it is necessary to compile with an extended memory model.

Generally, the small memory model uses 16-bit addressing, whereas the large memory model allows for addresses up to 20 bits (extended addressing). The use of extended addressing brings the obvious advantage in the maximum possible size for our application (a maximum of 20 bits can address 1 MB). However, there is the disadvantage that addressing operations usually take twice as many clock cycles.

This extended addressing is only possible because the ALU\(^7\) and the microcontroller registers are purposely extended to 20 bits so that the necessary operations can be performed.

```
Figure 6: Block diagram of the MSP430X CPU\(^8\) architecture. We can verify that in the CPU the Data BUS is of 16 bit, however it allows the addressing of up to 20 bits (Address BUS ) [19].
```

7 ALU – Arithmetic Logic Unit.
8 CPU – Central Processing Unit
C. MSPGCC legacy update for MSP430-GCC

Due to a problem when linking the code, the compiler in use until now can’t be used for this extended compilation. Because the interrupt routines are compiled last, the normal compilation of code occupied LOW ROM first and then HIGH ROM. When trying to save the ISR, which only execute specifically on LOW ROM due to the microcontroller architecture, this memory zone was already fully occupied and a linker error occurred.

Through TI’s website we download the latest version of msp430-gcc for the virtual machine. After the transfer, the installation is performed. This tool has been configured in the path /opt/msp430-toolchain/ and new configuration is made to Scons toolchain to use the newly installed compiler.

D. Mote Resident Monitor (MRMv3)

The Mote Resident Monitor, the bootloader used, was updated to allow loading of the code for the platform since the use of a new compiler interfered with some preconditions considered by the old versions.

The main difference of the new MRM for its predecessors is the decoding of the files, which are sent in IHEX [20] format, to binary on the platform itself. This allows the code to be loaded from any computer through a small serial terminal with data flow control instead of a specific program for this task as before.

This modification to code loading simplifies the needs of the programs to be used, however, it does not take into account possible errors that may occur when loading code into the MoteIST platform.

The status diagram in Figure 7 represents the MRM program that runs on the platform.

When the mote is turned on, the first operation that MRM performs is to copy a preprogrammed interrupt table to a specific zone of RAM.

In this way it is possible to modify what the microcontroller interprets as interrupt table, from the space in ROM to the interrupt table just defined in RAM through a specialized register of the microcontroller.

After that, the configuration of the clock, the activation of the CB’s power and the configuration of the peripherals that will be used in the operation of the MRM are made.

In case MRM receives the program command, it is called the IHEX interpreter algorithm in which it remains for an indefinite time until the loading operation is finished or a transmission error detected by the platform occurs.

The flowchart that succinctly represents the actions taken by this algorithm can be seen in Figure 8.

The main operations performed by the IHEX interpreter, responsible for decoding the received data, are well defined and are in agreement with the format defined by Intel HEX [20].

E. Adapting FreeRTOS for extended mode

The FreeRTOS adaptation to the existing MSP430F5438A in the group repository does not support extended mode.

Therefore, to use the protocol stack provided by OpenWSN it is necessary to correct the problem. In addition, the existing version in the repository, one of the first V7 versions was outdated and as there was
already a new V8 version was also decided to upgrade the version of FreeRTOS.

Although an attempt was made to adapt the new FreeRTOS files using the old portable files, a problem with internal compatibilities of the FreeRTOS did not allow it to be done and another solution had to be searched.

Some more consideration has led to the idea of adapting the portable files, kept from source by CCS, which are written mostly in Assembly language. It would only be necessary to arrange a form of the assembler provided in the msp430-gcc tool to integrate these files into the assembly phase.

A project on the GitHub platform that implemented this solution was discovered. The compilation using the portable files in the FreeRTOSv8 assembly with mspgcc. Even this adaptation was performed for the microcontroller used in our platform. The code can be accessed at [github](https://github.com/sbach/msp430-freertos-smartwatch) – Jan 2017.

Since FreeRTOS is partially integrated with OpenWSN, as previously mentioned, all that is need to do is add and update the FreeRTOS files that were just modified to the compilation environment.

In addition, a project definition has been added in order to perform the commutation of the compilation in extended mode or in simple mode.

In order to validate the implementation of the OpenWSN project we start by using sample projects that are possible to find in the OpenWSN project.

### VI. Validation

First we perform a validation of the operation of the protocol stack used in the OpenWSN project with the minimum possible changes. This implies performing the tests with an original version using the TelosB platform.

In addition to this application there are features that allow ICMP ping reply observable in Figure 11.

![Figure 9: TelosB Platform.](image)

![Figure 10: image shows the result of a GET request executed in the Mozilla Firefox browser using the CoAP Copper application.](image)

A. Test of the adaptation of the OpenWSN project

A MoteIST is connected to the computer and is defined as the a gateway node. Other MoteIST are connected to a different computer (for power) and the TelosB is powered by two AA batteries since this platform is already equipped for this purpose.

Verification that in fact there are 5 motes (4 MoteISTv5 and 1 Telosb) and the network routing tree is flat because the nodes are within range of each other and no
restrictions were applied to their topology is visible in Figure 12.

![Routing](image1)

**Figure 12:** Natural network routing topology formed by the OpenWSN protocol.

### B. Network configuration

The test network consists of five MoteIST who are arranged on a desk. In this network there is a portal node that connects to the computer via USB. For practical purposes, all nodes are powered through the existing USB adapter in each.

In the configuration of the network it was decided to use a linear topology since this way it allows to run performance measurements with the worst type possible of topology for the network.

The test of this type of topology is particularly important in networks with synchronized communication because it allows, among other factors, to test the stability of the network, the propagation of synchronism at several distance jumps, measure various metrics of the network communication performance in between nodes namely latency, reception ratio and transfer rate.

Figure 13 provides a comprehensive illustration of the topological configuration used in the test network.

![Topological Configuration](image2)

**Figure 13:** Illustration of the topological configuration used in the test network.

Using the OpenWSN topology.c file it is possible to force the nodes to only accept packets from the adjacent nodes (in the case of this test), so that packets received from other nodes are ignored before they reach protocol layers higher than MAC as per Figure 14.

![Forced Network Routing](image3)

**Figure 14:** Forced network routing topology.

### VI. CONCLUSION

At the beginning of this project the intention was to use and adapt one of the distributed protocol stacks that come with the FreeRTOS, but it was quickly realized that these protocol stacks did not meet our objectives by not adapting to communications inside WSN.

The OpenWSN project adapts FreeRTOS to an innovative protocol stack, IEEE 802.15.4E. This protocol stack is a proposed improvement to the state of the art used in WSN, 802.15.4, and has as critical points in its improvement the use of TSCH. By allowing the use of multi-channel in the transmission of information in the network an increase in the global transfer capacity of the network and the reduction of multipath fading is achieved through the periodic variation of the transmission channel.

The outstanding differences between the 802.15.4 and 802.15.4E protocols are interesting to be explored in depth both in the area of Emergency Detection and Response and in Industrial Wireless Sensor Networks applications known as IWSN.

The use of synchronized networks in fact implies the additional consumption of energy for the maintenance of the network in the synchronization of the transmission slots and in the maintenance of the routing map of the network. In any case it prevents collisions between several nodes that have their state of emergency activated at the same instant. Since mesh networks have at least one gateway and where wireless activity is higher, multiple portal nodes can be used to extend the network lifespan, which is foreseen in the IEEE 802.15.4E protocol but which is not yet been implemented in the OpenStack protocol stack.

To finish the defined protocol IEEE 802.15.4E can be said that it is not yet mature and many times problems are encountered in its execution.

In the implementation of FreeRTOS there is still room for improvement. By adapting FreeRTOS to the protocol stack, many of the features provided by the OS such as queues, timers and true parallelization, have
been deprecated. Instead, OpenWSN adds somehow OpenOS as an overhead to FreeRTOS only taking advantage of the interruption of tasks, scheduling and replacement of running tasks with higher priority ones.

In the implementation of the new MRM3 the ease of loading code in IHEX format is an asset because it allows to use a generic serial communication tool to install the application.

In the course of developing this project I contributed with the implementation of:

- The adaptation of the OpenWSN project to the moteIST platform.
- The update of the source code of FreeRTOS to be used in this project, and is also useful in future projects carried out by the students of the group.
- The possibility of compiling code in extended mode in order to be able to use all the ROM existing in the msp430f5438a microcontroller.
- The update of the virtual machine used by the group with the new build environment installed and configured.
- The modification of MRM to allow loading code in IHEX format directly from a serial terminal with data flow control in software, eliminating the need to use specific software to carry out this operation.

B. Future Work

The original intent of this thesis was to compare the performance of the new IEEE 802.15.4E protocol with the original IEEE 802.15.4 version implemented in other existing projects such as TinyOS and Contiki, where the main difference is that the latter do not use an OS of real time.

In the case of MRM3 there is an opportunity to improve the reliability of this bootloader using a multi-image loading system. Knowing that in the moteIST platform there is a non-volatile external flash it would be interesting to modify the MRM3 to record images and upload images from it. In addition the use of generic channels for loading the image using the microcontroller's source bootloader is an improvement to take into consideration in the construction and development of a next platform

REFERENCES