Low Power WiFi: A study on power consumption for Internet of Things

---

Master's Thesis

Defense date: February 2, 2015

Ugaitz Amozarrain Perez

Supervisor:
Jose Maria Barceló Ordinas
Departament d'Arquitectura de Computadors

Co-supervisor:
Jorge García Vidal
Departament d'Arquitectura de Computadors

Master in Innovation and Research in Informatics
Computer Networks and Distributed Systems

Facultat d'Informàtica de Barcelona (FIB)
Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

Abstract

The trend of *Internet of Thing* has opened new possibilities for wireless networks. One of them is the creation of *Wireless Sensor Networks*, this kind of network works with a lot of sensor nodes measuring several physical phenomena, such as pollution, wind speed, noise, etc., and then sends the data to a central location in order to process it and be able to obtain information from it.

The creation of such a network involves the use of autonomous devices that must work most of the time without any kind of wire, so all transmissions must be wireless, and the device must function with a battery. But we also want the sensor nodes to be able to keep sending information for as long as possible This is the reason behind the development of different low-power transmission mechanisms, such as 802.15.4 or Low-Power WiFi.

One of the main and clearer benefits of using *Low-Power WiFi* is that it can seamlessly connect to existing network infrastructures and send data through common protocols, *HTTP* for example. The power requirements of Low-Power WiFi is also orders of magnitude below normal WiFi, putting it around the power requirements of devices implementing 802.15.4, but without the inconvenience of adding another network device to translate from one standard to another, and connect to the Internet.

In this document I describe the tests I did in order to analyze the feasibility of a Low-Power WiFi device in a wireless sensor network. The tests involved measuring the power the module needed to function in different situations, the range and reliability of the connection, how easily was the chip configured and how it worked.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

Contents

This section gives an introduction to the work done at the *Facultat d'Informàtica de Barcelona (FIB)* [10], for the *CompNet Research Group* [6].

First, a little bit of background is given explaining the project, for this a brief explanation on *Internet of Things* is also given. After this, I describe the goals that I want to achieve with this work and we list the phases I followed to achieve those goals. Finally, a general description of the thesis is given, summarizing what is written in each section.

1.1 Internet of Things

The *Internet of Things(IoT)* is a new paradigm that is increasing in popularity. It is defined as *"an interconnection of uniquely identifiable embedded computing devices within the existing Internet infrastructure, offering advanced connectivity of devices, systems, and services that goes beyond machine-to-machine communications and covers a variety of protocols, domains, and applications"* [32].

The main idea of IoT has also evolved since its creation, from offering status information to automating whole systems. The evolution of hardware has also helped in the expansion of this paradigm, it is cheaper, consumes less power, and nowadays nearly everybody has a mobile phone with capability to use a whole lot of connection types, so people can interact with objects anywhere they are.

In order to implement an IoT project several technologies have to work together as explained by Atzori et al. [29]. First there are the *identification, sensing and communication technologies*, these are the basis of IoT, they are the building blocks of any project. They are small devices with very low or non-existent power consumption, little processing power and limited communication capabilities. These devices usually communicate between them, in a *collaborative* way, and all outside connections are done through a sink node. Above them is the *middle-ware*, which usually functions as a software layer to hide the *"things"* from the application, this offers an abstraction layer that makes it easier for the programmer to separate concerns. Lastly there are the *applications*, an application uses the information provided by the middle-ware to create a representation for the user to use, Figure 1.1 shows several application domains that IoT technologies have.

1.2 CommSensum

There has been an increasing concern on air quality in the last few years. In order to respond to this worry the governments have begun installing air quality sensors on several cities [1, 3, 4]. These air quality measurement systems offer free updated information on air quality through all Spain. But these kind of systems have one big drawback that the ecologist groups warn against, they are controlled by the same government that *maybe* does not want the citizens to know that the air quality is really bad, since there have already been several cases of incorrect information being obtained by them [5, 7, 12]. This does not mean that the entities in charge of the systems are lying about the information they obtain, the error can be as simple as not analyzing well enough where to put the measurement stations and due to several atmospheric conditions the measurement stations are not reading the correct measurement.

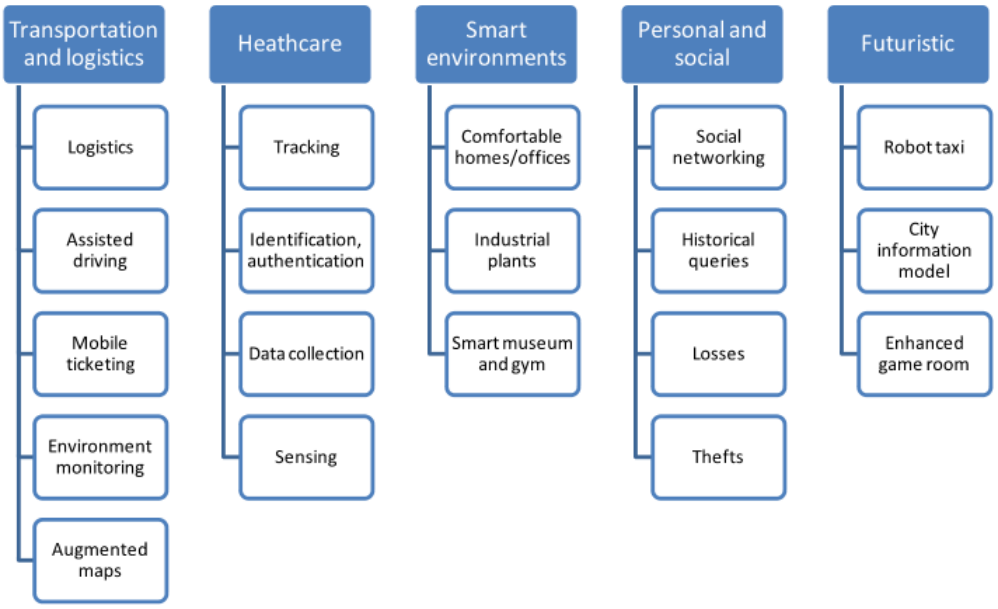


Figure 1.1: IoT application domains [29]

One way that the ecologist groups believe that will be able to solve this issue is to allow people to connect to the system and send their own data, creating a *Community Sensing Platform*, in which people can join as easily as bigger entities, as governments or companies. So, this platform will have all kinds of information from different sources and the users will be able to differentiate the source of the information they are seeing and compare the different sources.

This is one of the factors for the creation of *CommSensum*, a Community Sensing Platform based on *Open Linked Data* [21]. The main idea of the project is the monitoring of the air quality in different geographic areas, giving users the power to connect their own measuring devices to the platform and showing information obtained from all users, and even integrating that data with the one provided by government entities when that information is in public databases.

The project is already online [22], though it is just for experimentation while the different parts are being developed, and a mobile app exists [2] showing the already available information from sensors online. Figure 1.2 shows a brief overview of the whole platform. Having this structure we can observe that the platform follows the Internet of Things idea of having interconnected smart devices in this case sensor nodes.

1.3 Goals

There are different communication systems used on wireless sensor networks, some of them could be 802.15.4, radio, 3G or WiFi. Some of these communication protocols are designed around the intercommunication of embedded devices and focuses on being Low-Power, such as 802.15.4. Though the problem of using such a method is that even if the nodes themselves are able to communicate easily, when trying to connect this network to the Internet we need some type of adapter that is able to communicate with both the

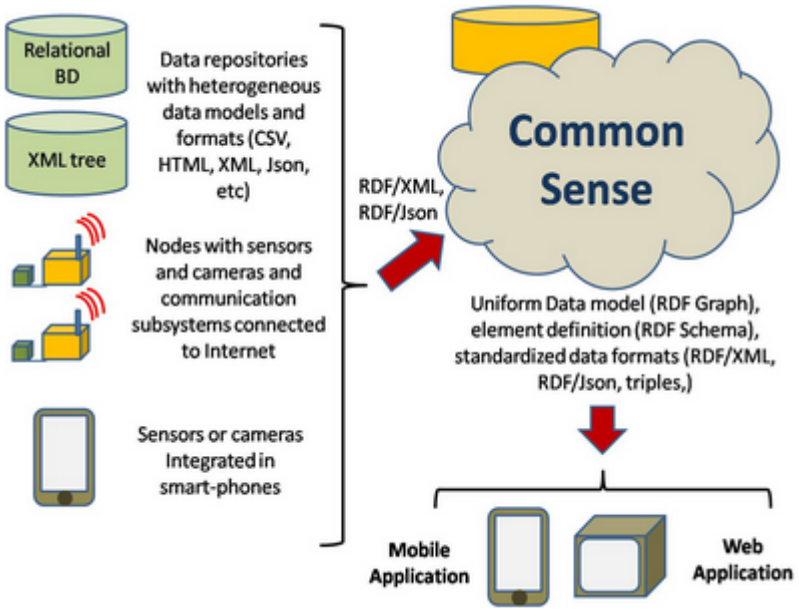


Figure 1.2: CommSensum platform overview
Taken from CommSensum project webpage [21]

sensors and the Internet.

When trying to develop a Community Sensing Platform we have to make it as easy as possible to have people join the community, and having them buy expensive hardware is not attractive. Nowadays everybody has a WiFi access point at home, and if we could use this same device to connect the sensors to the Internet it will make it much easier to install a sensor node. But WiFi is not a Low-Power specification, and this is where Low-Power WiFi becomes relevant.

Low-Power WiFi is a variant of the 802.11 specification that focuses on reducing the power consumption, while still being able to communicate with normal WiFi devices, so a device with a Low-Power WiFi will be able to connect to the access points people already have without the need of any interfacing hardware. In order to see how it works and if it is as good as the manufacturers claim we have the following objectives in this project:

1. Analyze the viability of using LowPower WiFi technologies to build a Community Sensing Platform.
2. Analyze power consumption of LowPower WiFi devices.
3. Analyze the timing constraints of LowPower WiFi devices.
4. Analyze the performance of LowPower WiFi devices.

1.4 Phases

1. Project management.

Weekly meetings took place in order to share the work done by each member of the laboratory and solve possible doubts on how to follow with the investigation. They where held on noon on Wednesdays.

2. Initial research of the state-of-the-art.

   An initial research was done to learn how IoT networks work, the technologies used and the techniques for reducing the power consumption.

3. Experimenting with LowPower WiFi.

   Several experiments were made in order to test if LowPower WiFi devices worked as described by the manufacturer and to analyze its viability in a future implementation.

4. Thesis writing and presentation.

   The thesis was written through the investigation though most of it has been written at the end. The presentation will take place in Facultat d'Informàtica de Barcelona in February 2015.

1.5 Chapter description

- Chapter 1 is this introduction.
- Chapter 2 gives a brief overview on technologies and techniques that are used on Internet of Things sensor networks in order to lower the power consumption of the devices used.
- Chapter 3 describes the hardware used for this project and how it is used.
- Chapter 4 explains the experiments done to test the feasibility of using Low-Power WiFi.
- Finally Chapter 5 gives the conclusions obtained from the experiments and some lines for future work.

Chapter 2

State of the Art

Contents

2.1 Introduction

One of the consequences of the Internet of Things is the creation of wireless sensor networks (WSN). This allows the interconnection of several geographically distributed sensing devices to be able to connect between them and send their data in a easy way, this devices are used for monitoring physical phenomena such as temperature, humidity, gases and so on [27]. The main problem of having a sensor somewhere is how to obtain data from it, will it store the information in a memory and somebody has to go and dump the data periodically to other device? Will the sensor itself send the data to a server? How will it send the data, wired or wireless? The main benefit of wireless transmission is that there is no need to install cables so the sensors will be able to be deployed anywhere, but being wireless also means that it has to rely on a battery to sustain itself. Figure 2.1 shows a possible topology for a WSN, the sensor nodes are able to communicate between them, and all the data that needs to go outside the network is sent through a sink, this means that the sensors can use low power wireless technologies to "talk" between them and the sink is the only one that will need a connection outside.

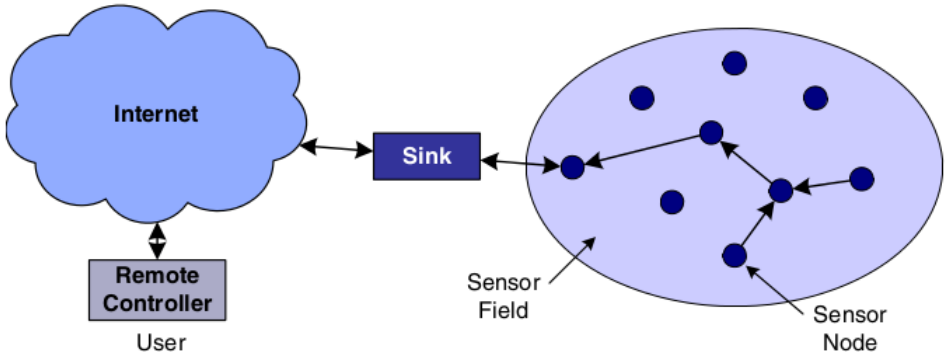


Figure 2.1: Wireless Sensor Network representation [28]

2.2 Lowering power consumption

Internet of Things devices were simply *RFID* chips [29], these chips can be used as passive transponders or if they need power they are able to harvest the energy they require to function from the reader itself, through electromagnetic induction. Thus eliminating the need for an external power source. But the inconvenience that this kind of chips have is that they can only answer to questions, they are not able to process data themselves autonomously. Last years there has been an increase in the operations that these *smart objects* have to perform [33], increasing their computational power and with it introducing the need for an external power source.

Several technologies have been developed in order to try to lower the power consumption of smart objects that have to function for years with the same battery [28]. In this document I will mention the two most widely used wireless communication methods [37]: IEEE 802.15.4 and IEEE 802.11 (WiFi).

2.2.1 IEEE 802.15.4

IEEE 802.15.4 is a standard specifying the physical layer and media access control for low-power and low-rate wireless personal area networks. It has a maximum data rate of 250,000 bits/s and a maximum power output of 1 mW. Having such low power requirement also means that its range is in the tens of meters, but being cheap to make and using so little power have made this technology an interesting choice for interconnecting IoT objects.

The standard specifies what happens in the *physical* and *media access control (MAC)* layers. IEEE 802.15.4 operates in three distinct radio frequency bands, depending on the local regulations of different parts of the world: 902–928 MHz on the United States, 868–868.8 MHz on Europe and 2400–2483.5 MHz on the rest of the world. The last band overlaps with the IEEE 802.11 standard and since IEEE 802.11 operates with more power it will cause high interference with the devices implementing IEEE 802.15.4.

Figure 2.2 shows the power consumption of a device that uses this technology to communicate. The transmission power can be configured by software depending on the needs of the network. In idle mode the device has very little power draw, however the device is not able to listen in this mode.

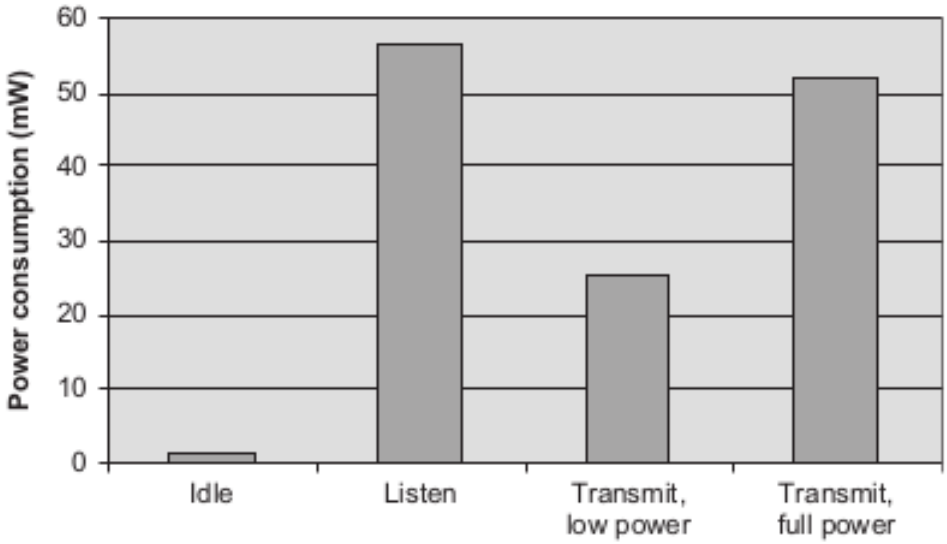


Figure 2.2: Power consumption of the CC2420 IEEE 802.15.4 radio transceiver [37]

2.2.2 IEEE 802.11

This standard, also known by the WiFi brand was designed for high-speed, short-range communications, mainly for laptops and general purpose PCs. The maximum speed varies depending the version of the standard, the latest being 802.11n and having a maximum data rate of 600 Mbits/s.

IEEE 802.11 is a widely used standard, and this makes it a compelling choice when building a network of smart object since there is an infrastructure already built that supports this standard, but it was designed for laptops or PCs, where the power requirements

are different the standard was deemed too power hungry for battery-powered smart objects. This changed when manufacturers started developing low-power 802.11 circuits. Besides reducing the power requirements for transmitting or listening they greatly reduce the sleep mode power consumption, making it very convenient for IoT applications since these devices are mostly in the sleep state. Low-power 802.11 also have a standby mode where the power consumption is on the level of μW. The comparison between conventional and Low-Power WiFi can be seen on Figure 2.3

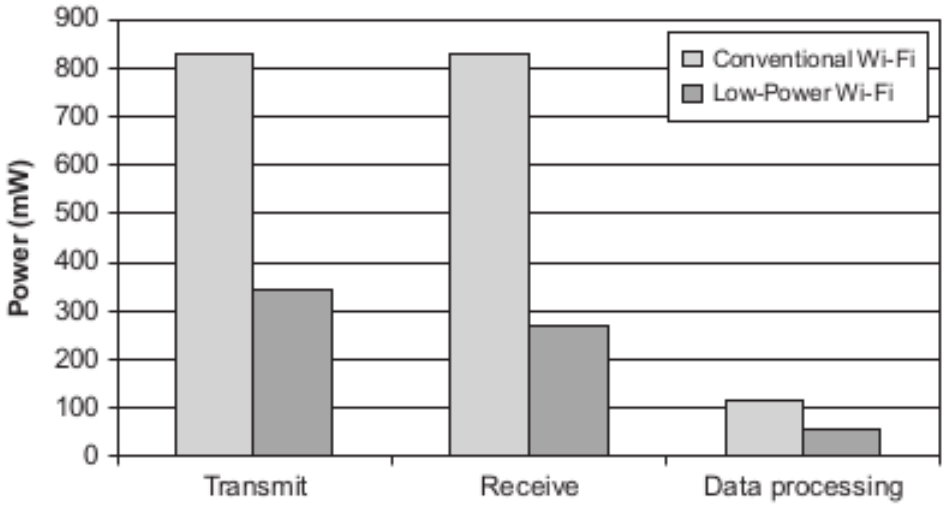


Figure 2.3: Comparison of power consumption for conventional 802.11 and low-power 802.11 [37]

2.3 Energy conservation schemes

Depending on the logical structure the sensor deployment follows we can choose between different energy conservation schemes. The most common architecture is the one shown on Figure 2.1. In this example not all sensors are able to connect to the sink directly so data must be transmitted between nodes until it reaches the sink. In an ideal world the sensors will have a continuous, infinite source of energy, in this case all nodes will be always awake and common mesh network protocols will be deployed. In reality the sensors have batteries and they must waste as little power as possible by sleeping most of the time. Since nodes are sleeping to conserve energy we cannot always use the same routes when sending a message, thus different protocols must be created.

Lots of work has gone into solving this problem and several approaches have been suggested and found to be successful. The different approaches can be classified into three groups [28]:

Duty cycling: The idea here is that there is no need for all nodes to have their radio on all the time. Nodes will turn their radio off when there is no network activity and wake up depending on the chosen schema. The power consumption of the radio is greatly reduced if the nodes use this approach, though in sensors where the measurement need a lot of power it will not work as well.

Data driven: This approach tackles the issue of when to take a measurement and how to send less data to the network, also reducing the time the radio and the sensors are on.

Mobility-based: It tries to handle the power consumption when the node is in movement following predictable or totally random patterns.

Though there are three groups, this does not mean that they are mutually exclusive, a sensor node can implement as many approaches as necessary from all the groups, this way the total life of the sensor's battery is increased.

2.4 Related work

The use of smart objects and the Internet of things idea has increased the interest if Low-Power alternatives to normal hardware, and the Low-Power alternative to WiFi is no exception. There have been several analysis of the power consumption of a Low-Power WiFi device in a real environment and compare it with other existing technologies built with the Low-Power idea in mind [36], such as 6LoWPAN over IEEE 802.15.4 based networks, and they conclude that in terms of power consumption while 6LoWPAN would be better for small packages, the higher throughput and bigger packet size of Low-Power WiFi will give it an advantage for larger transmissions. Others try to analyze not just the power consumption, but also if there is any overhead when using such a device [31].

Some researchers also tried to test the Low-Power WiFi devices in environments where normal WiFi is used and using common connection protocols such as HTTP [34], having applications run as well as with normal WiFi. And some go even further and try to use it instead of normal WiFi in an indoors environment [35].

Chapter 3

Hardware

Contents

In this chapter I will explain the hardware that is used in the project, and how to use it. First I will describe the Low-Power WiFi chip used, *Roving Networks RN-171*. Then the boards used to test this chip: the *Waspmote*, a board sold by Libelium and the *Raspberry Pi*. Finally I will describe the devices used on the testing environment.

3.1 RN-171 Low-Power WiFi chip

The RN-171 is a Low-Power WiFi chip created by Roving Networks [15]. This module has embedded a TCP/IP stack, cryptographic accelerator and power management subsystem, making it a standalone embedded wireless LAN access device. It can be connected to other devices or function independently by configuring it to send data when receiving a signal from a sensor from the different inputs it has.

In this project the chip was used to give WiFi connectivity to the sensor nodes.

It follows IEEE 802.11 b/g standards, it can also use WEP, WPA and WPA2 security, so we are able to connect to normal WiFi network using this chip, reaching speeds of 1-11 Mbps for 802.11b and 6-54 Mbps for 802.11g. It has a 40 mA current consumption while listening, a maximum of 120 mA while transmitting (transmission power is configurable), and 4 μA while sleeping.

One of the main benefits of using this chip is that it has built in network applications for TCP, UDP, HTTP and FTP for sending and receiving data, besides DHCP, DNS, ICMP and ARP. This way the developer does not need a OS with those applications and he can just concentrate on what it is transmitting.

All the configuration is made trough a serial interface to the chip, once it is configured the chip is able to store the configuration in a flash memory and reload it even after it has been shut down. The transmission and retrieval of data is also done from the serial connection.

Listing 3.1 shows the commands required to setup a simple HTTP client on the RN-171 chip [25], these commands have to be sent through the serial connection to the chip. The first line is the command that tells the chip to enter command mode so that the following lines will be processed as commands and not as text to send to an already open connection. This first command has to be sent without line feed or carriage return, however all the following commands have to be sent with a carriage return. The next set of commands will configure the chip to use the HTTP client application set up a WiFi connection and join to an specific network. The next set of commands configures the chip to connect to a server with a predefined request, and the last line opens the HTTP connection.

```
1 > $$$    - With no carriage return to enter command mode
2
3 > set ip dhcp 1                - Configure DHCP
```

```
> set ip proto 18                 - Set HTTP mode
> set wlan join 0                 - Set join mode manual
> set wlan phrase PASSWORD        - Set WLAN password for WPA
> join SSID                       - Join SSID network

> set ip host IP                  - Set the remote IP address
> set ip remote PORT              - Configure the remote port
> set comm remote GET$/....       - Define what is sent
> set opt format 1                - Normal mode for HTTP


> open                            - Open connection
```

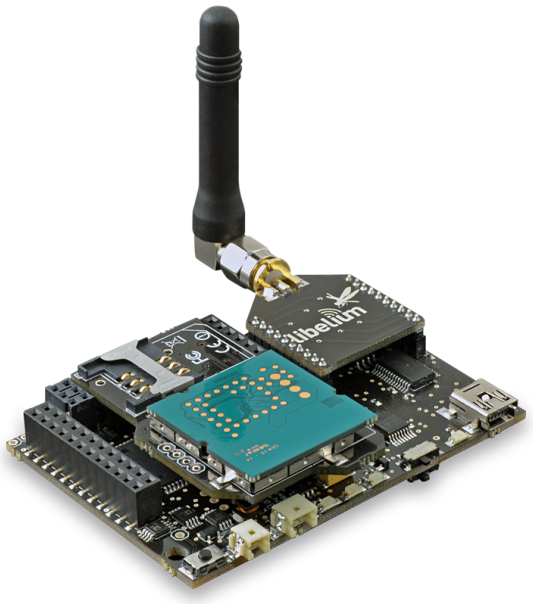
Listing 3.1: RN-171 configuration example

3.2 Sensor nodes

Two completely different single board computers have been used for the tests. The Waspmote is a specialized sensor node while the Raspberry Pi is more similar to a general purpose PC. Their computational capabilities and therefore the power consumption is completely different between both of them, while the Waspmote was designed with the idea of functioning with a battery and being completely low-power no such limitations where imposed on the Raspberry Pi.

3.2.1 Waspmote

The Waspmote is a ultra low-power *sensor platform* by a company called Libelium. Besides having really low power requirements the node is designed to be modular, this way the device is able to easily support a good quantity of sensors and communication technologies, the company also offers an API that interfaces to all the modules they provide for easy programming. This device is widely used for different projects where sensors will help, from smart agriculture project to improve wine production in a vineyard [16] to measuring water quality for better water cycle management [19], and even more [17, 18].



Hardware specifications of the Waspmote can be seen on Table 3.1, and an overview of the connections the board has on Figure 3.1. As it can be seen on the diagram the sensor modules are built atop the base board, all the sensors are built and sold by Libelium,

General data	
Microcontroller	ATmega1281
Frequency	14.7456 MHz
SRAM	8 KB
EEPROM	4 KB
FLASH	128 KB
SD Card	2 GB
Weight	20 gr
Dimensions	73.5 x 51 x 13 mm

Power consuption	
ON	15 mA
Sleep	0.055 mA
Deep sleep	0.055 mA
Hibernate	0.0007 mA

Table 3.1: Waspmote specifications

though it is possible to programatically control the inputs/outputs the board has in order to communicate with custom build devices.

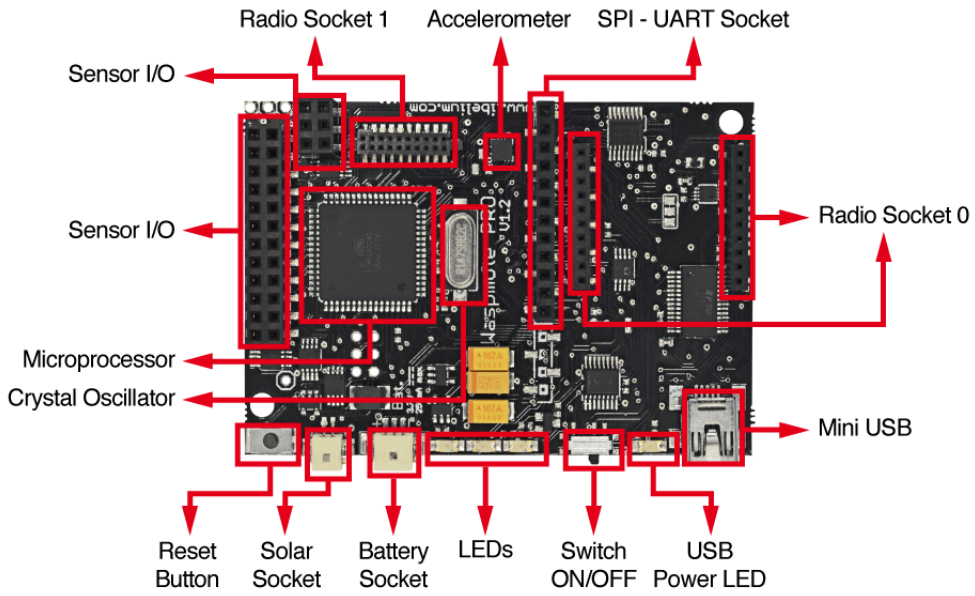


Figure 3.1: Waspmote overview

Power management

As shown on Table 3.1 there are four states where the Waspmote is using power [24].

ON: This is the normal operational mode. The main program is running, and the modules receive power if they are on.

Sleep: Main program is stopped. The node can be woken up by asynchronous interruptions made by the modules, or by the watchdog. The programmable duration interval of this state is between 32ms and 8s. In this mode all variables and log values are stored so that when the Waspmote wakes up the program continues its execution as normal.

Deep sleep: Its the same state as the sleep mode, but instead of waking up by the watchdog it uses the Real Time Clock (RTC). Thanks to that the duration of this state can be defined in seconds, minutes, hours and days.

Hibernate: The main program is stopped, along with all the modules connected to the Waspmote. The only way to wake it up is using the RTC interrupt that as in the deep sleep state can be defined in seconds, minutes, hours and days. It is the state where less power is used since it is just powering the RTC. This state does not keep any variables in the program stack, when the Waspmote is woken up from this state the micro-controller is reset as if it were just turning on, so the program has to run the setup phase again. The sensor node is protected against going to this state by accident with a switch that must be turned off once the program starts and the Waspmote must be restarted for the program to access this mode correctly.

The modules connected to the Waspmote also have four operation modes, the power consumption on each of them depends on the purpose and the manufacturer of the module:

ON: Normal mode.

Sleep: Some functions are stopped, it operates differently on each module and depends on the manufacturer.

Hibernate: All functions are stopped and operates asynchronously, this mode also changes its behavior depending on the module manufacturer.

OFF: Using digital switches controlled by the micro-controller the module is switched off. The implementation is done by Libelium, and does not depend on the manufacturer as previous states.

Development

The programs that are run on the Waspmote are separated into two main components as can be seen on Listing 3.2. Those components are referenced as two functions, `setup()` and `loop()`. When the Waspmote is turned on or awakes from the hibernate state it will execute the `setup()` function and then it will keep on calling the `loop()` function until told otherwise by turning it off or sleeping. The important part is that `setup()` is just executed once, while `loop()` as its name implies is executed continuously. The programming language it uses is *C++*, though without the standard libraries that the language provides.

```
1 void setup()
2 {
3   USB.ON();
4 }
5
6 void loop()
7 {
8   USB.print(F("Battery␣Level:␣"));
9   USB.print(PWR.getBatteryLevel(),DEC);
```

```
10    USB.print(F("␣%"));
11
12    delay(5000);
13  }
```

Listing 3.2: Waspmote program example

All the programming is done through the IDE provided by Libelium. The IDE is preloaded with the programming API for the Waspmote and contains useful example programs. It is also used to upload the program files into the Waspmote and debug the micro-controller through a serial interface monitor.

One of the main problems of programming for this device is the space requirement of the program. We are using an embedded device and as such the 8KB of RAM it has, that is shared between the program itself and the variables declared, makes it difficult to program things that might require a bit of memory such as security schemes.

3.2.2 Raspberry Pi

The Raspberry Pi is a small factor single-boar computer developed by the Raspberry Pi Foundation [14] originally with the idea of introducing computer science to children in school [30].

There are four board configurations currently, called the *A*, *A+*, *B* and *B+*. The "+" versions are boards with improvements over the previous versions and while the *A* series is intended for low power usage the *B* series approaches a simple general purpose PC.

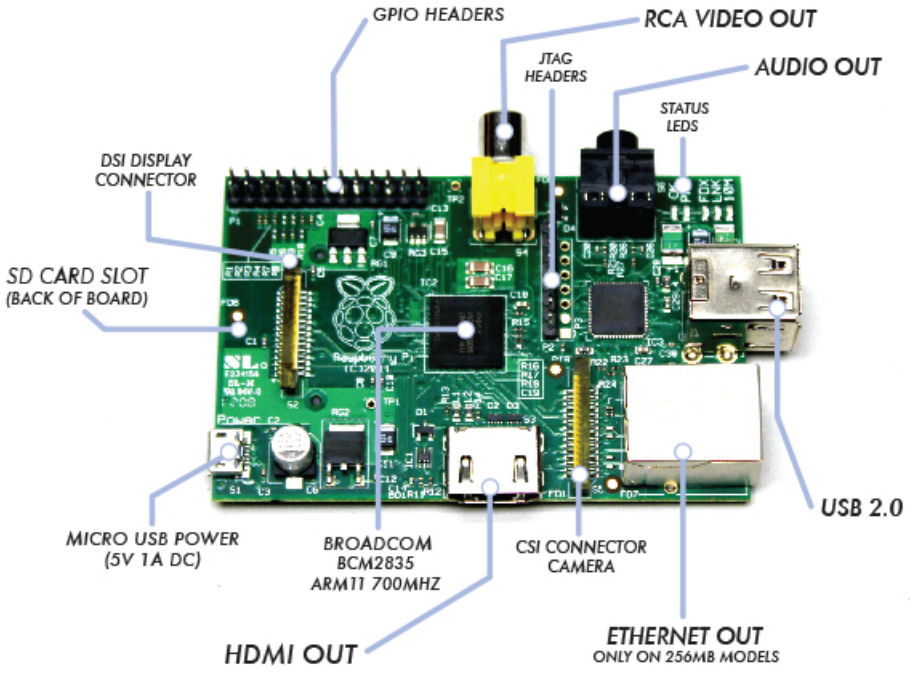


Figure 3.2: Raspberry Pi model B

The model used on this project was the model *B* since it was the one available at

Specifications	
CPU	ARM1176JZF-S
Frequency	700 MHz
SRAM	512 MB
Weight	45 gr
Dimensions	85.60 x 56.5 mm

Table 3.2: Raspberry Pi model B specifications

the time The board is shown on Figure 3.2 and its characteristics are displayed on Table 3.2. Only the most general characteristics are shown since the rest are similar to what one can find on a general purpose PC such as a GPU, HDMI and RCA jack, USB ports, 10/100 Mbit/s Ethernet adapter and more. As I said at the beginning of this section the Raspberry Pi is not designed with the power consumption in mind this means that is has a current draw of 700 mA (3.5 W) and does not have specialized modes for lowering power consumption as the Waspmote.

Development

Since the Raspberry Pi is able to run several Linux and Unix based operating systems, programming for this device is similar to programming for any other computer that also has those OSs. For this project since the Low-Power WiFi chip requires a serial connection to function I used first a USB adapter for testing and then a connection bridge for the Raspberry Pi that let me use a similar program for both the Waspmote and the Raspberry Pi.

Arduino shields connection bridge

This connection bridge was created by Cooking Hacks [8] with the idea of allowing people to use their Arduino shields with the Raspberry Pi. This bridge, that can be seen on Figure 3.3, allows the connection of any Arduino wireless module or shield, and even offers more connections trough i2C, SPI or UART. The bridge also comes with a library to make it easier to use the standard sockets in it.

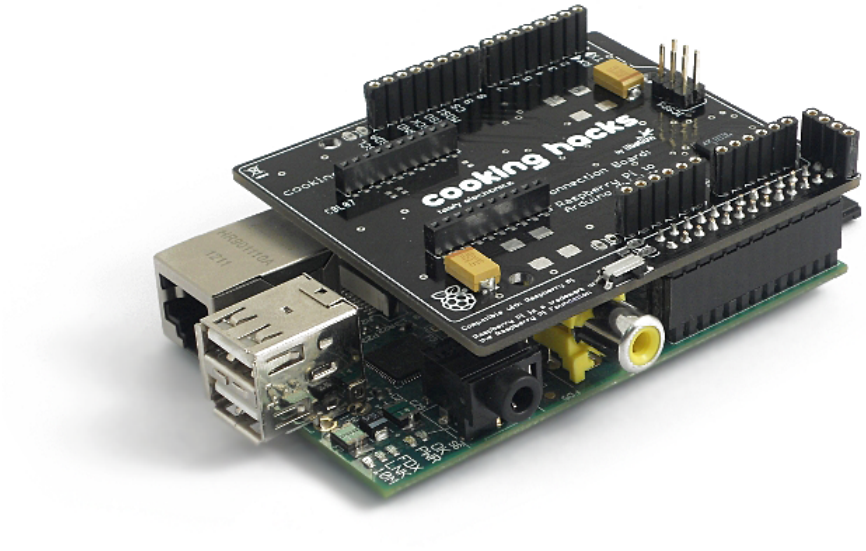


Figure 3.3: Raspberry Pi model B with Arduino shields connection bridge

Listing 3.3 has a sample of the code used to program this device, here since we do not have the help of an already created API for the connection to the Low-Power WiFi chip we have to manually write all the commands to send to it, these commands are already explained on Listing 3.1.

```
1  void setup()
2  {
3  Serial.begin(9600);
4  }
5
6  void loop()
7  {
8  while (Serial.available()>0) {}
9  Serial.print("$$$"); check();
10
11 Serial.print("set␣ip␣dhcp␣1\r"); check();
12 Serial.print("set␣ip␣protocol␣2\r"); check();
13
14 Serial.print("set␣wlan␣join␣0\r"); check();
15 Serial.print("set␣wlan␣phrase␣PASS"); check();
16 Serial.print("join␣SSID"); check();
17
18 Serial.print("set␣i␣r␣80\r"); check();
19 Serial.print("set␣c␣r␣GET$/test-get.php?\r"); check();
20 Serial.print("set␣o␣f␣1\r"); check();
21
22 Serial.print("open\r"); check();
23 Serial.print("exit\r"); check();
```

```
24 }
25
26 int main (){
27   setup();
28   while(1){
29     loop();
30   }
31   return (0);
32 }
```

Listing 3.3: Arduino shields connection bridge program example

3.3 Testing

This section will describe the tools used to test the devices mentioned on the previous section, and the reason of choosing them over possible alternatives.

3.3.1 WiFi router

The WiFi router used for the experiments was a Linksys WRT54G router. The router implements 802.11 b/g standards and the main characteristic that separates this router from similar ones is that it can be easily hacked in order to install new operating systems. This router had DD-WRT [9] installed, having a custom operative system helps in configuring what exactly the router does with the network traffic, this way we can effortlessly configure default routes and even monitor network traffic from the router itself allowing us to see devices connected, speed and power of each connection, packet errors received and more.

Figure 3.4: Linksys WRT54G router

3.3.2 Laptop

When working with WiFi connections it might be interesting to listen into the packets that are sent directly from the Low-Power WiFi device and not just to the ones that the router transmits to the server as being originated on the Low-Power chip.
In order to do this we need a WiFi chip that can be put in monitor mode using the Aircrack-ng suite for auditing wireless networks. The card must allow us to listen into the WiFi frequencies and dump the packets it reads in order to analyze them. This is done with the `airmon-ng` and `airodump-ng` tools. The suite functions better in a Linux system, in a Windows OS it requires additional libraries and modified WiFi driver DLLs.

Once the network interface is in monitor mode and `airodump-ng` is obtaining all the data, we can use Wireshark to analyze the dump file. Wireshark allows us to obtain a lot of information from the WiFi traffic, usually Wireshark will put the network interface in promiscuous mode and will try to read all the packets it can, but this mode will process 802.11 packages as belonging to 802.3 (Ethernet), so WiFi header information is lost. Instead, if we use monitor mode we can read those headers.

3.3.3 Server

In order to test the embedded applications that the Low-Power WiFi module has we need a server to receive the connections. This server must be able to create HTTP, FTP, UDP and TCP servers and be able to monitor network traffic for debugging purposes. For this, the best and easiest solution I found, is to have a Linux server with the XAMPP package [26] installed. With XAMPP we have a full HTTP server with PHP support to effortlessly create little scripts to log the information retrieved from connections with the sensor nodes and it also comes with a simple FTP server to test the built-in embedded networking applications that the RN-171 chip has.

Having a Linux system also gives us access to a lot of network tools such as `netcat` [13] or `socat` [20] to create TCP or UDP servers for the sensor node to connect to and `tcpdump` gives us the opportunity of debugging all the network connections that pass trough the network interface on the server. We can also use `iptables` [11] to redirect connections to and from the server to separate networks.

And since we are using a Linux system we can easily create shell scripts to make little tests with the sensor node, in the case of this project I used the *Bash* shell.

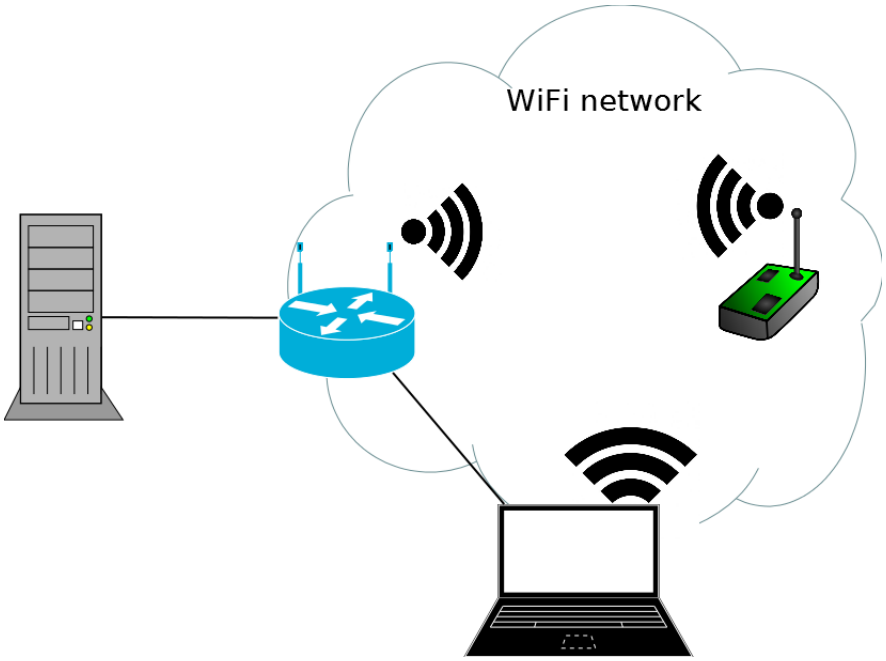


Figure 3.5: Testing environment

Chapter 4

Experiments

Contents

In this section I will explain the experiments I did to test how the sensors nodes and the Low-Power WiFi chip behaved taking the power consumption of the devices into consideration.

4.1 Power consumption

The first experiments where designed to test the raw power consumption of the RN-171 chip and compare it with the power requirements of the Waspmote. I did not compare it with the Raspberry Pi since the power needed to run the Raspberry Pi model B is several orders of magnitude higher than that of the Waspmote. This is enough to ignore the power consumption of most low-power devices that can be used on the Raspberry Pi.

In order to do this tests, first, I tested the battery usage by charging the battery to full and then having it send data non-stop to the server until it ran out of battery. This is a simple test to see how the device behaves with the battery and also to quickly test the low power modes the Waspmote has, and if it really lowers the power consumption as much as it is described in the specifications.

After this a multimeter was used to measure the current the devices needed to work.

One thing I noticed when making these tests is that any little device that is used impacts a lot in the consumption. For example at the beginning I was using the LEDs on the Waspmote to debug on which state the program was, since I could not use the USB connection to send debug information (if the USB is connected the battery is charging.) What I did not take into account is that compared to the already low current requirement of the Waspmote the LEDs, even while being low-power, need a lot of current.

Another thing that affects the testing is the behavior the Waspmote's power saving states have with the modules connected to it. In the Waspmote API once a `sleep()`, `deepSleep()` or a `hibernate()` function is called it either keeps the power ON or turns it completely OFF, it does not allow to set the the module to sleep. And though the API has some functions to put the module to sleep, the method it uses puts the chip to sleep on a timer and *not* until an event wakes it up. Nonetheless this is not a problem since depending on the sleep time of the Waspmote it is better to just turn it completely OFF and turn it ON when the Waspmote wakes up. This is also the recommendation of Libelium on all the example programs they offer.

4.1.1 Battery

The main idea of this test is to see how long the Waspmote would run in a real environment and to see the battery discharge profile using the different power saving methods. In order to see how much battery the Waspmote has left the programming API provides a function to get the battery percentage that is left. This percentage is calculated based on the voltage the Waspmote receives from the battery, the voltage is lower the less power the battery has, then this function translates this voltage into a percentage based on internal profiling data.

In order to test this I used several methods of sending data with the network application the RN-171 chip has, HTTP, TCP and UDP. UDP was unreliable since a lot of packets were getting lost, this is further explained when I analyze the built in applications. The idea is to have a program listening on the server that will receive the remaining power

the Waspmote has and a variable that contains the amount of readings the Waspmote has already sent with some other debug information about times to execute different functions, this program will then store this information on a log keeping the time the message was received and all the data. This way we can see on the log how much time it needs to send each measurement, the measurement number tells us if there has been any package lost or not sent, so we can take this also into account, and lastly we have the measure itself.

From this measures we obtain some strange data. Even though the battery is not charging anymore, meaning that the system detects a full battery, the internal battery reading once the charging is disconnected varies from 72% to 43% instead of the 100% that it should read. Also, when discharging, the battery seems to stabilize on some percentages, 32%, 9% for example, the Waspmote is able to keep working on those percentages for more that four times it kept on other percentages. Those percentages are kept constant through the tests, though it varied from one battery to another.

With this we can conclude that the battery reading function is not reliable to estimate the remaining time the Waspmote has to keep sending information. Though we can say that the Waspmote lasted a mean time of 42.74 hours while sending information non stop.

4.1.2 Measured

The next test I did involved a multimeter to measure the current draw of the Waspmote and the RN-171 chip, and compare it with the specifications given by the manufacturer. The circuit to measure the current is quite simple and it can be seen on Figure 4.1. I decided to put the multimeter between the battery and the Waspmote and read the current draw of the full Waspmote instead of putting it just between the WiFi module and the Waspmote.

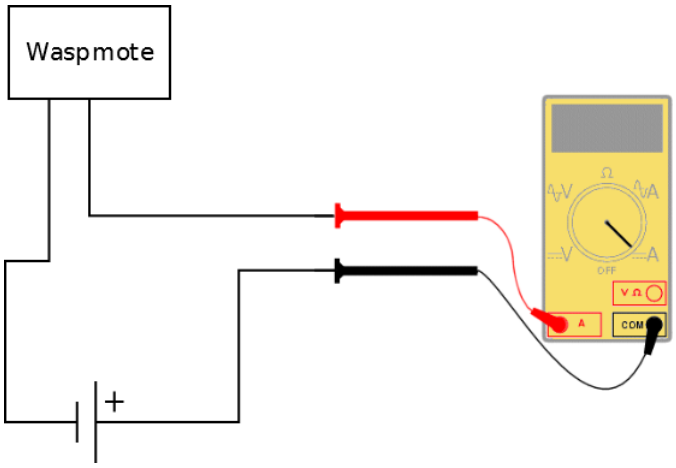


Figure 4.1: Current measurement circuit

For this experiment I decided to measure the current the Waspmote needed on its four states, normal, sleep, deep sleep and hibernate. For the WiFi module since it is not put in a sleep state just ON or OFF I measured the current while it was in different parts of the program, these parts where just having the module turned ON, the initial setup,

State	Manual	Measured
Normal	15mA	16mA
Sleep	55μA	57.3μA
Deep Sleep	55μA	57.3μA
Hibernate	0.06μA	See Figure 4.2

State	Manual	Measured
Normal	33mA	36mA
Setup	33mA	38mA
Connecting	38mA	40mA
Send	38mA	36mA

Table 4.1: Current measurements of the Waspmote and the WiFi module

connection to a WiFi network, and sending data. The results of these tests can be seen on Table 4.1.

After making the measurements we can see that the results obtained are quite similar to the ones promised by the specifications. There is a weird behavior when entering the hibernate state in the Waspmote, the plot of the current with respect to the time can be seen on Figure 4.2. This strange behavior means that the hibernate is actually worse that the sleep modes if the Waspmote uses it for less than 11 seconds since once it enters the sleep modes the current draw drops to 57.3μA, but if hibernating it needs about 11 seconds to reach that value, if we take into account that each time the Waspmote wakes from the hibernate state it must execute the setup function again, this time might be even more.

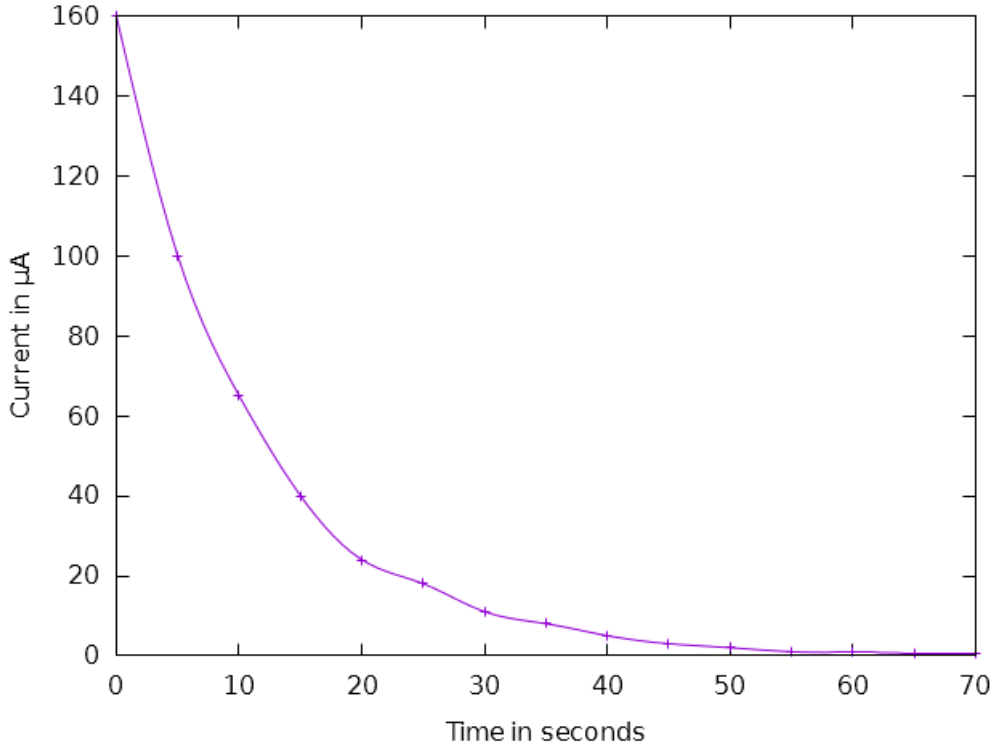


Figure 4.2: Hibernate state current draw behavior graph

4.2 Performance

In this section I will describe the performance results of using the RN-171 Low-Power WiFi module with the Waspmote, using the libraries provided by Libelium. In order to further test the module I also tried it in a Raspberry Pi since it should have a similar behavior independently of the system it uses. There is a huge difference between the Waspmote and the Raspberry Pi in terms of performance as it can be seen on Chapter 3, this means that even if the module works at the same speed in both cases the speed of the device in order to transmit data to and from the WiFi module might affect some of the results. But we are connecting to the module through an serial connection, and this further limits the connection speed to and from the WiFi module making so that it has the same speed for both of them.

In this section I will explain the performance results of the WiFi module, beginning with measuring the time it needs to do several tasks, then an analysis of the built in applications and finally measuring the strength and range of the module.

4.2.1 Working speed

Besides having a low current requirement, the WiFi module needs to be as fast a possible to lower the total power consumption. In order to see the time the module requires to do several tasks I measured it using the Waspmote API functions that allow us to calculate the time it needs to execute some function using the CPU cycles between both calls. The Raspberry Pi libraries did not have such a function available so I tried to manually measure the time just to compare it with the Waspmote, and the results were quite surprising on the Waspmote.

Table 4.2 show the times measured on both the Waspmote and the Raspberry Pi in order to first configure the module then join a network and finally send the data through different means.

Phase	Waspmote	Raspberry Pi
Turn ON	2500ms	<1000 ms
WiFi setup	9196ms	<1000 ms
Join network	2879ms	<1000 ms
HTTP send	13538ms	<1000 ms
TCP send	7421ms	<1000 ms
UDP send	6286ms	<1000 ms

Table 4.2: Comparison of times needed in several phases of the WiFi module

As we can see on Table 4.2 the Waspmote needs rather more time to do each of the phases than the Raspberry Pi. Even more strange was that I was *always* obtaining a result of 2500ms when turning the module ON on the Waspmote. In order to see what was causing such differences I tried to understand how the Waspmote API code worked since it is all available on GitHub [23]. The main difference I noticed was that while on the Raspberry Pi the program sent the command to the WiFi module and then waited until an answer was available on the serial connection to send the following command,

the Waspmote API did several things each time a function was called and most of the time waiting was not simply waiting on the connection while there is no data available, but halting the program for a predefined time checking the connection and halting again until data was found or a timeout expired.

An example of this can be seen on Listing 4.1, this Listing shows a simplified version of some functions that are used to interface with the WiFi module. If we follow the thread that happens when we want to turn the module ON we can analyze what happens and why it needs so much time, first we call the function `ON()` and here we can see that the first thing it does is to turn it OFF Then we go to the `OFF()` function on line 18, here the code powers OFF the module by cutting the current and closing the connection, and finally the program halts for 500ms with the command on line 24, so we already have an accumulated delay of at least 500 milliseconds when turning ON. If we go back to the `ON()` function we see that it powers ON the module and then calls a function called `commandMode()`, this function puts the WiFi module in command mode so following calls to the API will be able to send the commands directly. The part that interests us from this function is shown between lines 27-33 of the Listing. In order to enter command mode the module has to receive an specific signal, `$$$` in this case. Before sending the signal, the API does several things not shown on the Listing, then it waits 1000ms as seen on line 29, sends the signal and waits another 1000ms before continuing with the execution, which gives a delay of 2000 milliseconds for this function. If we add the 2000 millisecond delay of this function and the 500 millisecond delay of the `OFF()` function we get that the total delay of the `ON()` function is 2500ms which is the value we obtained before. This means that the API spends more time waiting than really executing code.

```
1  bool WaspWIFI::ON(...)
2
3    OFF();
4
5    pinMode(XBEE_PW,OUTPUT);
6    digitalWrite(XBEE_PW, HIGH);
7
8    if( commandMode() == 1 )
9    {
10     return 1;
11   }
12   else
13   {
14     return 0;
15   }
16 }
17
18 void WaspWIFI::OFF()
19 {
20   closeSerial(SOCKET0);
21   pinMode(XBEE_PW,OUTPUT);
22   digitalWrite(XBEE_PW, LOW);
23
```

```
24   delay(500);
25 }
26
27 uint8_t WaspWIFI::commandMode(){
28 ...
29   delay(1000);
30   printString(cmd_mode_request,_uartWIFI); // "$$$"
31   delay(1000);
32 ...
33 }
```

Listing 4.1: Simplified version of Waspmote API WiFi `ON()` and `OFF()` functions

The delays are needed since the program has to wait in order to read the messages that the WiFi module is sending, but in most cases it can be lowered significantly. In the previous example the RN-171 user manual [25] specifies that the programs must wait *at least* 500ms after sending the signal to enter command mode before sending any other command, but as we can see on line 31 of Listing 4.1 it waits for 1000ms instead. There are even more delays through the API, I tried lowering some of them, whenever I thought that that it would not affect the normal execution of the program, and I managed to cut nearly by half the time needed to send a message through WiFi without any visible problem. These delays can possibly be lowered even more though it would require extensive testing to make sure that it is not affecting the normal functioning of the module with the API.

4.2.2 Built in Applications

In this section I will describe the built in network applications that the module has in order to send and receive data, the options they have, how they work and how reliable they are to send the data. In terms of how secure these applications are the chip does not support the secure versions of the protocols, such as HTTPS and FTPS, so all data transmissions will be in plain text though the Waspmote can add a layer of encryption to the data.

TCP-UDP

This module has the capability of creating client and server TCP and UDP sockets, so they can send data through them or receive commands from a monitoring device. The way they work is that the developer defines a connection through the programming API giving an IP to connect to, and a local and remote port for the connection and the we can already send data through the connection. The developer must be careful and explicitly close the connection in order to flush the buffer that the module has and send the information.

I also made some test in order to test the reliability of the connections, since we are using a low-power device the WiFi connection may also be weaker than that of a normal WiFi and it might be more prone to losing packets. For this test I put the module sending small messages with just a message sequence number to the server, and then in the server I checked what messages arrived and calculated how many were lost with the missing sequence numbers. After sending sequences of 1000 TCP and UDP packets on

five different days, some further tests modifying the distance to the router were also made. These where the results:

TCP: 98% of the packets sent arrived to the destination. This means that it fails more than in a normal environment but not enough to make a successful connection impossible.

UDP: Just 63% of the packets sent arrived to the destination. This value dropped drastically the further from the WiFi router the Waspmote was located. Such a high rate of packets lost makes it highly impractical to use this method for sending information.

Sending messages through UDP has some benefits over TCP, since it is faster we lose less power while sending the message, but we cannot be sure that the packet has arrived to the destination and with a probability of not arriving of 37% we might want to send at least 2 packets in order to make sure at least one of them arrives so that we do not lose the time it took to take the measurement on a wireless sensor network. But instead if we use TCP we are able to know if the packet has arrived and resend it if something fails, and the overhead to do this is not as much in terms of processing on the Waspmote to choose UDP over TCP.

HTTP

The RN-171 chip also has the ability to function as an HTTP client. It works as a wrapper around its TCP connections, and is just able to use the HTTP GET function. So if a developer wants to send data using this method, which is not actually recommended, he can use the URL it is calling to append the data. Though we have to take into account that if we are using the Waspmote programming API there exists a limit to the length of the URL when I did the tests the request URL was copied to a character array with a length of 128 characters before sending the command to the WiFi module, this means that even if the URL is larger than that the API will cut it internally to fit it into the array. This issue has been partially solved on recent versions of the API and they have increased the size of the array to 512 characters. It is also possible using the Waspmote API to create a frame, a data structure created by Libelium to ease the process of obtaining and sending measurements from sensors, with the readings of the sensors and send the entire frame.

Though it is not possible to send POST request method using the HTTP client, this can be bypassed by creating the HTTP request body directly in the Waspmote and send it as a normal TCP connection.

FTP

The last application that the low-power WiFi chip has is an FTP client. In conjunction with the Waspmote API we can use it to send files from the SD card in the Waspmote to an FTP server. In order to test the application I created six different files with random data and increasing size, and sent them to the server simulating a sensor that takes a photo and sends it to the server. I measured the time it took to send the files and used the *md5* hash to test if the files received where the same as the original.

Size	Time	Successful
10KB	13.196s	92%
15KB	19.025s	90%
20KB	25.753s	86%
30KB	39.115s	80%
40KB	52.653s	66%
80KB	106.294s	32%

Table 4.3: FTP file transfer times and successful tries

Table 4.3 shows the time the API needed to send each of the files and the percentage of successful transmissions. As it can be seen the bigger the file the more probable is that it fails somewhere in the transmission, the failure is not just receiving some random bytes instead of the real ones, one the transmission failed it stopped and no more bytes where sent to the FTP server so only part of the file was received.

The transmission speed can be calculated with this data and it gives us a mean speed of 6283bps. This speed is not just limited by the WiFi connection since the tests where done with a 24Mbps connection. Since the file must be read from the SD card this is the main factor that was slowing the connection and it could also cause some of the errors in the transmission.

4.2.3 Strength

Another interesting test is to compare the strength of the Low-Power WiFi with that of the normal WiFi. In a real life scenario the sensor nodes will be at a certain distance from the sink node or gateway router and it might be interesting to see how far they can be from each other while still being able to communicate between them. For this test I tried two different cases one was to calculate the total range of the connection and the other to test how it will work inside a building.

Range

In order to know the range of the low-power WiFi I had to do the test outside, in a place with direct visibility between the router and the Waspmote. For this I put the router in the window of the laboratory and headed outside to take measurements every 8 meters in a straight line, the measurements involved being 3 minutes in each point and having the Waspmote send the connection strength information to the server, at the same time I was taking notes on the connection strength on a mobile phone also to compare them and I also had a script running on the router to send information to the server every second with the reported connection strength by the router of both, the Waspmote and the mobile phone.

An overview of this system can be seen on Figure 4.3. On each point, using the mobile phone, I connected to a web page on the server and started the dumping of the information with the ID of the point I was, this PHP script sent the information to another server with superuser permissions in order to start the `tcpdump` program and copy the information sent by both the Waspmote and the router to several files with the ID of the measurement

point so they could be later retrieved easily. The Waspmote then starts sending packets with sequence numbers to the server in order to know how many have reached and how many failed, the mobile also did the same so later the results can be compared.

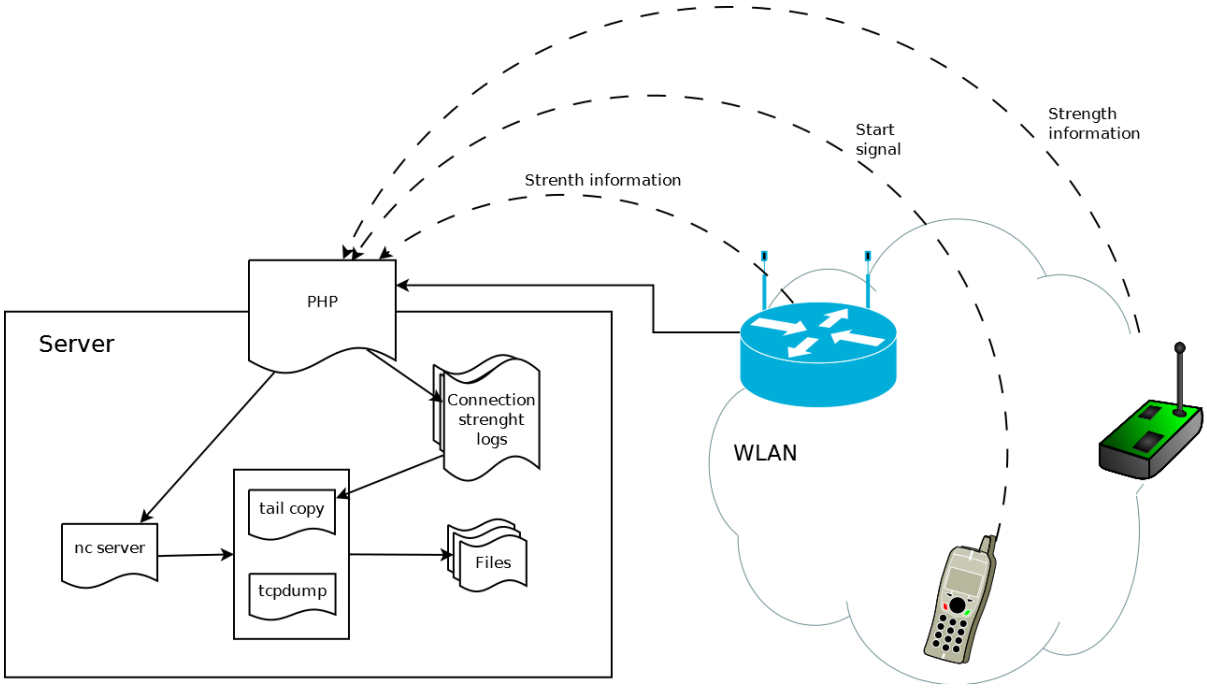


Figure 4.3: Range measuring system overview

The range tests were made by having a WiFi 802.11g specification network at 54Mbps, the wifi module had the standard configuration of 24Mbps and maximum power a +5dBi antena was also used with the module. Figures 4.4, 4.5 and 4.6 show the results of this experiment.

As we can see on Figures 4.4 and 4.5, the low-power WiFi module always reported having better connection strength than the mobile phone and the same happened from the router's point of view, but the mobile phone managed to last some 32 meters more before losing connection and being unable to reconnect. Though as shown on Figure 4.6 the mobile phone had a much more stable connection to the router than the Waspmote since from the beginning the Waspmote was losing packets. This means that the mobile phone has more advanced power conservation mechanisms, that make it able to lower the WiFi connection power while still managing to have an stable connection.

After this experiment we decided to do the same test, but changing the speed on the Waspmote, since lower speed at the same power means that each bit sent has more power and is less susceptible to interference, meaning that it should have better range. After doing the experiments we obtained an strange result, *the range was worse*. Lowering the transmission speed improved the reliability of the network, but at the same time the total range of the module went 16 meters down.

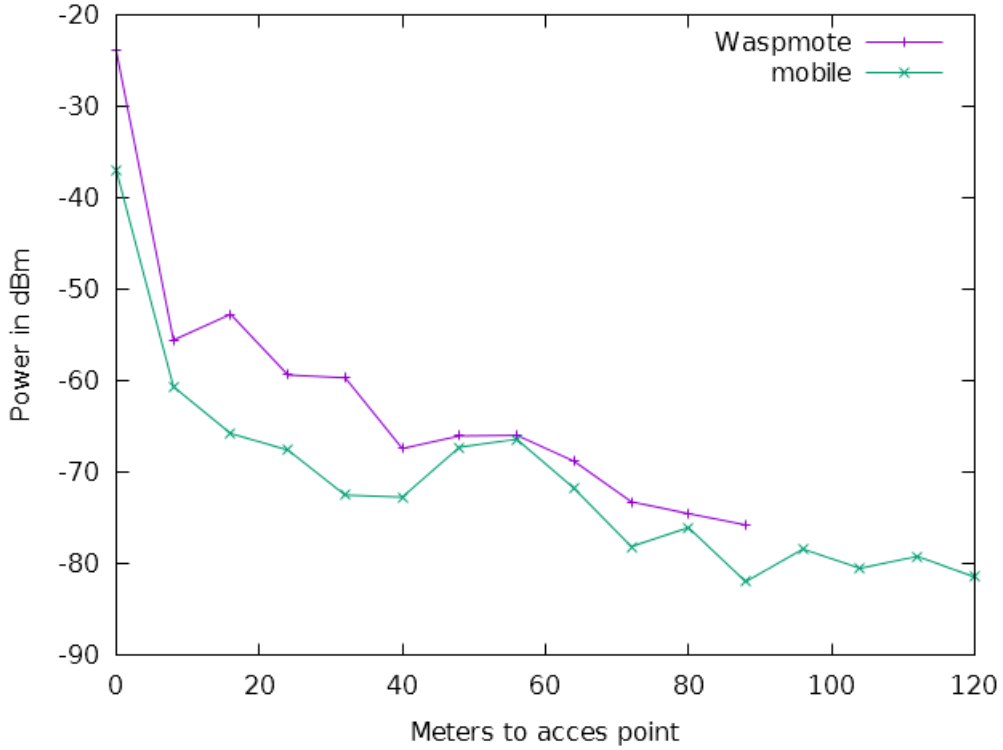


Figure 4.4: Router reported strength of the connection

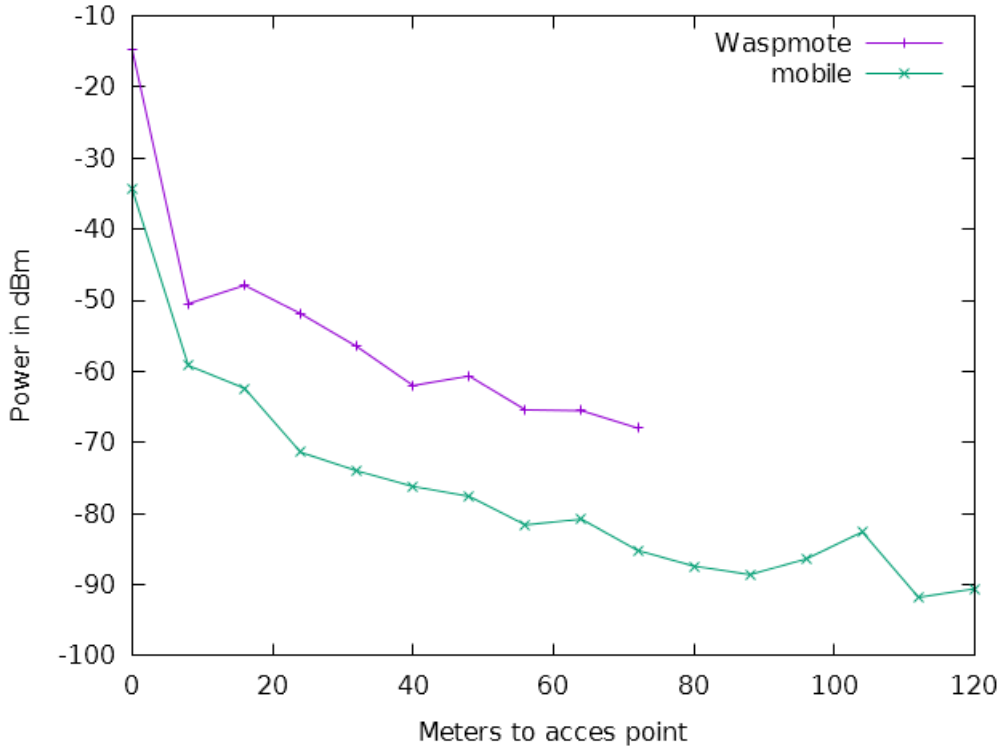


Figure 4.5: Device reported strength of the connection

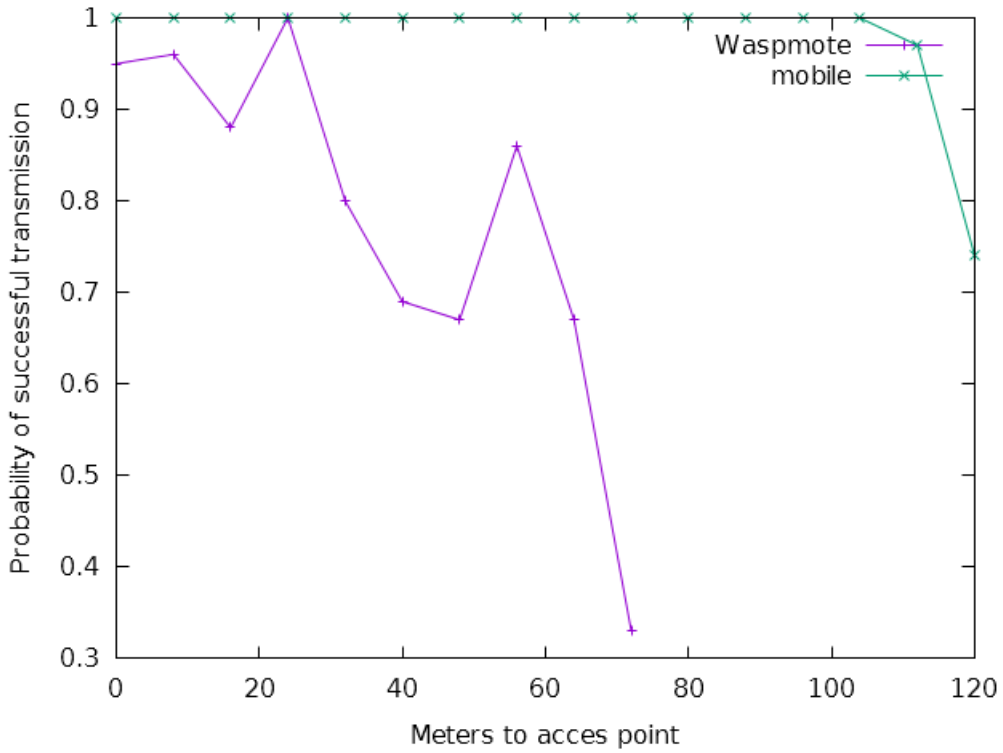


Figure 4.6: Probability of having a successful transmission

Indoors

The other test to see how its range is to measure it indoors. Being inside a building adds barriers to the router so there is no direct visibility between the Waspmote and the Router. The test points inside were not as many as outside since the connection strength dropped quite fast when putting a wall or another such obstacle between the router and the Waspmote. For the test there were 4 points chosen inside a building: the first one was just outside the door of the laboratory, the second was behind another wall, the third and the fourth we going up or down the stairs, putting the floor or the ceiling between the node and the access point.

I tried using more points, but for the Low-Power WiFi module even one wall was enough to drastically lower the signal quality, so while the mobile phone had a good connection to the router, the Waspmote was struggling to connect to the access point.

In this test, the maximum range of the RN-171 chip was found to be of 15 meters, but always depending on the number of obstacles to the router, if the router is behind some walls this range falls to 7-10 meters at maximum.

In this case I also tested if lowering the transmission speed and found that the connection behaved the same way as when lowering the speed on the previous test, even if the reliability is better, the range is further reduced. Thought the reduction in range is not as significant inside as it was outside.

4.3 Packet sniffing

In order to analyze the strange behavior the module had with the transmission speeds in the previous section we decided to try to see the packets that the module was sending to the access point and see at what speed they were being sent. This way, by sniffing packets we could also see if the device was transmitting packets if it found that they were not being acknowledged and see if it behaved in a similar way to a normal WiFi device.

The tests were not conclusive since the packet capture was highly erratic. I did not find a single case of a transmitted packet, but since I was not able to capture a high enough amount of packets transmitted by the low-power chip I cannot assure that it does not re-transmit packets.

What I found out is that even if the module is configured to work at 24Mbps and all transmissions are done at that speed, all the packets regarding the 802.11 connection were being sent at 1Mbps. So all connection configuration packets were being sent at a lower speed than messages sent from the Waspmote.

Chapter 5

Conclusions

Contents

5.1 Conclusions

In this project I had to analyze the feasibility of using a low-power WiFi chip, more specifically the RN-171, in order to build it in a wireless sensor network. There are several other technologies that offer better power consumption, but they also require additional hardware to connect the network to the Internet, or an already created network by normal users.

The chip is easy to configure and has the normal configuration options one can find in any non Low-Power WiFi device such as DHCP, several WiFi password schemes etc. It also comes with a simple application that can be run when it first starts so that a non technical user can configure the connection options through a graphical interface by connection to the module as if it were an access point.

In the case of the Waspmote the company that commercializes it, Libelium, provides an API for developers so they do not have to be searching in the manufacturer manual for the specific syntax of the commands to send to the module. This API also makes it easier to interface with WiFi module since it provides an abstraction layer and joins several configuration steps in one. Though the API has some problems, I already mentioned that it seems to cut some messages sent through the HTTP client, and there are a lot of areas that could be improved still, the timing for example. But it seems that it is still in development and Libelium is adding improvements steadily to the API.

5.1.1 Experiment results

From the tests we see that the WiFi chip has a really low-power requirement that is achieved lowering the transmission power and improving internal circuitry. Though if using the Waspmote API the power consumption is artificially increased by delaying the time it takes to send a message.

The embedded network applications make it much easier to program and have all networking functionality separated from the device that is doing all the data processing. This also means that it can be used in simple sensor nodes that themselves would not be able to have even a micro TCP/IP stack.

With the range tests we see that it has a lower range than that of the normal WiFi, but not by that much considering the power difference. But when trying to use the module inside a building we have to be careful if we want it to be in the range of an access point since walls and other obstacles affect it much more that to the normal WiFi.

5.2 Future work

Some improvements could be made in the Waspmote API so that the time to send a message can be lowered, this way since the WiFi module must be less time ON it will also reduce the consumption. Further improvements also could be made, such as adding simple functions to send a HTTP POST message even if it is not using the internal HTTP client program.

A function can also be created where depending on several factors like the signal strength or the previous errors when sending messages the power and the transmission speed of the RN-171 is controlled. Lowering the transmission power if everything works alright and increasing it if the connection is not good enough for example.

The strange behavior observed when changing the transmission speed of the module can also be further analyzed to see the real reason behind the apparent loss in range.

References

[1] AEmet, Calidad del aire. http://www.aemet.es/es/eltiempo/prediccion/calidad_del_aire.

[2] Aire limpio ya. http://microdonaciones.hazloposible.org/proyectos/detalle/?idProyecto=112.

[3] Calidad del aire. Departamento de Territorio y Sostenibilidad, Generalitat de Catalunya. http://mediambient.gencat.cat/es/05_ambits_dactuacio/atmosfera/qualitat_de_laire/.

[4] Caliope, Sistema de pronóstico de la calidad del aire. http://www.bsc.es/caliope/en.

[5] Ciudadanas y ciudadanos por el cambio en Leganes. http://www.ciudadanosporelcambio.com/dblog/articolo.asp?articolo=175.

[6] CompNet Research Group. http://compnet.ac.upc.edu/intranet/.

[7] Contaminación en Madrid: Gallardón y el Ayuntamiento mienten. http://www.ecologistasenaccion.org/article731.html.

[8] Cooking Hacks. http://www.cooking-hacks.com/.

[9] DD-WRT webpage. https://secure.dd-wrt.com/site/.

[10] Facultat d'Informatica de Barcelona (FIB). http://www.fib.upc.edu/fib.html.

[11] Iptables userspace command line program to configure packet filtering ruleset. http://www.netfilter.org/projects/iptables/index.html.

[12] La calidad del aire en Madrid. http://elpais.com/diario/2003/05/05/madrid/1052133871_850215.html.

[13] Netcat utility to read and write data across network connections. http://nc110.sourceforge.net/.

[14] Raspberry Pi Foundation. http://www.raspberrypi.org/.

[15] Roving Networks RN-171 datasheet. http://ww1.microchip.com/downloads/en/DeviceDoc/70005171A.pdf.

[16] Smart Agriculture project in Galicia to monitor vineyards with Waspmote. http://www.libelium.com/smart_agriculture_vineyard_sensors_waspmote/.

[17] Smart City project in Serbia for environmental monitoring by Public Transportation. http://www.libelium.com/smart_city_environmental_parameters_public_transportation_waspmote/.

[18] Smart Parking and environmental monitoring in one of the world's largest WSN. http://www.libelium.com/smart_santander_smart_parking/.

[19] Smart Water project in Valencia to monitor Water Cycle Management. http://www.libelium.com/smart_water_cycle_monitoring_sensor_network/.

[20] Socat multipurpose network relay. http://www.dest-unreach.org/socat/.

[21] The CommSensum Platform overview. http://compnet.ac.upc.edu/intranet/?q=node/199.

[22] The CommSensum webpage. http://commsensum.pc.ac.upc.edu/.

[23] Waspmote API Repository. https://github.com/Libelium/waspmoteapi.

[24] Waspmote Technical Guide. http://www.libelium.com/downloads/documentation/waspmote_technical_guide.pdf.

[25] WiFly Module Command Reference User's Guide. http://ww1.microchip.com/downloads/en/DeviceDoc/50002230A.pdf.

[26] XAMPP web page. https://www.apachefriends.org/index.html.

[27] Ian F Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. Wireless sensor networks: a survey. *Computer networks*, 38(4):393–422, 2002.

[28] Giuseppe Anastasi, Marco Conti, Mario Di Francesco, and Andrea Passarella. Energy conservation in wireless sensor networks: A survey. *Ad Hoc Networks*, 7(3):537–568, 2009.

[29] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

[30] Rory Cellan-Jones. A 15 pound computer to inspire young programmers. http://www.bbc.co.uk/blogs/legacy/thereporters/rorycellanjones/2011/05/a_15_computer_to_inspire_young.html.

[31] Andres Garcia-Saavedra, Pablo Serrano, Albert Banchs, and Giuseppe Bianchi. Energy consumption anatomy of 802.11 devices and its implication on modeling and design. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pages 169–180. ACM, 2012.

[32] Jan Holler, Vlasios Tsiatsis, Catherine Mulligan, Stefan Avesand, Stamatis Karnouskos, and David Boyle. *From Machine-to-machine to the Internet of Things: Introduction to a New Age of Intelligence*. Academic Press, 2014.

[33] Gerd Kortuem, Fahim Kawsar, Daniel Fitton, and Vasughi Sundramoorthy. Smart objects as building blocks for the internet of things. *Internet Computing, IEEE*, 14(1):44–51, 2010.

[34] Benedikt Ostermaier, Matthias Kovatsch, and Silvia Santini. Connecting things to the web using programmable low-power wifi modules. In *Proceedings of the Second International Workshop on Web of Things*, page 2. ACM, 2011.

[35] Bozidar Radunovic, Dinan Gunawardena, Peter Key, Alexandre Proutiere, Nikhil Singh, Vlad Balan, and Gerald Dejean. Rethinking indoor wireless mesh design: Low power, low frequency, full-duplex. In *Wireless Mesh Networks (WIMESH 2010), 2010 Fifth IEEE Workshop on*, pages 1–6. IEEE, 2010.

[36] Serbulent Tozlu. Feasibility of wi-fi enabled sensors for internet of things. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2011 7th International*, pages 291–296. IEEE, 2011.

[37] Jean-Philippe Vasseur and Adam Dunkels. *Interconnecting smart objects with ip: The next internet*. Morgan Kaufmann, 2010.