A Multilayered algorithm for topology control and self-healing on Wireless Sensor Network

by

Hernan David Santos Barrera
Director: Diego Mendez Chaves

A thesis submitted in partial fulfillment for the degree of Magister on Electronics Engineering

in the
Faculty of Engineering
Department of Electronics

June 2017
Acknowledgements

I would like to thank my parents for all the support, love and caring they have game me through all these 25 years. Without them this would have never been possible. Special thanks to my mom for staying with me when the nights were long and coffee was not enough.

Thanks to my grandmother Maria de Jesus who does not always express her love with words but with many simple special actions. You are a second mother to me.

Thanks to my friends, for helping me through the process and standing by my side. Special thanks to my friends and my girlfriend Carolina from the robotics Lab for helping me coding, generating ideas, commenting on my papers and also for distracting me with games, trips and movie nights when stress came in. I want to thank again my girlfriend for being unconditional staying with me many times until late in the night, for always always caring about my well being and for always expressing me her love even on rough Thesis days.

Thanks to my Director Diego for guiding me trough the process, for the patience he had and for opening many doors for me.

And finally thanks God the architect and biggest promoter of my life.
Contents

Acknowledgements i

List of Figures iv

1 Introduction 1

2 Theoretical Framework 4
  2.1 Contiki ............................................. 4
    2.1.1 Physical and Data Link Layers ................. 4
    2.1.2 Network and transport Layer .................. 6
      2.1.2.1 6LoWPAN .................................. 6
      2.1.2.2 RPL ....................................... 8
    2.1.3 Transport Layer ................................ 9
    2.1.4 Application Protocols .......................... 9
      2.1.4.1 CoAP ..................................... 9
      2.1.4.2 MQTT ................................... 9
  2.2 Selecting a Subset of nodes ........................ 10
    2.2.1 K-means ..................................... 11
    2.2.2 Particle Swarm Optimization (PSO) ............ 12

3 Specifications 13
  3.1 Architecture .................................... 13
  3.2 Requirements ..................................... 14
    3.2.1 Host-PC ..................................... 14
    3.2.2 Wireless Nodes ................................ 15
      3.2.2.1 Hardware employed on the Tests ............ 15
    3.2.3 Contiki Software ............................... 17

4 Development 19
  4.1 Algorithm ....................................... 19
  4.2 Implementation .................................. 22
    4.2.1 Topology Construction ......................... 23
    4.2.2 Clustering and CHs Selection ................. 24
    4.2.3 Topology reconstruction ....................... 26
      4.2.3.1 RDC .................................... 26
      4.2.3.2 Leaf mode ................................ 26
    4.2.4 Disjoint Path Construction ................... 27
5 Results and Analysis

5.1 Individual tests

5.1.1 Topology construction Test

5.1.2 Clustering and CHs Selection over Matlab Tests

5.1.3 RDC Tests

5.1.4 Leaf mode test

5.1.5 Test of dead node detection and correction

5.2 Energy Tests

5.3 Cooja Tests

5.3.1 Real Node Tests

5.3.2 MQTT - Tests

6 Conclusion

Bibliography
List of Figures

1.1 A Wireless Sensor Network .............................................. 1
2.1 Contiki Network Stack ................................................. 5
2.2 6LoWPAN Header Compression. Taken from: [1] ..................... 7
2.3 MQTT publish/subscribe architecture taken from [2] ............... 10
2.4 A Simple example of a Clustered Network ............................ 11
3.1 Architecture proposed ................................................ 13
3.2 Architecture proposed ................................................ 16
3.3 Architecture proposed ................................................ 16
4.1 Contiki Network Stack ................................................. 20
4.2 Architecture proposed simplified .................................... 22
5.1 Architecture proposed simplified .................................... 30
5.2 A 6LoWPAN Network displayed on a 100m$^2$ area. 30 nodes randomly positioned are shown on Blue. The Sink node on the center of the figure at coordinates (50,50). ........................................... 31
5.3 The same network after K-means algorithm was executed. 6 different Clusters are shown on different colors. ......................... 31
5.4 After executing PSO 6 nodes (One of each cluster) are selected as a CHs. ................................................................. 32
5.5 Resulting Coverage of the new Network with the subset of nodes. This coverage includes the coverage of the Sink Node too. ................................. 33
5.6 Resulting Coverage of the new Network with the subset of nodes. This coverage includes the coverage of the Sink Node too. ......................... 34
5.7 A network with 25 Nodes running RPL ................................ 35
5.8 The same network shown on Figure 5.7 but now with node 26 as a Leaf node ................................................................. 35
5.9 A WSN of 25 nodes running RPL algorithm. ........................ 36
5.10 RPL Historic Routing Metric on the WSN. ......................... 37
5.11 New Topology after RPL is stopped and node 16 reselcts a new parent thanks to the heartbeat he was running. ......................... 37
5.12 Received packets per node. Here only node 16 from all of the node’s 7 childs recovered and lost a few packets but started to work normally. The other childs of node 7 never recovered because heartbeat algorithm was not active ......................................................... 38
5.13 Topology used for the test on Cooja simulator ...................... 39
5.14 Historical power consumption on the sky motes on Cooja ....... 40
5.15 Current consumption of a CH node ................................... 41
5.16 Current consumption of a node changing from CH to Non-CH . . . . . . . 42
5.17 Network deployment Test, the nodes are identified by their last 2 octets
    of their IPv6 addresses. The Border-Router is identified by the "BR"
    letters on the map . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 43
5.18 Web Browser application. Routes and Neighbors of the Border Router
    are sent to the Browser on HTTP format . . . . . . . . . . . . . . . . . . 44
5.19 Batteries of the nodes after more than 30h of test. The CH nodes de-
    pleted their batteries very fast while Non-CHs still continued to operate
    normally, saving their energy. . . . . . . . . . . . . . . . . . . . . . . . . . 44
5.20 MQTT server application running on the Host PC . . . . . . . . . . . 46
5.21 MyMQTT App subscribed to the topic DACA/info/# showing two mes-
    sages of the nodes f3f and aace, a total of 870 messages received from the
    6LoWPAN network . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 47
5.22 MQTTlens application running on windows subscribed to the topic DA-
    CA/info/# . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 48
Chapter 1

Introduction

A Wireless Sensor Network (WSN) consists of tenths, hundreds or even thousand of wireless nodes that are equipped with different types of sensors used to detect several characteristics of its environment such as temperature, light, humidity, sound, pH, pressure, and many others. Having all this information available online is one of the goals for the Internet of Things. A brief example of a WSN can be seen on Figure 1.1. here we see a WSN conformed of several nodes (on green) and a Border-Router (on red) that communicates the network with the Host-PC. The network might often have a internet connection through the Host-PC this way all the resources and the data gathered from the network can be accessed online.

WSNs are normally deployed on remote locations with no connection to a electrical line (for instance a WSN deployed on a remote crop to monitor humidity, light, PH of the ground) so they normally operate with a limited power source such as batteries. This limitation could present several problems over all the network; for example one of the most sensible scenarios is when a node that is central on the network and routes information of many nodes (acting as a routing father) depletes its energy source, this
would disconnect all the nodes connected to him from the network, resulting on: packet loss, and energy waste on a big portion of the network.

To address the limited energy on the nodes, and stopping network disconnection problems, different algorithms can be implemented to extend the lifetime of the network [3]. On this document we present an extension of the algorithm DACA - Disjoint Path and Clustering Algorithm for Self-Healing WSN [4] on which we aim to extend the lifetime of the network by selecting a subset of nodes that will be working normally while the other nodes that are not selected would be on low-power mode and stay as back up nodes. We extend DACA and implement it over a real nodes using different techniques adapted to real hardware using Contiki OS\(^1\) (the Operative System running on the nodes).

Energy consumption also limits the options of the wireless connectivity, wireless technologies such as Wi-Fi and Bluetooth use a huge amount of energy. Therefore, low power radios with duty cycling mechanisms are crucial and commonly found on this types of networks. One of the most used standards for the physical layer is the IEEE802.15.4, this standard is designed for networks that have low throughputs and need to consume low energy and it works on the 2.4GHz band. Technologies like Bluetooth Low Energy (BLE) and ZigBee are supported over IEEE 802.15.4 but have several restrictions, one of the most crucial restrictions is that they do not use IP addresses. This makes communication between IP networks rather hard.

IPv6 is expected to accommodate a huge number of devices which is perfect for IoT and WSN but this protocol has very large headers that would occupy almost all of the IEEE802.15.4 packets so to make IPv6 on WSN more efficient the 6LoWPAN adaptation layer was created by the IETF. We implemented a 6LoWPAN network with IPv6 addressing and a Border Router to communicate to normal IP networks. Additionally we published all the information gathered from the sensor nodes using an MQTT (Message Queuing Telemetry Transport) [5] Server running on the Host PC.

On this work we implemented a Border-Router to enable seamlessly communication with other IP networks and furthermore with public IPv6 address on the network we can make all the nodes available on the internet. This makes reaching the network, configuring the nodes, requesting data from the sensors or triggering an actuator on the nodes from a remote pc a lot more easier that on networks ZigBee, Z-wave or BLE [6]. Using MQTT we can also make the nodes subscribe to MQTT so they publish the data they gather and also they can be configured all at the same time when they are subscribed to the same topic.

\(^1\)http://www.contiki-os.org/
To explain the development of the presented solution we divide this document as follows: on chapter 2 we present the theoretical framework needed to develop the solution, on Chapter 3 we present the specifications and requirements for the solution, on chapter 4 we explain the development and the separate tests done to implement the algorithm, on chapter 5 we present the results and analysis of the work and finally we conclude the results of this implementation on chapter 6.
Chapter 2

Theoretical Framework

To develop, implement and test the algorithm we used the Contiki Operative System on the nodes (real and simulated). Here we describe the Contiki Operative system and mostly its network Stack.

2.1 Contiki

Among different Operative systems for WSNs like Tiny OS, MANTIS, Nano-RK we selected Contiki OS. Contiki is an open source operative system for IoT specially designed for networked, memory-constrained and power constrained wireless devices created by Adam Dunkels [7].

To enable the IPv6 communication on the whole network the Contiki Network stack was used. A diagram of the Contiki network stack and the protocols selected for the implementation is shown on figure 2.1. All of the protocols chosen on each layer are explained one by one on the following subsections.

2.1.1 Physical and Data Link Layers

Given that the devices on a WSN are memory constrained and use limited sources of energy (for example on this project we used 1300mAh Li-Po batteries) the communication should be as short and simple as possible. Sending the minimal information required saves the energy on the node which extends the lifetime of the network overall. Contiki uses the standard IEEE 802.15.4 on its physical and Data link layer to save energy. All the radios ported on Contiki as cc1200, cc2520, cc2420 and others, all designed to compile with IEEE 802.15.4.
The protocol IEEE 802.15.4 is a technical standard specifies the physical layer and Medium Access Control (MAC) of low-rate wireless personal area networks. The data link layer defined by the protocol is managed by Contiki using 3 different layers: the framer, Radio Duty-Cycle (RDC) and Medium Access Control (MAC). Contiki has different drivers for all of these three layers. For our implementation we selected on the framer the 802.15.4 framer, for RDC Contiki-mac [8] and for MAC we used Carrier-sense multiple access with collision avoidance (CSMA/CA).

The RDC layer handles the sleep period of the nodes. This layers decides when the packets are transmitted and makes sure neighbors nodes are awake and ready to receive packages. The RDC used (Contiki-mac) allows the nodes to keep their radio off for most of the time depending of the selected channel check rate configuration. On its lowest configuration (8Hz channel check rate) the radio will be off for more than 99% while being able to route multi-hop messages. Selecting a low channel check rate saves a lot of energy given that listening on the radio is one of the most energy consuming processes. Still this will make neighbor discovery slow and this will delay topology construction so this parameter must be selected carefully depending on the application.
CSMA protocol keeps a list of packets to each of the neighbors and calculates statistics such as number of re-transmissions, collisions, etc. CSMA receives the incoming packets from the RDC layer and uses this same layer to transmit packets. The RDC and the physical layer sense the medium and if it is busy CSMA will retransmit later the packages. The medium access check is performed by the RDC driver. Although CSMA normally uses carrier-sense medium access, the Contiki implementation does not rely on carrier sensing because the medium access is performed by the RDC protocol.

2.1.2 Network and transport Layer

Contiki provides three different network stacks: IPv4, IPv6 and Rime. Rime is a lightweight layered communication stack for sensor networks [9] but it does not support IP communications, making the nodes in the network not available on the internet, so we decided to use either IPv4 or IPv6. On IoT an estimated of 50 Billion devices (or things) would be connected to the internet by the year 2020 [10]. Having this in mind and the fact that IPv4 has a very limited address space IPv6 was created (IPv6 covers a space of $3.4 \times 10^{32}$) so IPv6 was selected for this implementation.

IPv6 was designed having in mind devices with high power and very capable processors, the constant increase of internet speeds IPv6 increased the minimum maximum transmission unit (MTU) from 576 on IPv4 to 1280 bytes on IPv6. On the other hand, 802.15.4 was designed for low cost ultra-low-power devices, the throughput is limited to 250Kbps, and the full length of its frame is limited to 127 bytes. Since IPv6 is not designed for 802.15.4 devices this presents several problems, for example on 802.15.4 the effective payload is of 81 bytes and IPv6 and UDP use headers of 40 bytes and 8 bytes respectively, this leaves 33 bytes for application information which seems too low. Additionally IPv6 is not designed to take into account link failures and asymmetric links that are present on all WSNs. There for 6LoWPAN was created as an adaptation layer to comply with the requirements and limitations of 802.15.4.

2.1.2.1 6LoWPAN

6LoWPAN is an open standard defined in [RFC6282] by the Internet Engineering Task Force (IETF). One of the best features of 6LoWPAN is that while originally conceived to support IEEE 802.15.4 low-power wireless networks in the 2.4-GHz band, but now it is now being adapted and used over a variety of other networking media including Sub-1 GHz low-power RF [1].
The main focus of the IETF working group, was to optimize the transmission of IPv6 packets over low-power and lossy networks (LLNs) such as IEEE 802.15.4 and led to the specifications of Header compression, fragmentation and reassembly and stateless auto configuration.

Header compression is done to the 40-byte IPv6 and 8-byte UDP headers by assuming the usage of common fields. The way that the headers can be compressed is one of the factors that led to the standard only supporting IPv6 and not IPv4.

Three scenarios of header compression and the resulting size of the 6LoWPan header are listed below:

In the example in Figure 2.2 can be seen three example of the header compression:

1. If the communication is with another device inside the same 6LoWPAN network it can be compressed to 2 bytes

2. If the communication is between a device outside the network and the network prefix is known it will use 12 bytes.

3. If the communication is with an external device and the network prefix is not known it will occupy 20 bytes

Due to the difference on the length of the MTU of both IPv6 and 802.15.4 6LoWPAN fragments and re-assembles the packages. 6LoWPAN also auto generates on every device its own IPv6 address and has mechanisms to detect if there is another device on the network with the same address.
Contiki uses different routing protocols like the routing protocol for low-power and lossy networks (RPL) or the Ad hoc On-Demand Distance Vector (AODV). AODV is a routing protocol for mobile ad hoc networks RFC[3561], it is the routing protocol used on ZigBee networks and it works with IPv4. RPL was selected as a routing protocol because is the only protocol on Contiki that has full IPv6 support.

2.1.2.2 RPL

RPL is the IPv6 routing protocol for low power and lossy networks designed by the IETF. RPL is a distance vector protocol, it starts finding routes as soon as the RPL network is initialized by the sink. RPL is used primarily with a 6LoWPAN network.

It can support three different traffic patterns:

- Multipoint-to-point (MP2P)
- Point-to-multipoint (P2MP)
- Point-to-point (P2P)

RPL creates a destination-oriented directed acyclic graphs (DODAGs) rooted towards one sink that is called DAG-ROOT. All the DODAGs are optimized using an Objective Function (OF) metric. This metric can be hop count, latency, expected transmission count, parents selection, energy consumption, or others defined by the user. Over this implementation its used the Minimum Rank with Hysteresis Objective Function (MRHOF), this OF uses the Expected Transmissions metric (ETX) as routing metric, this is the same metric that is used over Collection Tree Protocol (CTP) and many other routing protocols. With the metric is calculated a rank number on each node and this can be useful to determine its relative position and distance to the root over the DODAG.
2.1.3 Transport Layer

Contiki fully supports TCP and UDP protocols, for several applications with critical measurements such as e-health where the patient needs to be monitored remotely with high precision a protocol like TCP would be preferable. Still, for this application losing a few packets might not change the results; additionally 6LoWPAN is optimized for UDP so for that reason UDP was selected as the transport protocol.

2.1.4 Application Protocols

IoT uses several protocols such as CoAP, MQTT, XMPP, AMQP. In this section we will talk about CoAP and MQTT.

2.1.4.1 CoAP

The Constrained Application Protocol or CoAP is a specialized web transfer protocol for use with constrained nodes and constrained networks in the Internet of Things. This protocol is designed for machine-to-machine (M2M) applications such as smart energy and building automation[RFC7252].

Coap is intended for simple electronic devices and it allows them to communicate over the Internet, it is targeted for small, low power sensors, switches, and similar. CoAP like HTTP provides a request/response model between application endpoints. CoAP is designed to translate to HTTP for simplified integration with the web while fulfilling several requirements such as multicast support, low overheads and simplicity. This protocol runs on most devices that support UDP.

2.1.4.2 MQTT

MQTT (or MQ Telemetry transport) is a lightweight messaging protocol that runs on top of TCP/IP protocol. The protocol uses a publish/subscribe architecture in contrast to HTTP with its request/response paradigm. It is designed to connect remote devices with memory constrains or the network bandwidth is limited. The architecture of MQTT can be seen on Figure 2.3. The clients publish a message to the broker including a topic, this topic will be the routing information for the broker. The clients that want to receive message subscribe to a topic and the broker delivers all the messages with the selected topic to the client. Here we can notice that the message broker is the central part of the communication, he is in charge of dispatching all messages between the senders and the
rightful receivers. The biggest difference to HTTP is that a client does not have to ask for the information it needs, but the brokers push the information to the client if there is new information from the senders. Therefore each MQTT client has a permanently open TCP connection to the broker.

![MQTT publish/subscribe architecture](image)

**Figure 2.3:** MQTT publish/subscribe architecture taken from [2]

### 2.2 Selecting a Subset of nodes

As a technique to increase the lifetime of the network clustering algorithms were created. They divide the network into subsets of nodes in certain geographical areas. After defining the clusters they choose one of the nodes that will characterize the zone, this node will be called Cluster head (CH), as shown in Figure 2.4, and the nodes that are in the same group but not the child Cluster Head. Initially the LEACH algorithm was proposed to define the clusters. This algorithm was proposed in 2000 and a large part of algorithms used for this purpose are based on it. There are other algorithms such as HEED, EEHC, DCA, GAF, SPA, TEEN and others[12].

We can initially classify all types of algorithms depending on the purpose of Clustering. Energy preservation is one of the most important of this purposes. This kind of algorithms can be classified by *scalability, fault tolerance, data fusion/aggregation, load balancing, network topology stability or network lifetime*.

LEACH is one of the first attempts in the area of clustering techniques on WSNs. The main idea behind LEACH is to rotate the role of CH among all nodes to achieve load
balancing. The algorithm first defines the clusters and then makes the clusters change state, changing the CHs. The algorithm works as follows: Each node selects a random number between 0 and 1. If the number is less than a certain threshold the node becomes the CH. LEACH is very simple and distributed, this generates few messages for CH selection, the load is balanced and the percentage CHs in the network is adequate and can be fixed. However LEACH has certain flaws, the communication between CHs and the Sink is straight (this means there are no intermediate hops) so the power of these CH nodes, especially from ones that are closer to the sink, can be rapidly depleted. In addition, the algorithm does not take into account the remaining battery of each node, so that a node with little battery can be chosen as CH. From this problems there have been presented new algorithms to improve LEACH like TEEN, APTEEN, PEGASIS, CLUBS, EEHC and others that take into account other objectives and parameters to optimize.

### 2.2.1 K-means

K-means is a method of vector quantification, originally from signal processing. K-means uses an iterative refinement technique, and it involves two different phases. On the first phase on each iteration all nodes are reassigned to their nearest cluster centroid all at once, afterwards the cluster centroids are recalculated. Sometimes this phase does not converge to a solution that is a local minimum, so on phase two the nodes are reassigned one by one if by doing this it reduces the sum of Euclidean distances from a cluster then cluster centroids are recalculated. This individual reassignment is done to all the nodes one by one on every iteration. This way k-means will converge to a local minimum.
K-means is of low complexity and good results when only the distance is used as a parameter to define the clusters.

2.2.2 Particle Swarm Optimization (PSO)

PSO is a multi Objective Optimization method was inspired by the social behavior of a bird flock or a fish shoal. The PSO algorithm works by having a population (or swarm) of \( n \) candidate solutions (particles), on this specific scenario the number of particles will be the number of clusters defined previously by k-means. The particles explore the area of interest for the WSN, this can be a 100m x 100m crop for example, in search of a global solution. The movement of each particle is defined by the velocity (4.1) and (4.2) equations:

\[
V_i(k + 1) = \phi(k) + \alpha_1[\gamma_1(p_i - X_i(k))] + \alpha_2[\gamma_2(G - X_i(k))] \tag{2.1}
\]

\[
X_i(k + 1) = X_i(k) + V_i(k + 1)\Delta t \tag{2.2}
\]

Where \( i \) is the particle index, \( k \) is the discrete time, \( p \) is the best position found until that iteration, \( G \) is the best global position, \( \gamma \) is a random number between 0 and 1, \( \alpha \) is acceleration constant and \( \phi \) is the inertia (this might also be a function). On each iteration the velocity and the position are updated while evaluating the OF on each particle.
Chapter 3

Specifications

The main objectives of this project includes creating a WSN that has algorithm for topology construction, dead node detections and self-healing. All of this supporting IPv6 protocol so that the network can be reached from a normal IP network. With this in mind we would show the Architecture used and the platforms used.

3.1 Architecture

Having in mind the connection to the internet with a WSN, we proposed the architecture shown on figure 3.3.

![Figure 3.1: Architecture proposed](image)

Here we have a WSN made entirely by nodes are running Contiki OS. All of the nodes are running the exact same program except from the Border-Router Node and they make
their own tree and route all the information to the Sink (Border Router). The border router is connected to the host using the tool tunslip6. This tool is used to bridge all the IP traffic between the host and the border router over a serial interface (in this case USB). It creates a virtual network interface called tun0 on the host pc and uses SLIP (serial line internet protocol) to encapsulate and route all the IP traffic to and from the other side of the serial line. With an internet connection from the Host and a public IP on the border router this network would now be visible from the internet. Using this bridge, all the nodes send their measurements to the host pc and all this information is sent to the MQTT (Message Queue Telemetry Transport) Broker running a simple MQTT server on the host PC.

### 3.2 Requirements

Having the architecture previously presented in mind, to execute the Algorithm different requirements are needed and will be presented on the following subsections.

#### 3.2.1 Host-PC

On the host PC there should be a Ubuntu linux distribution running natively or there should be a Virtual linux machine with Ubuntu OS on a PC with a different Operative system. The following software must be installed on Ubuntu:

- Matlab 2015
- The GNU Compiler Collection (version 4.6 or higher)
- Git
- Python 2.7

To execute the Algorithm over Matlab the following several requirements have to be met and have to be declared on Matlab:

- The total of active nodes the network has initially
- All the exact relative positions on the area of interest (the area that the user wants to analyze in terms of coverage) of every node and has to be sent to matlab on a 2xN array to Matlab where N is the total number of nodes.
• The IPv6 address of every node on the network so that they can be remotely configured.

• The estimated area of coverage of a node, this means the area where a measured physical phenomenon (for example Temperature) does not change drastically. This allows us to assume that a certain area will be characterized by the measurements done by just one node and the information of near by nodes inside the area of coverage will be redundant.

3.2.2 Wireless Nodes

The WSN must be conformed with nodes that are already supported on Contiki, a list of some the supported nodes that are already on Contiki OS can be found at: http://www.contiki-os.org/hardware.html. If there is a different node a porting has to be done to Contiki and must have the following characteristics:

• A IEEE 802.15.4 Radio working over 2.4GHz

• At least one Serial Port

• A Contiki supported processor such as AMR Cortex M3 or Texas Instruments MSP430

• ADCs or in-built Sensors

• A battery voltage sensor

• At least 50kB of flash memory

• At least 10kB of RAM memory

3.2.2.1 Hardware employed on the Tests

For our implementation we used two different motes already supported over Contiki: The Re-mote by Zolertia and The CM5000 from advanticsys. For the simulations we used the Sky mote, this is the same mote as the CM5000, already supported on Cooja Simulator and for the real tests we used the Re-mote by Zolertia.

The sky motes are based on the TelosB mote, this mote has the following specifications:

• IEEE 802.15.4 compliant
Figure 3.2: Architecture proposed

- 250 kbps, high data rate radio
- TI MSP430 microcontroller with 10kB RAM
- Integrated onboard antenna
- Data collection and programming via USB interface
- Open-source operating system
- Integrated temperature, light and humidity sensor (SHT11)

The Re-mote has the following specifications:

Figure 3.3: Architecture proposed

- ISM 2.4-GHz IEEE 802.15.4 and Zigbee compliant radio.
- ISM 863-950-MHz ISM/SRD band IEEE 802.15.4 compliant radio.
• ARM Cortex-M3 32 MHz clock speed, 512 KB flash and 32 KB RAM (16 KB retention)

• AES-128/256, SHA2 Hardware Encryption Engine

• ECC-128/256, RSA Hardware Acceleration Engine for Secure Key Exchange

• User and reset button

• Consumption down to 150 nA using the shutdown mode.

• Programming over BSL without requiring to press any button to enter bootloader mode.

• Built-in battery charger (500 mA), facilitating Energy Harvesting and direct connection to Solar Panels and to standards LiPo batteries.

• Wide range DC Power input: 3.3-16 V.

• Small form-factor (73 x 40 mm).

• MicroSD (over SPI).

• On board RTCC (programmable real time clock calendar) and external watchdog timer (WDT).

• Programmable RF switch to connect an external antenna either to the 2.4 GHz or to the Sub 1 GHz RF interface through the RP-SMA connector.

• Supported in Open Source Operative Systems as Contiki or RIOT.

3.2.3 Contiki Software

To develop the solution we used Contiki versions 2.7 and 3.0 and also the Contiki on 5 days repository over the nodes. We also used from Contiki on the Host PC The Cooja Simulator, this is Contiki’s network simulator. This simulator allows large nad small networks of Contiki motes to be simulated. Cooja currently can simulate the following motes:

• exp5438

• Z1

• Wismote

• micaz
The Sky mote is based on the TelosB mote and the CM5000 motes used on this project have the same architecture so we designed the solution to also fit on the CM5000 mote taking into account its very limited ROM. Contiki also provides a Graphic User Interface called Collect-view that connects to a real or virtual serial port of a node and reads the data sent to this node printed over the UART. Collect-view graphs Network parameters such as, packets received, latency, hops to the sink, measurements of all the sensors connected to the nodes, energy usage and others. From Contiki we also used the tunslip6 interface as it is explained over the Architecture Section.
Chapter 4

Development

4.1 Algorithm

On WSN several nodes sense variables like temperature that do not necessarily change substantially on small distances, this makes that different sets nodes that are close to each other send the exact same information creating redundant data. The purpose of this algorithm is to select a subset a nodes that can cover almost the same area as if the network was using all the nodes, this would sacrifice a little of the coverage of the network while extending a greatly the network lifetime by saving the battery of the unselected nodes. Additionally we aimed to implement preemptive mechanisms to recover from certain faults and also implement self-healing mechanisms to correct the faults.

The algorithm proposed is based on DACA [4] and is shown on Figure 4.1. The algorithm has a mixed distributed and centralized approach. Some of the techniques here proposed were implemented over Matlab software (this would be centralized) and others are implemented directly over the nodes using the operative system Contiki on a distributed manner. To connect the nodes running Contiki with Matlab software we implemented a UDP communication that would allow Matlab to send configuration messages, this communication will be expanded later on.

As follows we will explain the different steps or layers in which is divided the algorithm:

1. **Topology construction:** Initially the network must create a topology to route messages from all the WSN to the Sink. Several routing protocols have been implemented for WSN. On DACA the Collection Tree Protocol was used but over this implementation we selected the IPv6 Routing Protocol for Low Power and Lossy Networks (RPL) given that this protocol was specifically designed for Low
Power and Lossy Networks and is also optimized to support IPv6 which is one of the goals we were looking for on the project. Initially the nodes construct and maintain the topology by using RPL. After a small time the nodes start to gather data from all the sensors and then they send it to the Host PC through a UDP connection, this messages include the voltage level of the battery and are sent periodically.

2. **Cluster Definition:** After Topology is constructed the network can now send all kinds of messages so we look to select the best subset of nodes that would save the energy of the network while trying to maintain the same coverage. So we selected k-means to divide the network into clusters because k-means is simple, it often divides the clusters looking to similar density, its fast and efficient in terms of computational cost. We assume that all the positions of all the nodes on the network are known on Matlab, then we execute K-means Algorithm on Matlab assigning each node to a new specific cluster.
3. **CHs Selection:** We then select one Cluster head from each cluster using a Multi-objective optimization. We use PSO because it has been a very popular technique used to solve different optimization problems over WSNs due to its fast convergence, simplicity, high quality solution and minor computational burden. To start the optimization we first update the battery level of the nodes with the report messages that are being sent from each node periodically through UDP. Using the battery level value of the nodes and the approximate coverage area of each node (where measured variables do not change drastically) we select a node from each cluster. Therefore, PSO algorithm is executed to select a node that will become the Cluster Head (CH) on each cluster, maximizing the coverage area of the whole network while taking into account the battery levels of each node.

4. **Topology reconstruction:** After the CHs are selected on Matlab we send a configuration messages to all the nodes indicating if they are selected or not as CH. The non-CHs will: enter a low power mode, stop sensing and sending information and become a leaf node (a node that does not have children or route information). We used leaf nodes because they are part of the RPL definition and it is an effective way to exclude the non-CHs of different processes of the network to save more battery and extend overall network lifetime. We then wait for RPL to converge with this new leaf nodes re-configuring the topology of the network and creating a topology where the CHs are the central nodes of the communication, while the non-CHs can wait for further instructions.

5. **Disjoint path construction:** Now that the CHs were selected and the topology is re-configured we stop RPL execution on all the nodes to evaluate the algorithm performance without the help of RPL. Afterwards we construct the disjoint paths for each node so that the nodes have alternate routes to reach the sink in case its own parent dies. We based our disjoint path algorithm on N-to-1 protocol [13] and modified it given that this protocol stores on the messages the node exchange the full path of a node to the sink and with IPv6 addresses the overhead huge. This resulted on a very light receiver-initiated algorithm that constructed disjoint paths correctly.

6. **Dead Node Detection and Correction:** To finish we use a simple heartbeat on the CHs so that every node can check the status of its parent and if it dies use a disjoint path and advertise the user that the parent is dead. When this happens the algorithm is re-executed from the cluster definition to re-select CHs excluding the dead node from the possible selection. This heartbeat is almost like an echo-request (ping6) from node to node keeping it simple and with low overhead while
it is done on a distributed manner on each node on the network making it faster for fault and specially dead node detection.

4.2 Implementation

As noted before the implementation of the algorithm is partially done over the nodes and another part is implemented on a Host PC using the software tool Matlab. To implement the algorithm we used the architecture shown on Figure 4.2. On this Architecture we used a 6LoWPAN WSN because we wanted the network to have IPv6 protocol so that each of the nodes with its unique IPv6 address could be reached from another IP network connected to the Host PC. To implement the algorithm we used the architecture shown on

![Figure 4.2: Architecture proposed simplified](image)

With just an Internet connection on the Host PC this network can be reached from anywhere on the internet. Having a 6LoWPAN network with full IPv6 addressing allows us to send a UDP package from Matlab with the desired configurations to a specific node on the network with just knowing the IPv6 address of the node.

On the architecture we have a 6LoWPAN network made of N nodes and a Border-Router that is connected to the Host PC through a USB connection. We used tunslip6 tool on the host to bridge all the IP traffic between the host and the border router over the serial
interface (in this case USB). Tunslip6 creates a virtual network interface called tun0 on
the Host PC and uses SLIP (serial line internet protocol) to encapsulate and route all
the IP traffic to and from the other side of the serial line. With an internet connection
from the Host and a public IP on the border router this network can be visible from the
internet. On the following subsections its explained the implementation of each step of
the algorithm.

4.2.1 Topology Construction

To construct the network topology we implemented RPL (IPv6 routing protocol for low
power and lossy networks) [14] designed by the IETF specially for Low Power and Lossy
Networks (LNNs) using Contiki OS. The basic goal of RPL is to provide efficient routing
paths for different traffic patterns like Multipoint-to-point (MP2P), point-to-potin (P2P)
and Point-to-multipoint (P2MP) on LLNs. This protocol is a distance vector routing
protocol that synergizes well with IPv6.

On RPL the protocol starts when the Sink node builds a destination Oriented Directed
Acyclic graph (DODAG) routed towards him (the DODAG-root or Sink). The DAGs on
RPL are optimized using an Objective Function (OF), this function can be changed de-
pending on what is going to be optimized, on this paper we implemented the Minimum
Rank with Hysteresis Objective Function (MRHOF). This OF uses the Expected Trans-
missions metric (ETX) as routing metric, the same metric used several others routing
protocols such as Collection Tree Protocol (CTP). Using this metric the protocol cal-
culates a node’s relative position with respect to the DODAG-root, this is called Rank.
The rank increases as the node moves away from the root on the DAG and decreases on
the other direction.

To maintain the topology of the network RPL specification defines four types of control
messages as ICMPv6 information messages with a requested type of 155. This are the
RPL control messages:

1. DODAG Information Objects (DIO) are the main source of routing control in-
formation. It contains information like the current Rank of the node, the IPv6
address of the root, etc.

2. Destination Advertisement Object (DAO) messages are used to support downward
traffic. Each child node upon joining to a DAG sends a DAO message to its parents.

3. DODAG Information Solicitation (DIS) messages are used to request DIO messages
from a reachable neighbor.
4. DAO-ACK messages are sent by a DAO recipient in response to a DAO message.\cite{14}

RPL specifies two different modes of Downward traffic (From the Sink to a node): Fully stateful or Fully source routed. On stateful mode all the nodes have a routing table with the addresses of their children and another one for their parents. On Source routed the nodes only have a table for their parents, so on downward traffic the root has to know the complete routes to each node. We selected fully stateful on our implementation.

To start the topology creation a SLIP bridge interface is created between the Border-Router and the Host PC, when the Border-router detects the interface is up he will now become the DODAG-root and will create a new DODAG so all the nodes will start to exchange DIO, DAO and DIS messages to create the topology and to maintain it. After a while the optimum DAG will be achieved and every node of the network can be reached from the Host-PC.

\subsection{4.2.2 Clustering and CHs Selection}

Once the network is up and stable the nodes start sensing and transmitting the measurements taken with the sensors including also the measurement of the Voltage level of their own battery, this will be important later on for PSO. On the Host PC we publish all the data to the internet and update the value of the batteries of the WSN. Afterwards we divide the WSN on several clusters using K-means and we select one of the nodes of each cluster to become the CH. We will now explain briefly k-means and PSO algorithm:

K-means is a technique already used on different applications for WSN \cite{15}, it aims to partition N nodes into M cluster with each node belonging to the cluster with the nearest mean distance. It uses an iterative refinement technique, and it involves two different phases. On the first phase on each iteration all nodes are reassigned to their nearest cluster centroid all at once, afterwards the cluster centroids are recalculated. Sometimes this phase does not converge to a solution that is a local minimum, so on phase two the nodes are reassigned one by one and if by doing so it reduces the sum of Euclidean distances from a cluster then cluster centroids are recalculated. This individual reassignment is done to all the nodes one by one on every iteration. With the second phase k-means will converge to a local minimum.

Once the Clusters are created the CHs are selected using PSO, a technique also used on WSN \cite{16}. PSO is a Multi Objective Optimization method that was inspired by the social behavior of a bird flock or a fish shoal. The PSO algorithm works by having a population (or swarm) of $n$ candidate solutions (particles) and they are moved around the area of interest trying to look for the global maximum, on this specific scenario the
number of particles will be the same number of clusters defined by k-means so that each particle moves over just one cluster. The movement of each particle is defined by the velocity (4.1) and (4.2) equations:

\[ V_i(k + 1) = \phi(k) + \alpha_1[\gamma_1(p_i - X_i(k))] + \alpha_2[\gamma_2(G - X_i(k))] \quad (4.1) \]

\[ X_i(k + 1) = X_i(k) + V_i(k + 1)\Delta t \quad (4.2) \]

Where \( i \) is the particle index, \( k \) is the discrete time, \( p \) is the best position found until that iteration, \( G \) is the best global position, \( \gamma \) is a random number between 0 and 1, \( \alpha \) is acceleration constant and \( \phi \) is the inertia (this might also be a function).

On each iteration the velocity and the position of each particle is updated and global objective function is calculated. The objective function used on this paper takes into account three different aspects of the selected nodes: the voltage level of the batteries of the nodes, the Distance from the nodes to its cluster centroid and the total area of coverage created by the selected nodes. We try to find the nodes with the highest battery level, while minimizing the distance from the selected CHs to the centroid of their cluster. The OF evaluated on PSO is:

\[ OF = \max\left(\sum_{i=1}^{n} \frac{V_i}{\max(V)} - \frac{|D_i|}{\max(D)} + C\right) \quad (4.3) \]

Where \( V_i \) is the voltage on the battery on node \( i \), \( \max(V) \) is the maximum Voltage reported from all the nodes, \( D_i \) is the previously calculated distance from the node \( i \) to its cluster’s centroid, \( \max(D) \) is the maximum distance from all the nodes to its cluster. Finally \( C \) is the percentage of the area covered from the area of interest. To calculate the coverage we made approximations using images, to do this we first draw an image of \( N \times N \) pixels in black, then we positioned the selected nodes and we colored in white all the nearby pixels that where at a distance less than the coverage radio. This resulted on an image with overlapped circles, our expectation was that the OF would minimize the circles overlap by selecting nodes that were ideally 2 times the radius of coverage apart.

With the coverage image we calculated the percentage of pixels colored on white from the whole Image. The best solution found by PSO was selected as the subset of nodes.

Using k-means and PSO is an effective and low computationally expensive way to select the subset of nodes. These two algorithms have been used for different purposes on WSN but our contribution is mixing this two algorithm to work together outside of
the WSN on a centralized way using a Host-PC with more resources like energy and processing capacity. Additionally we implemented a UDP communication and different control messages to configure the whole network through Matlab making an important integration of a widely used tool and WSNs.

4.2.3 Topology reconstruction

Now that the CHs selection was done the network topology is reconstructed so we defined that CHs will keep sensing and sending data to the Host PC, while Non-CHs will now enter a low-power mode. From Matlab a UDP Message is sent to all the nodes with their configuration message, for Non-CHs we sent command ”CH0” and for CH command ”CH1”. This was the new behavior defined for Non-CHs:

- The node does not use its sensors.
- The node does not send any messages to the Host.
- The node starts using Radio Duty Cycling (RDC).
- The node changes its RPL mode to Leaf Mode.

4.2.3.1 RDC

As stated before we implemented RDC for Non-CHs, this layer is part of the Data Link layer on the Contiki Network Stack previously shown on Figure 2.1. The RDC layer handles the sleep period of the nodes, this layer is in charge of sensing the medium and deciding when to transmit a package received from upper layers. RDC makes sure that the parent node is awake and ready to receive packages. Different drivers can be implemented on this layer, here we used Contiki-MAC driver for RDC which in its lowest rate of operation (8Hz Channel Check Rate) will keep the radio off for 99% of the time saving a lot of energy. Selecting a low channel check rate saves a lot of energy on the nodes but it delays neighbor discovery making topology construction rather slow. So we turned on RDC on the non-CHs and we kept it off on the CHs. On Contiki exists a function to turn ON or OFF the predefined RDC driver so we used it to modify a node’s behavior.

4.2.3.2 Leaf mode

We implemented RPL leaf mode on the Non-CHs so that these nodes do not have any childs, saving energy and making communication faster (given that these nodes have now
RPL defines two modes of operation: mesh mode and leaf mode RFC[6550], on mesh mode the nodes can be reached through the network and can be selected as Parents and thus routing information through the DAG. On leaf mode the nodes can also be reached through the network but cannot be selected as Parents so they will not route any messages on the DAG.

On Contiki a node can be defined as mesh mode or leaf mode during compilation-time, but it cannot be changed during run-time. We modified the RPL library on Contiki to change during run-time a node from mesh mode to leaf mode. The specification defines some rules for the leaf node that can be found on the RFC[6550] Section 8.5. One of the rules is that a node on leaf mode must advertise a DAGRank of INFINITE_RANK on their DIO messages. As a result we modified RPL libraries on Contiki so that when a node receives the "CH0" command it will start advertising INFINITE_RANK on its DIOs. This will make their current children to select a new parent and leaving the Non-CHs as leaf nodes.

With the techniques previously explained we defined a new mode of operation to effectively save the battery of the non-CHs by implementing different mechanisms to avoid a node’s participation on a network. We also found a new way interesting way to change between RPL mesh and leaf during run-time using Contiki OS with a command send from a Host-PC. This is very useful because with a remote access to the network we can modify during run-time different kinds of behaviors of a WSN.

### 4.2.4 Disjoint Path Construction

After reconstructing topology we stopped RPL execution to save battery on all the nodes and to test the algorithm capacity to self-heal without the help of RPL. To stop RPL the Sink broadcasts a “Stop” message to the IPv6 broadcast address and the message was repeated once by all the nodes that received it for the first time. This resulted on a static DAG that is not modified by RPL.

After stopping RPL we created the Disjoint paths on all the CHs nodes (if possible) so that back-up routes where created to avoid network disconnection in case a parent node died. To create this Disjoint paths we implemented our own protocol based on N-to-1 protocol [13]. To implement the disjoint paths we created a 'disjoint message’, this message has only two fields branch_def, branch_ID. Branch_def is a boolean defining if the disjoint message comes from the sink or not and branch_ID is the branch ID included on the message.
Initially the Sink starts the process by broadcasting the disjoint message \{0,0\}. Every node, that hears this message creates a new branch with his own ID (last octet of his IPv6 address) replacing the fields Branch\_def to 1 and Branch\_ID with his own ID, for example node 0ff3 would broadcast \{1,ff3\}. The nodes that receive a disjoint message with Branch\_def=1 would add the sender node to his disjoint parents table with the information of the sender’s ID and the Branch they are part of. If the sender node is the parent of the receiving node the node will become part of the same branch as its parent and will re-broadcast his new branch ID to all it’s neighbors.

This algorithm has a big change compared to N-to-1 because N-to-1 algorithm stores on each hop the full route to the sink. Given that we executed RPL on fully stateful mode we do not need this overhead and also sending all the IPv6 address of the path would end up on a extremely big message. So we reduced the over-head of N-to-1 and exploited the benefits of having RPL on a WSN to construct easily disjoint paths.

4.2.5 Dead Node Detection and Correction

To detect that a parent is dead or unreachable for other reasons (for example interference) a simple lightweight heartbeat was implemented. After the network has its disjoint paths defined the nodes initializes a heartbeat counter in 0 and starts sending periodically a ”heartbeat request” to its parent, if this node receives an ”heartbeat reply” the counter is reinitialized. After 5 heartbeat requests with no reply the node assumes his parent is dead, the node then selects a new parent from the disjoint parent table and informs the Host-PC the ID of the ”dead node”. With this information Matlabs re-selects the CHs excluding the dead node from the possible CHs, this makes the algorithm proposed here a self-healing algorithm.

We successfully implemented a dynamic multi-layered algorithm that reacts upon a node disconnection and re-selects a new subset of nodes taking into account the recent failure. This results into lifetime extension and we also minimize the coverage loss when a node dies.
Chapter 5

Results and Analysis

Initially we tested the techniques implemented on each layer of the algorithm separately using Cooja simulator. The following section exposes the results of each individual test.

5.1 Individual tests

5.1.1 Topology construction Test

Using the architecture proposed we implemented the RPL network including the Border-Router with its SLIP interface on the Host PC. A capture of the Tunslip6 tool running is shown on Figure 5.1. On this figure the Host-PC creates a new device (interface) called tun0 with the IPv6 address fd00::1/64 and it also adds a link local address fe80::1/64. Then it sends a configuration message to the Border-Router configuring the prefix fd00:: so the Border-Router creates his own IPv6 address using this prefix and also creates a local link address. as seen on the last two lines of the figure the border router creates his local link IPv6 address (fe80::212::4b00:616:f3d) and his global IPv6 address (fd00::212::4b00:616:f3d). Several pings where done to the Border-Router and the other nodes on the network to make sure the DAG was created correctly and the routing protocol was working.
5.1.2 Clustering and CHs Selection over Matlab Tests

An example of an execution of K-means and PSO with 50 randomly positioned nodes on matlab is shown on Figure 5.2, Figure 5.3 and Figure 5.4. On Figure 5.2 we see 30 nodes disposed on random positions with the Sink node being at the center of the image at (50,50). On Figure 5.3 we see the same network now clustered by K-means into 6 different clusters differentiated on the figure with 6 different colors. As the positions were randomized we can see that all the clusters except green cluster has 4 or more nodes. After several iterations the CHs were selected by PSO, on Figure 5.4 the Clustered network now has CHs marked with a red X on top of it.
Figure 5.2: A 6LoWPAN Network displayed on a $100m^2$ area. 30 nodes randomly positioned are shown on Blue. The Sink node on the center of the figure at coordinates $(50,50)$.

Figure 5.3: The same network after K-means algorithm was executed. 6 different Clusters are shown on different colors.
Figure 5.4: After executing PSO 6 nodes (One of each cluster) are selected as a CHs.
The resulting Coverage image of the CHs is shown on 5.5. On this figure the Coverage of the 6 selected nodes and the Sink is about 89.7% while the coverage estimated without PSO was about 99.6% including all the nodes of the network. This means that if we selected only 6 nodes plus the sink the battery of 23 nodes will be extended while keeping the same information based on the assumption that the coverage radius is known.

![Area analyzed = 100m² | Coverage = 89.7024%](image)

**Figure 5.5:** Resulting Coverage of the new Network with the subset of nodes. This coverage includes the coverage of the Sink Node too.

### 5.1.3 RDC Tests

We ran a simple test on cooja with 25 nodes to show that after selecting some nodes as CHs these nodes would have a higher Duty-Cycle compared to the rest of the nodes on the network. On Figure 5.6 is shown the duty Cycle of 25 nodes. The nodes 6, 8, 17 and 20 where selected as CHs, and on the Figure is clear that their Duty Cycle is much Higher than from the rest of the nodes.
5.1.4 Leaf mode test

On Figure 5.7 we have a network with 26 Nodes, and the Topology created by RPL. Here on this Network we tested the Leaf-Node, so we selected node 26 to change his state from mesh to leaf. Now all the DIOs sent from this node would report INFINITE_RANK. Within a few moment all the nodes that had node 26 as his parent left him and selected a new parent as shown on Figure 5.8. Nodes, 15, 20 and 22 left the node and updated the infinite rank on their rpl parents table.
Figure 5.7: A network with 25 Nodes running RPL

Figure 5.8: The same network shown on Figure 5.7 but now with node 26 as a Leaf node
5.1.5 Test of dead node detection and correction

To test the dead node detection and the correction by selecting a new parent we first Stopped RPL and then we started the simple Heartbeat on only one node. On Figure 5.9 we see the resulting Topology of a WSN conformed by 25 nodes, all the nodes are running RPL on normal conditions. Then we started RPL Stop command so the nodes now can not change the network topology with RPL.

![Figure 5.9: A WSN of 25 nodes running RPL algorithm.](image)

We can see on figure 5.10 the historical RPL routing metric through time. We selected node 16 to start the simple heartbeat so now node 16 began sending requests to his father (node 7) and this node replied. Then we deleted node 7 and we can notice on Figure 5.10 that after second 19:00:50 several nodes stopped sending information of their metric. The metric still changes because DIO messages are still sent so the metric might decrease or increase just a little bit but nodes will not select new parents and Topology will not change. On Figure 5.11 we see that from all the childs of node 7 only node 16 (That was using the heartbeat) changes his parent to node 6 a new disjoint parent but the other childs of node 7 (nodes: 10,13,17,18 and 23) still have the same nonexistent node 7 as parent.
Figure 5.10: RPL Historic Routing Metric on the WSN.

Figure 5.11: New Topology after RPL is stopped and node 16 reselects a new parent thanks to the heartbeat he was running.
On Figure 5.12 is shown the received packets per node, all the nodes that did not have node 7 as parent sent 7 to 8 packages while all the childs from node 7 excepting node 16 sent just 2 packages maximum due to the deletion of their parent node 7. Here we also see that node 16 has sent only 6 packages, he lost some while he was trying to reconnect to the network by selecting a new parent but he eventually changed parent and continued to send messages normally.

![Fig 5.12](image)

**Figure 5.12:** Received packets per node. Here only node 16 from all of the node’s 7 childs recovered and lost a few packets but started to work normally. The other childs of node 7 never recovered because heartbeat algorithm was not active

### 5.2 Energy Tests

To check the performance of the algorithm we made several tests over a simulated WSN using Cooja Simulator and we also performed tests over a small real WSN. We evaluated the network in terms of energy consumption to verify how much was the current consumption on CHs and non-CHs.
5.3 Cooja Tests

To check the performance in terms of Energy on the network we implemented a WSN on the Cooja simulator, here we initially tested all the layers from the algorithm that did not involve Matlab by manually selecting the CHs to check RPL metrics, packets received and others using Collect-view tool from Cooja. The topology implemented is shown on Figure 5.13 where the node 1 is the Sink node and the other 24 nodes are normal nodes that send the information they gather through their sensors. Using the sky motes on Cooja we tested the algorithm selecting the nodes 6, 8, 17 and 20 manually as CHs.

To calculate on Cooja the energy consumption we use the power profile Energest [17] in Contiki OS to record all the energy consumption on each device. Energest tracks the power state to estimate the energy consumption from the activities: Rx (packet receptions), Tx (packet transmissions), CPU (CPU in active mode), LPM (Low Power Mode or standby mode on the CPU). Using the formula 5.1 on each activity the power consumption is calculated.

\[
EC = \frac{\text{Energest Value} \times \text{Current} \times \text{Voltage}}{\text{Rtimer} \times \text{Runtime}}
\]  \quad (5.1)

Figure 5.13: Topology used for the test on Cooja simulator
Where EC is the energy consumed, Energgest_value is the value given by the energest profile on the selected activity, current is the current consumed, voltage is the supply voltage, Rtimer_sec is the number of ticks per second on the rtimer, runtime is the run time between two Energgest track points.

Adding the current consumption on each activity we estimated the overall energy consumption. On Figure 5.14 we can see the Historical consumption on all the devices on the network. On this Figure it can be seen that the 4 CHs consume approximately 60mW while working on normal operation while the Non-CHs would almost always use less than 5mW. With this average consumptions we can estimate the duration of a charged battery on a CH node (sky mote based) and a Non-CH node.

![Figure 5.14: Historical power consumption on the sky motes on Cooja](image)

After completing this test we then proceeded to perform the energy test consumption over real nodes and we also tested the performance of the network on a small deployed WSN.
5.3.1 Real Node Tests

To evaluate the Energy consumption on the real nodes we used the Re-mote designed by Zolertia. We recorded the measurements of current consumption of two nodes using a 3.7V Li-po battery on each node. One of the nodes worked always as a CH and its average consumption was 28mA approximately as seen on figure 5.15 with some peaks of 32 mA due to message transmission or other processes. On the second node we made the same test but during the test we changed the node operation state from CH to Non-CH so its consumption changed drastically as seen on Figure 5.16. Here it went from 28mA on CH mode to almost 5mA on Non-CH mode.

![Current Consumption Normal operation](image)

**Figure 5.15:** Current consumption of a CH node

With the average consumption we estimated the duration of the node with a fully charged battery on a CH node and a Non-CH node. The batteries used had a capacity of 1300mAh so for a CH node the estimated duration of the battery would be $\frac{1300\text{mAh}}{28\text{mA}} \approx 46h$ and for a Non-CH node the estimated duration of the battery would be $\frac{1300\text{mAh}}{5\text{mA}} \approx 260h$, this would make the Non-CHs nodes last almost 6 times more than the CHs nodes.
5.3.1.1 7 Node deployment Network

Additionally we implemented a simple test on the Department of electronics engineering on the faculty of Engineering at the Pontificia Universidad Javeriana. 7 nodes were deployed on this floor and all the nodes sent the instantaneous voltage on the battery on a UDP message to the host PC. On Figure 5.17 is shown a map where the nodes where deployed. On this map the nodes are identified by the last octets of their IPv6 address.

With a Web server application running on the Border router we could read the neighbors of the Border routes (specified by the local link- address fe80::1) and the routes to the nodes on the network. Using a Google Chrome Browser we typed the address of the Border-Router and we obtained the result shown on Figure 5.18.

On the web browser we could see that 5 of these nodes were neighbors of the sink (1 Hop) and on the routes list we found out node that the node with the address fd00::212:4b00:616:ff3 is a child of fd00::212:4b00:615:ab2f (2 Hops to the Sink).

For the Tests the following nodes were selected as Non-CH:

- fd00::212:4b00:616:ff3
- fd00::212:4b00:616:f3f
- fd00::212:4b00:616:f3d
Initially the batteries were almost charged to its maximum. The Maximum voltage level for our batteries goes up to almost 4 V and rapidly discharges to stay on 3.7 V. The battery sensor on the nodes can measure maximum 3.5 V so we began the test and waited for the batteries to discharge while gathering data from every sensor of the nodes and sending them every 30 seconds including the voltage level of their own battery. After more than 30h since the nodes started reporting the battery level we obtained the Average Voltages per hour shown on Figure 5.19. The nodes fd00::212:4b00:615:aa72 and fd00::212:4b00:615:ab05 discharged their batteries completely. We can see from the figure that while the CHs nodes consumed their batteries faster, the Non-CHs stayed with their battery level over 3.5V

With this test we confirm the calculations we did with the average current consumption of the duration of the batteries where right. So CHs would last 40h approximately while non CHs would stay 6 times longer. If the Algorithm was not used all the network would die on a matter of 48h maximum, while with the algorithm implemented the duration could extend up to 260h maximum when the last node dies.
Figures 5.18: Web Browser application. Routes and Neighbors of the Border Router are sent to the Browser on HTTP format.

```
Neighbors
fe80::212:4b00:616:f3f  
fe80::212:4b00:616:ab2f  
fe80::212:4b00:616:ab65  
fe80::212:4b00:616:aa72  
fe80::212:4b00:616:f3d

Routes
fd00::212:4b00:616:f3f/128 (via fe80::212:4b00:616:f3f) 554s  
fd00::212:4b00:616:ab2f/128 (via fe80::212:4b00:616:ab2f) 1708s  
fd00::212:4b00:616:f3d/128 (via fe80::212:4b00:616:f3d) 552s  
fd00::212:4b00:616:ff3/128 (via fe80::212:4b00:616:ab2f) 1747s  
fd00::212:4b00:616:ab65/128 (via fe80::212:4b00:616:ab65) 1522s  
fd00::212:4b00:616:aa72/128 (via fe80::212:4b00:616:aa72) 556s
```

Figure 5.19: Batteries of the nodes after more than 30h of test. The CH nodes depleted their batteries very fast while Non-CHs still continued to operate normally, saving their energy.
5.3.2 MQTT - Tests

The nodes sent all the messages to the Host PC trough the network and the Border Router and the MQTT server application running on the PC published on the MQTT broker iot.eclipse.org. The output of the MQTT-Server application can be seen on Figure 5.20. On this figure we can see on line 1 the date and time of the incoming message and the IPv6 address of the node publishing the information (fd00::212:4b00:616:f3f) and the UDP port (8765) used. Next on lines 3 and 4 is the message sent over IPv6 from the node with the information of: the id(last 2 octets of the IPv6 address on decimal notation), the counter (number of package), core_temp (temperature on the core), ADC1, ADC2 and ADC3 measurements, and the voltage level of the battery on the node (on mV). From lines 6 to 37 the application prints the variables to be published on the MQTT-Broker. And finally on line 38 is the output of the application when the message is published; at the end of this line is the topic to be published on (0f3f) this are the last 2 octets of the IPv6 address of the node that sent the message. So now every mote will publish the data collected to their own topic which is the last 2 octets of its IPv6 address.
Figure 5.20: MQTT server application running on the Host PC

All the information now can be obtained on another device running a MQTT-Client that must connect to the host name of the broker: iot.eclipse.org using the port 1883. Once it is connected it must subscribe to the desired topic, all the nodes are publishing on the topic: DACA/info/(last 2 octets of the IPv6 address of the desired node). For example subscribing to the TOPIC: DACA/info/0f3f will send us a message every time the node with the address fd00::212:4b00:616:f3f publishes something like it is shown on figure 5.20. To subscribe to the publications of all the nodes on the network the TOPIC: DACA/info/# where the symbol # is the Multi Level wildcard, this wildcard covers an arbitrary number of topic levels.

Two MQTT-Clients subscribed to the topic DACA/info/# running on a Windows-PC and an Android smart-phone where use. The applications MQTTlens on Windows OS and MyMQTT on Android OS. A screen capture of MyMQTT is shown on Figure 5.21.
the message sent from the MQTT-Server shown on figure 5.20 is highlighted on a blue square. The capture of MQTTlens running on windows is shown on Figure 5.22 here the same message received on Android is highlighted on red square on the Figure.

Figure 5.21: MyMQTT App susbscribed to the topic DACA/info/# showing two messages of the nodes f3f and aace, a total of 870 messages received from the 6LoWPAN network
**Figure 5.22:** MQTTlens application running on Windows subscribed to the topic DACA/info/#
Chapter 6

Conclusion

In this paper we presented a multi-layered algorithm for self-healing on WSN. Here we can see that on a very high dense network we can optimally select a subset of nodes using K-means and PSO saving the batteries of many nodes. We implemented an algorithm that successfully controls topology on an efficient way while making it also fault-tolerant and self-healing. We presented different contributions such as: the integration of two well known techniques such as PSO and k-means working together to select the best subset of nodes, configuring a WSN from a Host-PC from a widely used tool such as Matalb allowing it to interact and set a desired behavior of a WSN. We also presented an algorithm that can dynamically readjust and self-heal after a specific fault (node disconnection) to minimize the loss of coverage while maximizing the network lifetime.

We found out that the battery saved and the network lifetime depends directly on the number of nodes and the coverage area assumed on each node. If the area of interest could be covered by a few nodes then most of the other nodes can go into low power mode extending network lifetime.

The low power mode can be implemented in many ways, for example the CC2538 chip from texas instruments has incorporated a low power mode that can shut down its chip to consume 170nA and wake up periodically. We did not used this mode because the node restarts every time and all the constants and variables stored on the node were completely erased, making difficult to construct the disjoint paths and keeping record of the state of the network.

One of the goals of this work was to expose the usefulness of IPv6 on WSN supported on enabling technologies like 6LoWPAN and RPL. This could be the base from where IoT developers can implement their own applications and bring solutions to specific problems with easy IPv6 addressing on a 6LoWPAN network.
Although it is out of the scope of this paper, implementing MQTT-clients directly on the nodes without using the MQTT-Server on the Host PC can enable them to publish and subscribe on their own to different topics of a MQTT-Broker. This will allow the node to publish information but also to be read publications from another client containing configurations for the nodes. For example a user can publish a LED ON message that will turn ON a LED of the node. Also on platforms like IFTTT using the messages sent from the nodes the platform can react to the received data and triggering actions; for example on a crop the temperature starts to go up and the soil moisture decreases, IFFFT checks the messages sent from the WSN triggering the sprinklers to moisture the crops and maintain a favorable temperature over the crops without the need of human intervention making automation of several process a really easy thing.

As a future work a more efficient mechanism could be implemented to save the battery from the nodes selected as Non-CHs; this kind of work was out of the scope of the work and we focused on selecting what nodes should go to low power modes and using RDC as ”low power mode“ for the selected nodes to check the behavior of the network. Also as a future work we would like to implement a test bench to have higher precision on the measurements done and check parameters like, latency, packets lost and others. The need for a test bench to perform some tests will might throw interesting and relevant data.
Bibliography


