ΠΑΝΕΠΗΣΤΙΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ
ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Σχεδιασμός και υλοποίηση συνεργατικών πρωτοκόλλων σε πραγματικά ασύρματα δίκτυα με χρήση λογισμικού ανοικτού κώδικα

Design and implementation of cooperative protocols in wireless networks using open source software

Διπλωματική Εργασία

Λαζαρίδης Λάζαρος

Επιβλέπων Καθηγητής: Κοράκης Αθανάσιος, Λέκτορας

Συνεπιβλέπων καθηγητής: Τασιούλας Λέανδρος, Καθηγητής

ΒΟΛΟΣ 2014

Ευχαριστίες

Με την περάτωση της παρούσας Διπλωματικής Εργασίας, θα ήθελα να ευχαριστήσω τον επιβλέποντα , Λέκτορα του Τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών κ. Αθανάσιο Κοράκη καθώς και τον συνεπιβλέποντα καθηγητή του τμήματος κ. Λέανδρο Τασιούλα για την ευκαιρία που μου δώσανε να ασχοληθώ με νέες τεχνολογίες και να κάνω έρευνα στο χώρο των ασυρμάτων δικτύων. Θέλω ακόμη από καρδιάς να ευχαριστήσω θερμά τον υποψήφιο διδάκτορα και μέλος του NITLAB Ιωάννη Καζδαρίδη για τις πολύτιμες συμβουλές του και για την σημαντική βοήθειά του στην ολοκλήρωση της διπλωματικής μου εργασίας

Επίσης, ευχαριστώ τους φίλους και συμφοιτητές μου για την αμέριστη υποστήριξή τους και την δημιουργία ενός ευχάριστου και δημιουργικού περιβάλλοντος κατά τη διάρκεια των σπουδών μου.

Τέλος, οφείλω ένα μεγάλο ευχαριστώ στην οικογένειά μου Αντώνη, Αργυρώ και Μαρίνα που με βοήθησαν να διαμορφώσω ένα ισχυρό υπόβαθρο καθώς και για την αμέριστη συμπαράστασή τους όλα αυτά τα χρόνια κατά τη διάρκεια των προπτυχιακών μου σπουδών.

CONTENTS

ABSTRACT

There are different wireless technologies that share the same 2.4 GHz frequency band. Such technologies usually operate in proximity and have to co-exist with each other. Frequency overlap across wireless networks with different radio technologies can cause severe interference and reduce communication reliability. The circumstances are particularly unfavorable for Wireless Sensor Networks(WSNs) that use ZigBee wireless technology and share the 2.4GHz ISM band with Wi-Fi. Due to the independent design and development in WSN, together with the unexpected dynamics during deployment of coexisting networks and devices, within the same frequency spectrum, it is crucial to ensure that a WSN maintains and provide its desired performance requirements. In order for a WSN to satisfy these requirements, it is necessary to avoid congested frequencies of the 2.4GHz band.

In this thesis, we report and analyze the existence of the aforementioned problem, and present a novel way to discover the most uncongested channels in which a WSN network can experience better performance in terms of data throughput. This work is consisted of two main parts. The first one is a prototype platform based on a SparkFun's pro Micro board equipped with a Cypress CYWM6935 radio receiver. Its purpose is to scan the 2.4GHz band and after applying a simple algorithm, returns the most uncongested ZigBee channel. The second part is a platform [1] designed by NITOS Lab, equipped with a SparkFun's pro Micro board and a slot unit for a XBee module. It is used to build communication between XBee modules. Finally we combine the measurements resulting from each platform in a Processing sketch in order to visualize the 2.4GHz band as well as offer frequency hopping functionality for a WSN.

1 Introduction

The continuous emerging of wireless technologies, in addition to the increasing demands for mobile applications, combined with the wide spread of new human–computer interaction models, such as ubiquitous computing, paved the way for the emergence of new networking frameworks and new applications.

Nowadays, users are equipped with more powerful, small, wireless mobile devices such as mobile phones, PDAs and sensors, these devices generate substantial amount of various data. Moreover, to support the new generation of situation–aware applications, there is a real need for new networking paradigms to support the exchange of information between these devices.

Personal mobile devices are typically small in size with limited processing, communication and energy capabilities. Thus, the new networks should support low power and low data rate transmissions. Several technologies were developed to satisfy the above mentioned prospects. A successful example of these technologies is IEEE802.15.4 networks, which is known as ZigBee networks. ZigBee is designed to be cost effective to guarantee a successful commercial spread. To reduce the cost these networks were designed to work in the crowded and overloaded 2.4 GHz ISM commercial band with different technologies such as IEEE802.11 known as Wi-Fi. The coexistence of different wireless technologies causes interference and packets collision, then packet retransmission. Which in turn, cause delay and reduce the delivery ratio. Moreover, ZigBee packet loss and retransmission leads to faster draining of the sensor battery. Therefore, the need to design analytical model to depict the impact of mutual interference between ZigBee and Wi-Fi.

In this thesis we focus on the coexistence between IEEE802.11 and IEEE802.15.4 wireless standards that operate over the 2.4GHz ISM band. Their overlapping frequency channels will be the point of our interest in addition to the interference issues on these channels. Regarding that ZigBee is not able to automatically hop to a clear channel, we propose an external software mechanism that scans and visualizes the 2.4GHz band as well as offering frequency hopping functionality.

There have been some studies about coexistence between the IEEE 802.11 and IEEE 802.15.4. According to [2], [3], [5] IEEE 802.15.4 has a little impact on the IEEE 802.11 performance. However, IEEE 802.11 can have a serious impact on the IEEE 802.15.4 performance if the channel allocation is not carefully taken into account [2], [4].

The remainder of the document is organized as follows:
Section 2 gives an overview of the IEEE 802.15.4 standard. Section 3 presents an overview of ZigBee wireless technology. Section 4 describes the IEEE 802.11a/b/g. Section 5 gives an analysis of the coexistence between IEEE 802.11b/g and IEEE 802.15.4 standards. Section 6 briefly presents the software and hardware components used for the purposes of this work and how these components are combined. In section 7 we describe our demo setup and experimental results. We conclude in section 8 and finally in section 9 we write down our references.

2 802.15.4 Protocol

2.1 Overview

The IEEE802.15.4 [6], [7], [8] is a part of the IEEE family of standards for physical and link layers. The standard is designed to offer the fundamental lower network layers of a type of wireless personal area network (WPAN) which focuses on low-cost, low-speed ubiquitous communication between devices (in contrast with other, more end-user oriented approaches, such as Wi-Fi). The emphasis is on very low cost communication of nearby devices with little to no underlying infrastructure, intending to exploit this to lower power consumption even more.

2.2 802.15.4 Layers

The 802.15.4 standard defines the physical layer (PHY) and media access control (MAC) layer of the Open Systems Interconnection (OSI) model of network operation (*Figure 1*). The PHY defines frequency, power, modulation, and other wireless conditions of the link. The MAC defines the format of the data handling. The remaining layers define other measures for handing the data and related protocol enhancements including the final application.

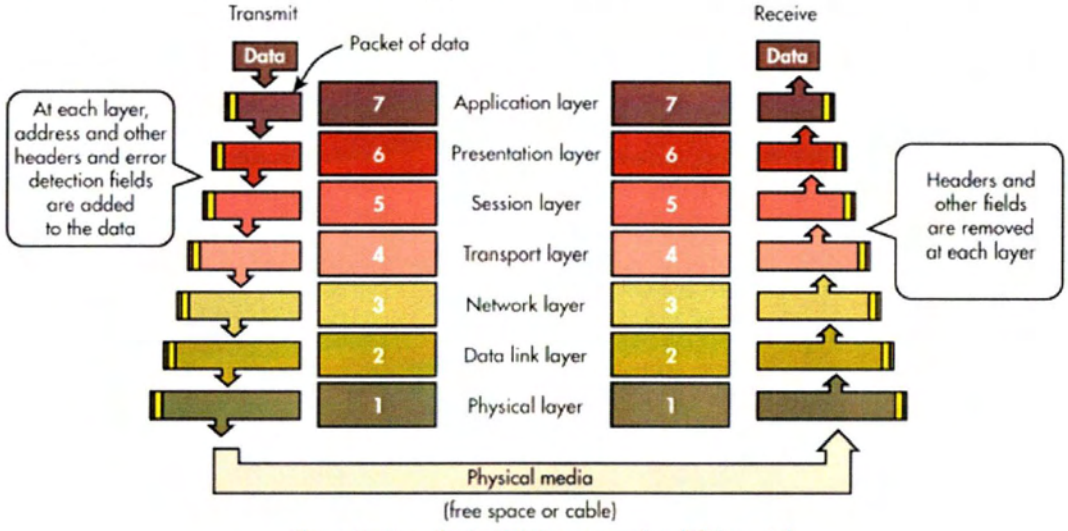


Figure 1. Open Systems Interconnection (OSI) model

Most networking systems, both wired and wireless, use the OSI communications model. Most systems also use at least the first four layers, but many do not use all seven layers. The 802.15.4 standard uses only the first two layers plus the logical link control (LLC) and service specific convergence sub-layer (SSCS) additions to communicate with all upper layers as defined by additional standards (*Figure 2*).

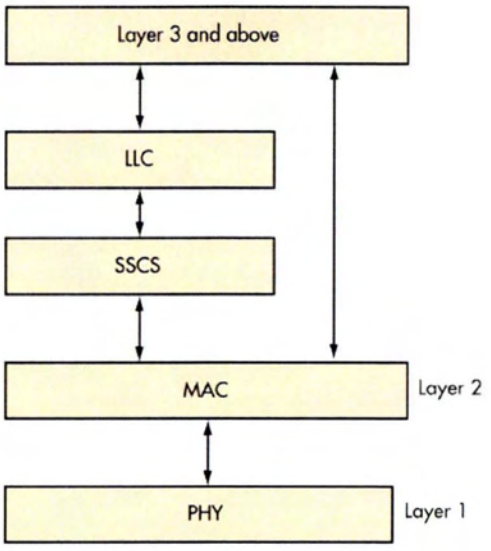


Figure 2. Layer 1 and layer 2 details of 802.15.4

2.3 802.15.4 Bands and Channels

The goal of the standard is to provide a base format to which other protocols and features could be added by way of the upper layers (layers 3 through 7). While three frequency assignments are available, the 2.4-GHz band is by far the most widely used *(Figure 3)*. Most available chips and modules use this popular ISM band.

OPTIONS FOR FREQUENCY ASSIGNMENTS			
Geographical regions	**Europe**	**Americas**	**Worldwide**
Frequency assignment	868 to 868.6 MHz	902 to 928 MHz	2.4 to 2.4835 GHz
Number of channels	1	10	16
Channel bandwidth	600 kHz	2 MHz	5 MHz
Symbol rate	20 ksymbols/s	40 ksymbols/s	62.5 ksymbols/s
Data rate	20 kbits/s	40 kbits/s	250 kbits/s

Figure 3. Options for frequency assignments

The standard defines 16 channels within this band, each 2 MHz wide with 3 MHz inter-channel gap-bands (*figure 4*). This is the most common frequency assignment worldwide.

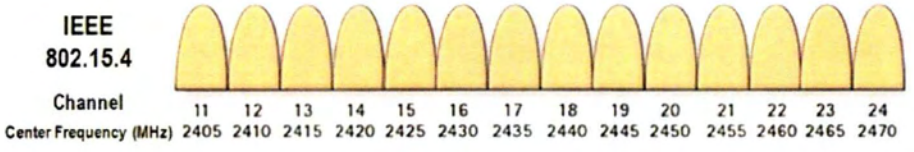


Figure 4. 802.15.4 Channels

2.4 802.15.4 Against noise

Direct Sequence Spread Spectrum (DSSS) is used to modulate the information before being sent to the physical layer (*Figure 6*). Basically, each bit of information to be transmitted is modulated into 4 different signals (another kind of bits alphabet), this process causes the total information to be transmitted to occupy a larger bandwidth but it uses a lower spectral power density for each signal. This causes less interference in the frequency bands used and improves the Signal to Noise Ratio (SNR) in the receiver due to the fact that is easier to detect and decode the message which is being sent by the transmitter.

There are different DSSS modulations depending on the hardware physical limits of the circuit and number of symbols which can be processed at a given time. Binary Phase Shift Keying (BPSK), Offset Quadrature Phase Shift Keying (O-QPSK) and Parallel Sequence Spread Spectrum (PSSS) let communication from bandwidths from 20Kb/s to 250Kb/s.

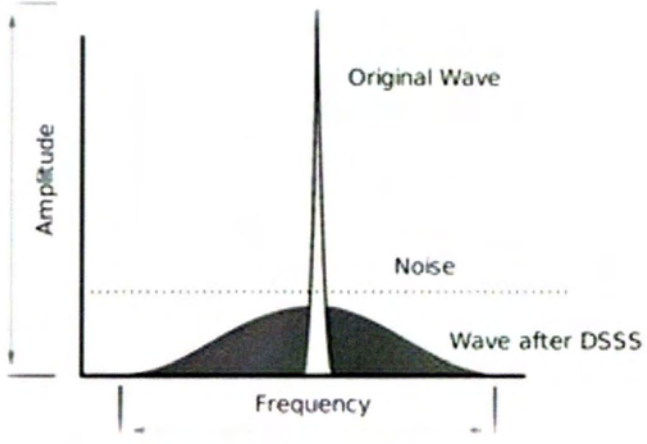


Figure 5. DSSS

2.5 802.15.4 against interference

2.5.1 Data Rate

A way to minimize the risk of interference is to reduce channel occupancy. This approach is followed by the IEEE 802.15.4 standard. While many intended applications for devices using IEEE 802.15.4 radios require a very low data rate (e.g. switching a light on and off, transmitting a temperature value), the underlying PHY layer communicates at 250 kbps. Compared to other RF systems targeting the same application range, this is a high data rate that allows to minimize time spent on air and reduce opportunities for collisions.

2.5.2 Build-in Scanning and Reporting

The IEEE 802.15.4 PHY layer provides the ability to sample a channel, measure the energy, and report whether the channel is free from interference and thus clear to transmit. This information is then made available to higher layers so that devices

using IEEE 802.15.4 radios have the possibility to select the best available channel for operation.

2.5.3 Multiple Channels

The IEEE 802.15.4 specification augments the opportunities for smooth coexistence by dividing the 2.4 GHz band into 16 non-overlapping channels, which are 2-MHz wide and 5-MHz apart (*Figure 4*).

2.5.4 Acknowledgements and Retransmissions

The IEEE 802.15.4 specification includes by default the acknowledgment of received frames. On receipt of a message, each device has a brief time window in which it is required to send back a short message acknowledging receipt. This technique allows messages that are transmitted but not successfully received to be detected. If the transmitting device does not receive the acknowledgment, it will assume that the message has not been delivered and will try again. Retransmissions are carried out until the message and its acknowledgment are both received or until, usually after a few tries, the transmitter gives up and reports a failure.

2.5.5 Carrier Sense Multiple Access-Collision Avoidance (CSMA-CA)

Each node listen the medium prior to transmit. If the energy found higher of a specific level the node the transceiver waits during a random time (including in an interval) and tries again. There is a parameter defined in the standard: macMinBE which sets the back-off exponent to be used when calculating this time slot.

2.5.6 Guarantee Time Slots

This systems uses a centralized node (PAN coordinator) which gives slots of time to each node so that any knows when they have to transmit. There are 16 possible slots of time. As a first step a node must to send to the PAN coordinator a GTS request message, as response the coordinator will send a beacon message containing the slot allocated and the number of slots assigned. There are some special frames like the ACK packets which doesn't require this method to be performed.

2.6 802.15.4 topologies

With regard to networking capability, 802.15.4 defines two topologies. One of them is a basic star *(Figure 6(a))*. All communications between nodes must pass through the central coordinator node. A basic peer-to-peer (P2P) topology is also defined *(Figure 6(b))*. Any device may then talk to any other device. This basic topology may be expanded into other topologies in the upper network layers, such as the popular mesh topology.

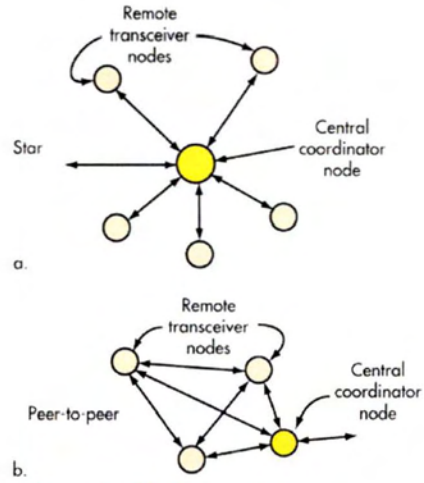


Figure 6. 802.15.4 topologies

2.7 802.15.4 as a low consumption protocol

It is ready to work with low-duty cicles. It means the transceiver can be sleeping most of the time (up to 99% on average) while the receiving and sending tasks can be set to take just a small part of the devices' energy. This percentage depends on the kind of communication model used. If beacon mode is used (star or PAN networks) the minimum amount of time used to transmit/receive this frames will increase the total time the transceiver is used. They can be sleeping for minutes or hours and wake up all at the same time to perform an Adhoc communication creating a mesh network just when really needed.

2.8 Protocols which extends 802.15.4

802.15.4 is the basis for the ZigBee, ISA100.11a, WirelessHART, MiWi and many others specifications, each of which further extends the standard by developing the upper layers which are not defined in IEEE 802.15.4. Alternatively, it can be used with 6LoWPAN and standard Internet protocols to build a wireless embedded Internet.

- ZigBee : The most widespread protocol which use 802.15.4
- ISA100.11a : is a wireless networking technology standard developed by the International Society of Automation (ISA). It defines procedures for implementing wireless systems in the automation and control environment with a focus on the field level.
- WirelessHART : The protocol utilizes a time synchronized, self-organizing, and self-healing mesh architecture. The protocol supports operation in the 2.4 GHz ISM band using IEEE 802.15.4 standard radios.

3 ZigBee

3.1 ZigBee Overview

The most widely deployed enhancement to the 802.15.4 standard is ZigBee, which is a standard of the ZigBee Alliance. ZigBee is a specification for a suite of high level communication protocols used to create personal area networks built from small, low-power digital radios. ZigBee is used in applications that require only a low data rate, long battery life, and secure networking. ZigBee has a defined rate of 250 kbit/s, best suited for periodic or intermittent data or a single signal transmission from a sensor or input device. Applications include wireless light switches, electrical meters with in-home-displays and other consumer and industrial equipment that requires short-range wireless transfer of data at relatively low rates.

3.2 ZigBee layers and kinds of different services

It uses layers 3 and 4 to define additional communications features (*Figure 7*). These enhancements include authentication with valid nodes, encryption for security, and a data routing and forwarding capability.

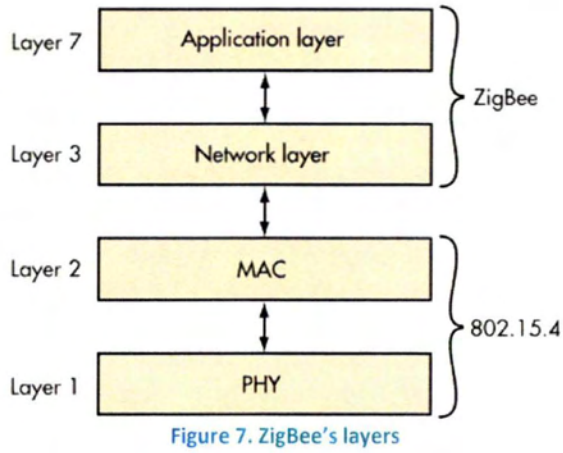


Figure 7. ZigBee's layers

ZigBee offers basically four kinds of different services:

- **Extra Encription services** (application and network keys implement extra 128b AES encryption)
- **Association and authentication** (only valid nodes can join to the network).
- **Routing protocol**: AODV, a reactive ad hoc protocol has been implemented to perform the data routing and forwarding process to any node in the network.
- **Application Services**: An abstract concept called **"cluster"** is introduced. Each node belongs to a predefined cluster and can take a predefined number of actions. Example: the "house light system cluster" can perform two actions: "turn the lights on", and "turn the lights off".

ZigBee is a layer thought to organize the network. The first thing a node (route or end device) which want to join the network has to do is to ask to the coordinator for a network address (**16b**), as part of the **association** process. All the information in the

network is routed using this address and not the 64b MAC address. In this step authentication and encryption procedures are performed.

Once a node has joined the network it can send information to its brothers through the routers which are always awake waiting for the packets. When the router gets the packet and the destination is in its radio of signal, the router first looks if the destination end device is awake or slept. In the first case the router sends the packet to the end device, however if it is sleeping, the router will buffer-ize the packet until the end device node is awake and ask for news to the router.

As well as the ZigBee standard adds network and application support on top the of IEEE 802.15.4 specification, it uses the coexistence techniques provided by IEEE 802.15.4 layers.

3.3 How a ZigBee network works

There are three kinds of nodes in a ZigBee network [6], [7]:

- **Coordinator (Master) :** Only one coordinator exists in each ZigBee network. Its function is to store information about the network and to determine the optimum transmission path between any two points of the network. Coordinator cannot be battery powered.
- **Full function device (Router) :** Routers act as an intermediate repeater that passes data from other devices. Routers cannot be battery powered.
- **Reduced Function Device (End Device) :** This device contains a minimal amount of functionality to enable it to talk to its parent node (either the coordinator or a router), it cannot relay data directly from other devices.

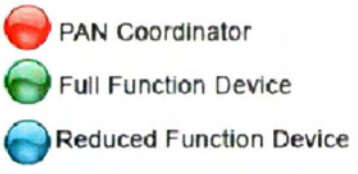


Figure 8. Coordinator, Router, End device

ZigBee creates start network topologies (*Figure 9*), not mesh ones. To create a completely mesh network all the nodes have to have the same role, all of them have to be "end devices + routers" so that they can route their brothers information and sleep when no action is required (saving energy). The DigiMesh protocol (over 802.15.4) sets a completely distributed network where all the nodes talk among them using p2p (equal to equal) datagram.

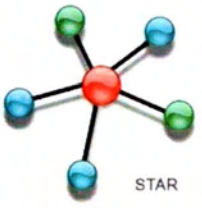


Figure 9. ZigBee's star topology

4 Wi-Fi / IEEE 802.11b/g/n

4.1 Overview

Wireless fidelity (Wi-Fi) includes IEEE 802.11b/g/n standards for wireless local area networks (WLAN), which are commonly used today to provide wireless connectivity in the home, office, and some commercial establishments. Wi-Fi is a popular technology that allows an electronic device to exchange data wirelessly over a computer network, including high-speed Internet connections. The Wi-Fi Alliance defines Wi-Fi as any "wireless local area network (WLAN) products that are based on the Institute of Electrical and Electronics Engineers' (IEEE) 802.11 standards" [9].

4.2 802.11 and OSI Model

The IEEE 802 standards committee defines two separate layers, the Logical Link Control (LLC) and media access control, for the Data-Link layer of the OSI model. The IEEE 802.11 wireless standard defines the specifications for the physical layer and the media access control (MAC) layer that communicates up to the LLC layer (*figure 10*).

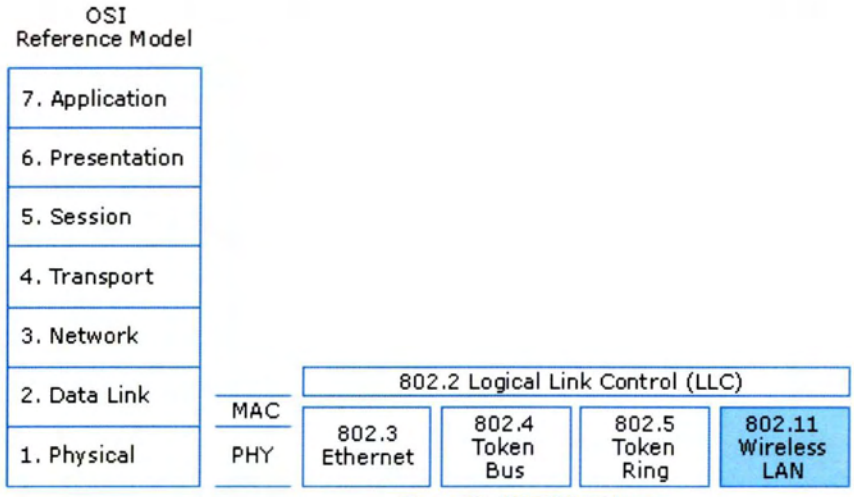


Figure 10. IEEE 802.11 layers

4.3 Channels and frequencies

The IEEE 802.11b and 802.11g are amendments to the IEEE 802.11 specification that extends throughput from 54Mbit/s to 600Mbit/s using the 2.4GHz band. The IEEE 802.11b operates in total of 14 channels available in the 2.4GHz band, each with a bandwidth of 22MHz and a channel separation of 5MHz (*Figure 11*). WLAN output powers are typically around 20dBm and operate within a 100m range.

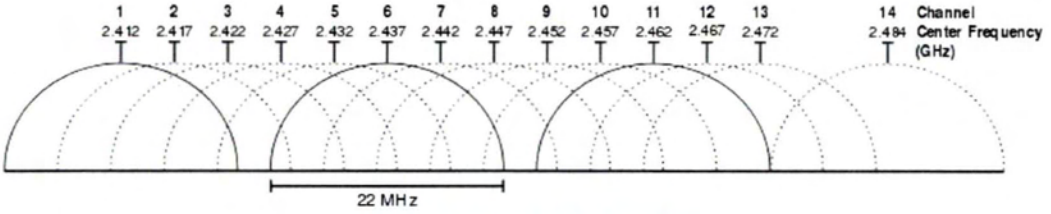


Figure 11. IEEE 802.11b channels

5 Coexistence between ZigBee and Other 2.4GHz technologies

5.1 Overview

ZigBee products based on IEEE 802.15.4 are acting more and more important roles in several applications, however, with the development of short range radio technology, more and more 2.4GHz ISM band of radio systems are being widely adopted in more fields, so designers have to deal with increasing interferences caused by other radio communication devices [10].

To make a ZigBee network work well, it is necessary to study on the mutual coexistence and interference between ZigBee (based on IEEE 802.15.4) and other 2.4GHz radio systems. The 2.4GHz Industrial, Scientific, and Medical (ISM) unlicensed frequency band is shared by many RF technologies, such as IEEE 802.11b/g Wi-Fi, ZigBee, Bluetooth, Wireless USB, cordless phone, etc. Due to its low transmit power feature, ZigBee is potentially vulnerable to the interferences introduced by these RF technologies rather than vice versa. In this section, the coexistences and interferences between ZigBee and Wi-Fi are mainly focused on.

5.2 Coexistence between ZigBee and Wi-Fi

The IEEE 802.11 Wireless LAN (WLAN) standard adopts DSSS and operates in a total of 14 channels available in the 2.4GHz band, numbered 1 to 14, each channel with a bandwidth of 22MHz and a separation of 5MHz.

The IEEE 802.15.4 is also based on DSSS. A total of 16 channels are available in the 2.4GHz band, numbered 11 to 26, each with a bandwidth of 2MHz and a channel separation of 5MHz.

Both ZigBee and Wi-Fi are based on DSSS. The signal is spread over a larger bandwidth, so narrow-band interferers block a smaller overall duty of the signal, allowing the receiver to receive the signal correctly. Because both are in 2.4GHz band, co-channel interference should be taken into account.

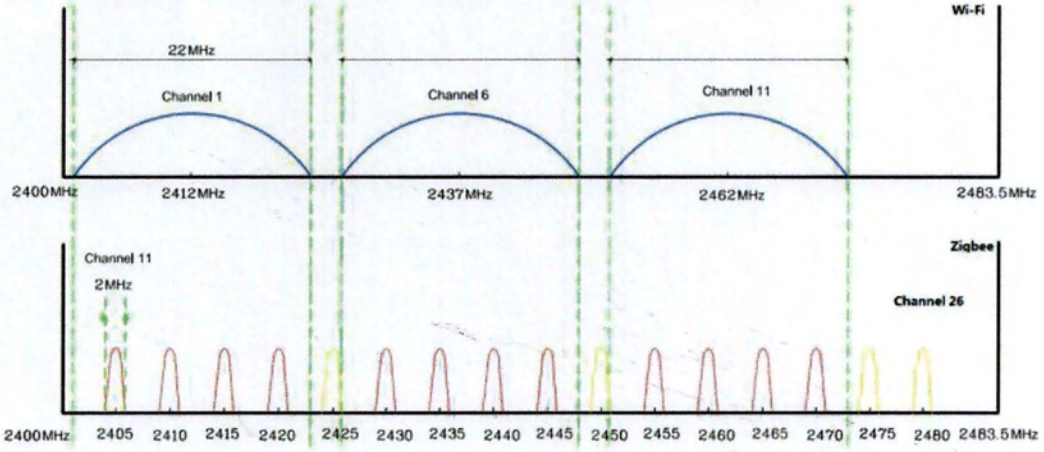


Figure 12. ZigBee's & Wi-Fi's channels on 2.4GHz band

The overlapping channels between 802.11 and 802.15.4 are shown in Figure 11, which shows channel 15, 20, 25, and 26 are not overlapped with Wi-Fi channels, these channels are of negligible interference. For each Wi-Fi channel, there are four overlapping ZigBee channels, two channels of them are at the edges and another two are close to the center frequency of the Wi-Fi channel. In fact, the interference level on the two channels close to the center is higher than those on the edges. PER (Packet error rate) has close relationship with distance (between interference source and receiver) and differences of center frequencies (between interference source and receiver). They can even coexist within very short range (2 meters) even when difference of center frequencies is considerable, however, when their center frequencies are very close, they can coexist only out of long distance (even dozens of meters). It shows that, if the interference source is more far from receiver, they can coexist better. Channel occupancy detection and dynamic channel selection are very important to keep good coexistence.

The interference with Wi-Fi, caused by ZigBee, is smaller than the interference with ZigBee, caused by Wi-Fi, it is because ZigBee's bandwidth (2MHz) is much smaller than Wi-Fi's bandwidth (22MHz), so ZigBee is a kind of narrowband interference source to Wi-Fi. 802.11b/g/n adopts spread spectrum technology, which can greatly restrain interference signal, in addition, for most ZigBee products the power is conditioned to 0dbm (1mW), which is not enough to pose a threat to Wi-Fi products, its power is far less than that of IEEE 802.11b/g, 20dbm (100mW), they can coexist very well if necessary measures are adopted.

What happens when the ZigBee products operate over an overlapped frequency (ZigBee , 802.11)? Especially, how do they behave on a congested frequency? How efficient can be the network be?

Later, this thesis will refer a way of countering this particular problem.

6 Components used & developed prototype

This section briefly presents the software and hardware components used for the implementation of our work. We firstly present the Arduino hardware and software components which were used along with the CYWM6935 RF radio transceiver in order to develop a prototype implementation that scans the signal strength of 2.4GHz band. Next we present the Processing IDE which used for the signal strength visualization. Then, we describe the XBee modules as well as how can be configured. Also, it is reported the essentials parts of Arduino's XBee Library which was used in order to program the Arduino for the desired XBee functionality.

6.1 Arduino

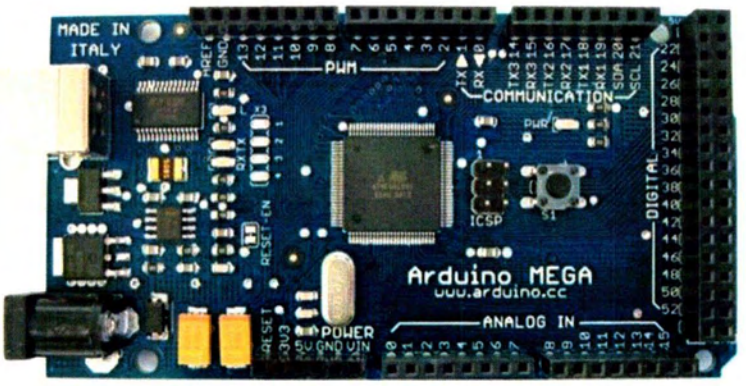


Figure 13. Arduino

6.1.1 Arduino Overview

Arduino [11] (*Figure 13*) is a single-board microcontroller, intended to make the application of interactive objects or environments more accessible. The hardware consists of an open-source hardware board designed around an 8-bit Atmel AVR microcontroller, or a 32-bit Atmel ARM. Pre-programmed into the on-board microcontroller chip is a boot loader that allows uploading programs into the microcontroller memory without needing a chip (device) programmer. Arduino boards can be purchased pre-assembled or as do-it-yourself kits. Hardware design information is available for those who would like to assemble an Arduino by hand.

6.1.2 Arduino Hardware

An Arduino board consists of an Atmel 8-bit AVR microcontroller with complementary components to facilitate programming and incorporation into other circuits. An important aspect of the Arduino is the standard way that connectors are exposed, allowing the CPU board to be connected to a variety of interchangeable add-on modules known as shields. Some shields communicate with the Arduino board directly over various pins, but many shields are individually addressable via an I^2C serial bus, allowing many shields to be stacked and used in parallel. Official Arduinos have used the megaAVR series of chips, specifically the ATmega8, ATmega168, ATmega328, ATmega1280, and ATmega2560. A handful of other processors have been used by Arduino compatibles. Most boards include a 5 volt linear regulator and a

16 MHz crystal oscillator (or ceramic resonator in some variants), although some designs such as the LilyPad run at 8 MHz and dispense with the onboard voltage regulator due to specific form-factor restrictions. An Arduino's microcontroller is also pre-programmed with a boot loader that simplifies uploading of programs to the on-chip flash memory, compared with other devices that typically need an external programmer. This makes using an Arduino more straightforward by allowing the use of an ordinary computer as the programmer.

At a conceptual level, when using the Arduino software stack, all boards are programmed over an RS-232 serial connection, but the way this is implemented varies by hardware version. Serial Arduino boards contain a level shifter circuit to convert between RS-232-level and TTL-level signals. Current Arduino boards are programmed via USB, implemented using USB-to-serial adapter chips such as the FTDIFT232.

The Arduino board exposes most of the microcontroller's I/O pins for use by other circuits. The Diecimila, Duemilanove, and current Uno provide 14 digital I/O pins, six of which can produce pulse-width modulated signals, and six analog inputs. These pins are on the top of the board, via female 0.10-inch (2.5 mm) headers. Several plug-in application shields are also commercially available.

There are many Arduino-compatible and Arduino-derived boards. Some are functionally equivalent to an Arduino and may be used interchangeably. Many are the basic Arduino with the addition of commonplace output drivers, often for use in school-level education to simplify the construction of buggies and small robots. Others are electrically equivalent but change the form factor, sometimes permitting the continued use of Shields, sometimes not. Some variants use completely different processors, with varying levels of compatibility.

6.1.3 Arduino Software

The Arduino integrated development environment (IDE) (*Figure 14*) is a cross-platform application written in Java, and is derived from the IDE for the Processing programming language and the Wiring projects. It includes a code editor with features such as syntax highlighting, brace matching, and automatic indentation, and is also capable of compiling and uploading programs to the board with a single click. A program or code written for Arduino is called a "sketch". Arduino programs are written in C or C++. The Arduino IDE comes with a software library called "Wiring" from the original Wiring project, which makes many common input/output operations much easier. Users only need define two functions to make a runnable cyclic executive program:

- setup(): a function run once at the start of a program that can initialize settings
- loop(): a function called repeatedly until the board powers off

A typical first program for a microcontroller simply blinks an LED on and off. In the Arduino environment, the user might write a program like this:

```
#define LED_PIN 13
```

```
void setup () {
  pinMode (LED_PIN, OUTPUT); // Enable pin 13 for digital output
}

void loop () {
  digitalWrite (LED_PIN, HIGH); // Turn on the LED
  delay (1000); // Wait one second (1000 milliseconds)
  digitalWrite (LED_PIN, LOW); // Turn off the LED
  delay (1000); // Wait one second
}
```

It is a feature of most Arduino boards that they have an LED and load resistor connected between pin 13 and ground. The previous code would not be seen by a standard C++ compiler as a valid program, so when the user clicks the "Upload to I/O board" button in the IDE, a copy of the code is written to a temporary file with an extra include header at the top and a very simple main() function at the bottom, to make it a valid C++ program.

The Arduino IDE uses the GNU tool chain and AVR Libc to compile programs, and uses avrdude to upload programs to the board.

As the Arduino platform uses Atmel microcontrollers, Atmel's development environment, AVR Studio or the newer Atmel Studio, may also be used to develop software for the Arduino.

Figure 14. Arduino's IDE

6.2 SparkFun's pro Micro 3.3V/8MHz

SparkFun's Pro Micro [12] (*Figure 15*) is an Arduino-compatible microcontroller. It comes with an ATmega32U4 microcontroller. The USB transceiver inside the 32U4 allows us to add USB connectivity on-board and do away with bulky external USB interface.

This tiny little board does all of the neat-o Arduino tricks : 4 channels of 10-bit ADC, 5 PWM pins, 12 DIOs as well as hardware serial connections Rx and Tx. It is also running at 8MHz and 3.3V.

Features :

- ATMega 32U4 running at 3.3V/8MHz
- Supported under Arduino IDE v1.0.1
- On-Board micro-USB connector for programming
- 4 x 10-bit ADC pins
- 12 x Digital I/Os (5 are PWM capable)
- Rx and Tx Hardware Serial Connections
- Smallest Arduino-Compatible Board Yet, with Dimensions: 1.3x0.7"

Figure 15. SparkFun's pro Micro

6.3 Processing

Processing [13] is an open source programming language and integrated development environment (IDE) (*Figure 16*) built for the electronic arts, new media art, and visual design communities with the purpose of teaching the fundamentals of computer programming in a visual context, and to serve as the foundation for electronic sketchbooks.

Processing includes a sketchbook, a minimal alternative to an integrated development environment (IDE) for organizing projects.

Every Processing sketch is actually a subclass of the PApplet Java class which implements most of the Processing language's features.

When programming in Processing, all additional classes defined will be treated as inner classes when the code is translated into pure Java before compiling. This means that the use of static variables and methods in classes is prohibited unless you explicitly tell Processing that you want to code in pure Java mode.

Processing also allows for users to create their own classes within the PApplet sketch. This allows for complex data types that can include any number of arguments and avoids the limitations of solely using standard data types such as: int (integer), char (character), float (real number), and color (RGB, ARGB, hex).

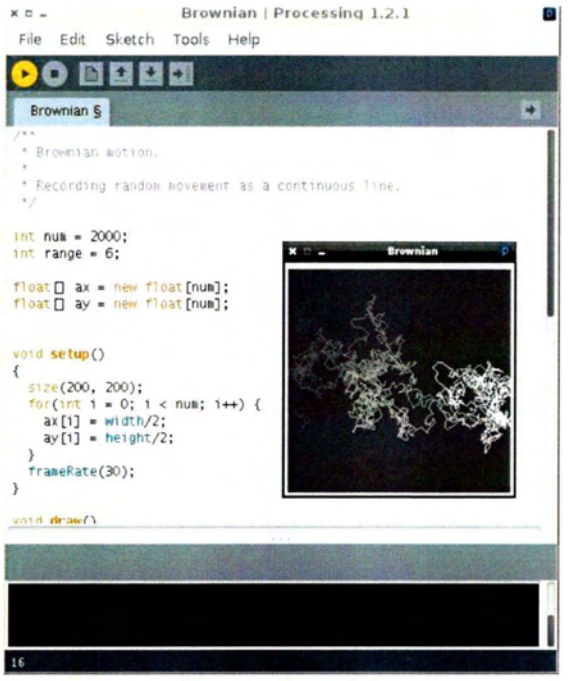


Figure 16. Processing

6.3.1 Processing's Serial library

Processing's Serial library (*Figure 17*) is for reading and writing data to and from external devices one byte at a time. It allows two computers to send and receive data. This library has the flexibility to communicate with custom microcontroller devices and to use them as the input or output to Processing programs. The serial port is a nine pin I/O port that exists on many PCs and can be emulated through USB.

We use this Processing's library in order to visualize the measurements gathered by the CYWM6935 radio module over the 2.4GHz band.

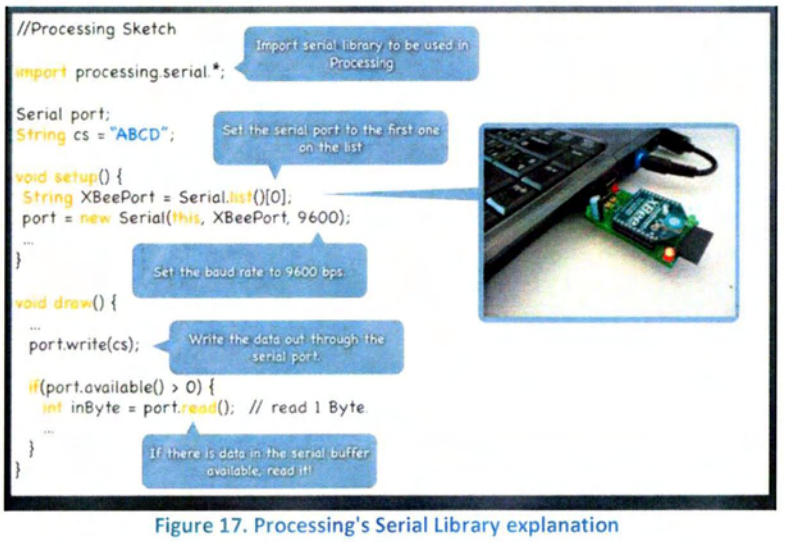


Figure 17. Processing's Serial Library explanation

6.4 CYWM6935

The Cypress CYWM6935 LRTM Radio Module [14] (*figure 18*) is a device that transmits and receives radio signals in accordance with the spectrum regulations for the 2.4-GHz unlicensed frequency range. This chip is designed to operate over the 2.4GHz band and has the ability to listen on a frequency for any other devices that may be already using the frequency. This is a complete low power radio transmitter/receiver chip for the 2.4GHz band and it is needed to be controlled by a microcontroller over a synchronous serial (SPI) interface. The microcontroller can write to various registers in the chip to set things like operating frequency and can read other registers to retrieve data from the chip.

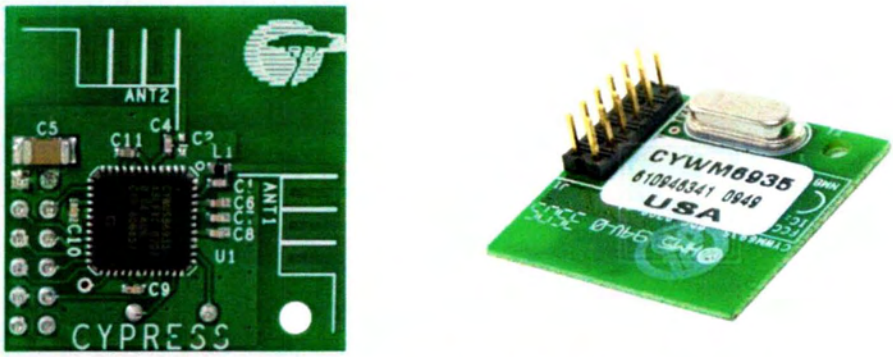


Figure 18. Cypress CYWM6935 Radio Module

Key Features:

- The CYWM6935 LR(TM) 2.4-GHz DSSS Radio SoC Module includes radio (CYWUSB6935) , antenna, and all external components
- Complete Radio Module with Dual PCB Trace Antennas
- Operates in the unlicensed Industrial, Scientific, and Medical (ISM) band (2.4 GHz-2.483 GHz)
- -95-dBm receive sensitivity
- Up to 0-dBm output power
- Range of up to 50 meters or more
- Data throughput of up to 62.5 kbits/sec
- SPI microcontroller interface (up to 2 MHz data rate)
- Operating voltage from 2.7V to 3.6V

6.4.1 Pin Signals

The CYWM6935 is available in a small PCBA design and can be mounted horizontally to the device PCB via a 12-pin header. The pin-out of the header is shown in (*figure 19*)

Pin QFN	Name	Direction	Description
1	GND	–	Ground
2	VCC	–	Supply voltage for the entire Radio Module (2.7 V-3.6 V). It is recommended that 3.3 V be used for most applications.
3	IRQ	Output	Interrupt signal from Radio Module to the MCU
4	nRESET	Input	Active low reset signal from MCU to Radio Module
5	MOSI	Input	Master out, slave in SPI signal from MCU to Radio Module
6	nSS	Input	Active low slave select signal from MCU to Radio Module
7	SCK	Input	SPI clock from MCU to Radio Module
8	MISO	Output	Master in, slave out SPI signal from Radio Module to MCU
9	GND	–	Ground
10	nPD	Input	Active low power-down signal from MCU to Radio Module
11	N/C	–	No connect—leave open
12	N/C	–	No connect—leave open

Figure 19. Pin Signals of Cypress CYWM6935 radio module

6.4.2 Developed prototype, 2.4-GHz spectrum analyzer with Cypress CYWM6935

The implementation [15], [16], [17] regards the development of a 2.4-GHz spectrum analyzer using the Cypress CYWM6935 radio transceiver module. The goal was to scan the 2.4 band in order to estimate the traffic conditions on each frequency.

The CYWM6935 chip's main purpose in life is the transmit/receive functions. However, it is adequate to indicate the signal strength in each frequency of the 2.4GHz band as a number typically up to 30, with zero representing no signal. It can detect signals made by any device which use the 2.4GHz band. The CYWM6935 is able to communicate and disseminate the recorded values to a microcontroller through its SPI port. This info can be later used to evaluate the congestion on each channel of the 2.4GHz band as well as to discover the most uncongested channel in which a WSN can efficiently operate.

6.4.2.1 CYWM6935 and Arduino Leonardo

Arduino Leonardo uses Atmega32U4. It incorporates and supports at the same time two serial ports, which is an important issue in our implementation.

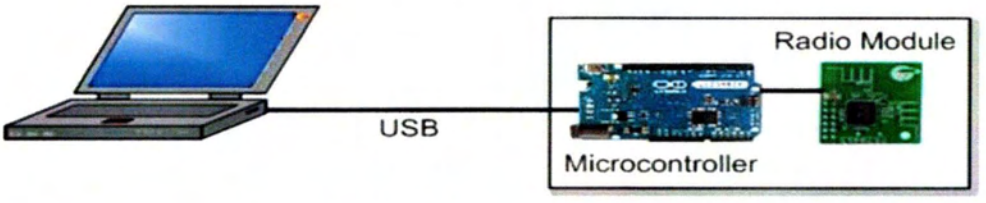


Figure 20. Arduino & Cypress CYWM6935 radio module

There is one snag here. The Arduino Leonardo is working at 5V. The CYWM6935 is working at 3.3V as mentioned above. In case of a direct connection between the CYWM6935 module and the Leonardo board there is a possibility of damaging the radio module. This is the case both for the direct power supply to the radio module, but also for the SPI bus. So all voltages to the radio module need to be converted to 3.3V. For the SPI can be used a simple voltage divider (*Figure 21*).

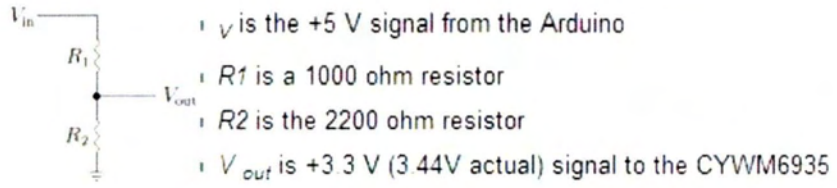


Figure 20. 5V to 3.3 Voltage Divider

A voltage divider divides the input voltage over two resistors, where part of the voltage goes to ground, and the other part is used as a (lower) output voltage, to a next part of the circuit. The ratio between the two resistors determines the output voltage.

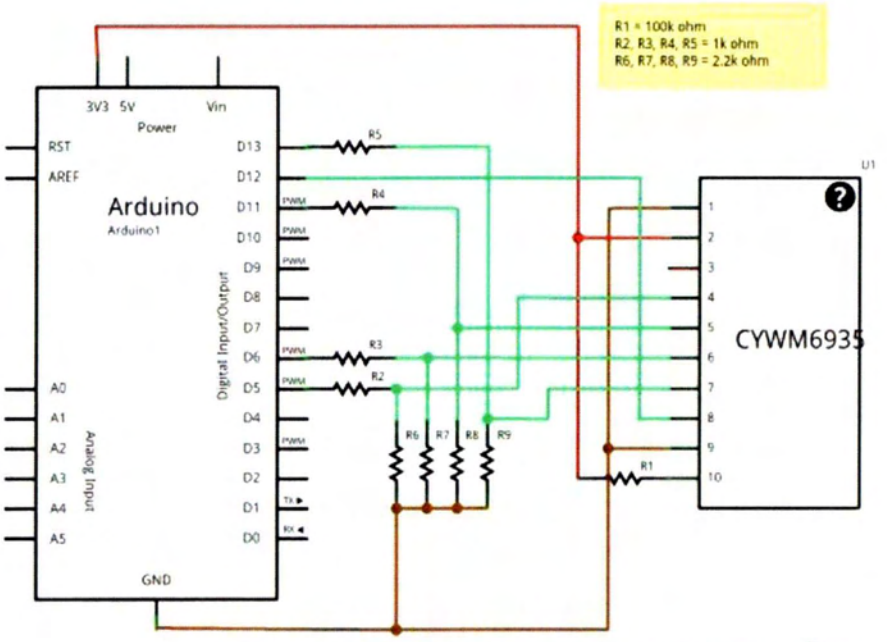


Figure 21. Arduino & CYWM6935 connection using a voltage divider

However, this solution is not suitable to feed the module because the power is too limited by the resistance values. As a result the module was unstable.

6.4.2.2 CYWM6935 and SparkFun's pro Micro

The need of usage 3.3V not only to feed the module but also for the SPI bus led us to use SparkFun's pro Micro. As mentioned before it is a tiny, Arduino-like device compatible with Arduino IDE. The SparkFun's pro Micro is working at 3.3V which is ideal for communication with CYWM6935 module without the need of voltage converters (*Figure 22*).

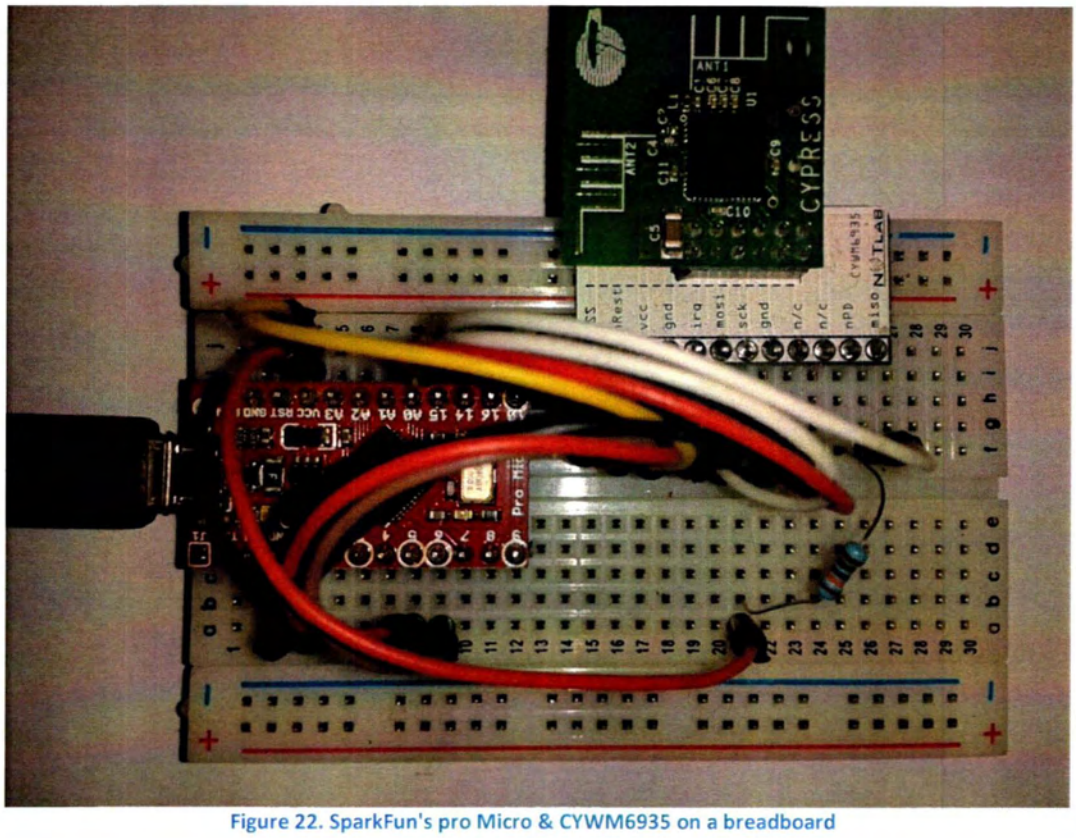


Figure 22. SparkFun's pro Micro & CYWM6935 on a breadboard

We program the SparkFun's pro Micro using the Arduino's IDE, Arduino's SPI Library and an Arduino's Cypress CYWM6935 Library. The Arduino code initializes the serial communication between SparkFun pro Micro and CYWM6935 radio via the SPI Library. Pro Micro is programmed to instruct the module to step to a frequency from 2,4GHz to 2.5GHz and measure the signal level at that frequency. The radio is checking the signal strength for each 1MHz twice so as to take the signal strength peak and to have more reliable measures avoiding any noise. In other words the pro Micro sets the radio chip to a frequency, reads the signal level from the chip twice, stores the value in its internal memory and steps on to the next frequency. This continues until the complete 2.4GHz band is covered.

After collecting the above measurements, the pro micro applies a simple algorithm to find the most uncongested 802.15.4 channel. The pro Micro then sends the data off to computer using USB. It sends both the received signal strengths from the radio and the "best" channel. Using the Processing's Serial library we are not only able to visualize the data we obtain from the radio module at PC but also to have knowledge of the most uncongested 802.15.4 channel. This information will be promoted later to a wireless sensor network.

6.5 XBee

The XBee/XBee-PRO ZB RF Modules [18] are designed to operate within the ZigBee protocol and support the unique needs of low-cost, low-power wireless sensor networks. The modules require minimal power and provide reliable delivery of data between remote devices. XBee modules operate within the ISM 2.4 GHz frequency band.

Key Features:

High Performance, Low Cost

- XBee
  - Indoor/Urban: up to 133' (40 m)
  - Outdoor line-of-sight: up to 400' (120 m)
  - Transmit Power: 2 mW (3 dBm)
  - Receiver Sensitivity: -96 dBm
- XBee-PRO (S2)
  - Indoor/Urban: up to 300' (90 m), 200' (60
  - m) for International variant
  - Outdoor line-of-sight: up to 2 miles (3200
  - m), 5000' (1500 m) for International variant
  - Transmit Power: 50mW (17dBm), 10mW
  - (10dBm) for International variant
  - Receiver Sensitivity: -102 dBm
  - Receiver Sensitivity: -102 dBm

Advanced Networking & Security

- Retries and Acknowledgements
- DSSS (Direct Sequence Spread Spectrum)
- Each direct sequence channel has over 65,000 unique network addresses available
- Point-to-point, point-to-multipoint and peer-to-peer topologies supported
- Self-routing, self-healing and fault-tolerant
- mesh networking

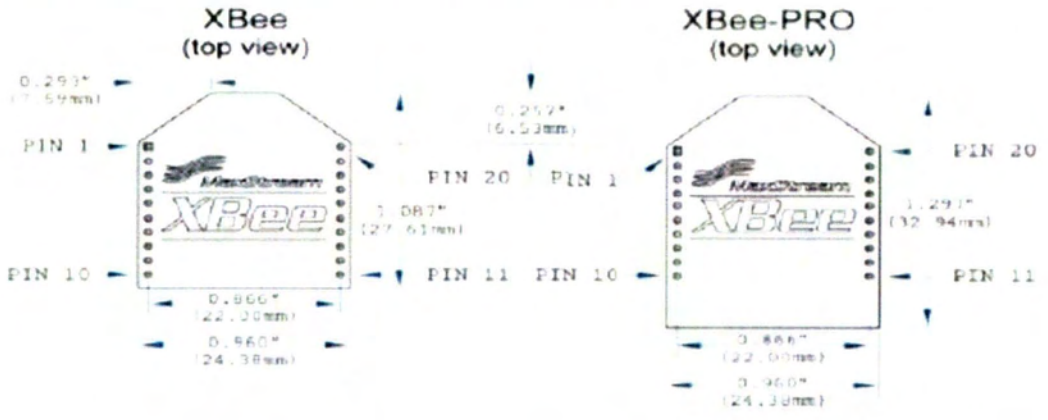
Low Power

- XBee
  - TX Peak Current: 40 mA (@3.3 V)
  - RX Current: 40 mA (@3.3 V)
  - Power-down Current: < 1 µA
- XBee-PRO (S2)
  - TX Peak Current: 295mA (170mA for
  - international variant)
  - RX Current: 45 mA (@3.3 V)
  - Power-down Current: 3.5 µA typical
  - @ 25 degrees C

Easy-to-Use

- No configuration necessary for out-of box
- RF communications
- AT and API Command Modes for configuring module parameters
- Small form factor
- Extensive command set
- Free X-CTU Software

6.5.1 Pin Signals and Recommended Pin Connections



Pin #	Name	Direction	Default State	Description
1	VCC	-	-	Power supply
2	DOUT	Output	Output	UART Data Out
3	DIN / $\overline{\text{CONFIG}}$	Input	Input	UART Data In
4	DIO12	Both	Disabled	Digital I/O 12
5	$\overline{\text{RESET}}$	Both	Open-Collector with pull-up	Module Reset (reset pulse must be at least 200 ns)
6	RSSI PWM / DIO10	Both	Output	RX Signal Strength Indicator / Digital IO
7	DIO11	Both	Input	Digital I/O 11
8	[reserved]	-	Disabled	Do not connect
9	$\overline{\text{DTR}}$ / SLEEP_RQ/ DIO8	Both	Input	Pin Sleep Control Line or Digital IO 8
10	GND	-	-	Ground
11	DIO4	Both	Disabled	Digital I/O 4
12	$\overline{\text{CTS}}$ / DIO7	Both	Output	Clear-to-Send Flow Control or Digital I/O 7. CTS, if enabled, is an output.
13	ON / $\overline{\text{SLEEP}}$	Output	Output	Module Status Indicator or Digital I/O 9
14	VREF	Input	-	Not used for EM250. Used for programmable secondary processor. For compatibility with other XBEE modules, we recommend connecting this pin voltage reference if Analog sampling is desired. Otherwise, connect to GND.
15	Associate / DIO5	Both	Output	Associated Indicator, Digital I/O 5
16	$\overline{\text{RTS}}$ / DIO6	Both	Input	Request-to-Send Flow Control, Digital I/O 6. RTS, if enabled, is an input.
17	AD3 / DIO3	Both	Disabled	Analog Input 3 or Digital I/O 3
18	AD2 / DIO2	Both	Disabled	Analog Input 2 or Digital I/O 2
19	AD1 / DIO1	Both	Disabled	Analog Input 1 or Digital I/O 1
20	AD0 / DIO0 / Commissioning Button	Both	Disabled	Analog Input 0, Digital IO 0, or Commissioning Button

Figure 23. XBee's pin Signals

The only required pin connections are VCC, GND, DOUT and DIN. To support serial firmware updates, VCC, GND, DOUT, DIN, RTS, and DTR should be connected. All unused pins should be left disconnected. All inputs on the radio can be pulled high with 30k internal pull-up resistors using the PR software command. No specific treatment is needed for unused outputs. For applications that need to ensure the lowest sleep current, inputs should never be left floating. Use internal or external pull-up or pull-down resistors, or set the unused I/O lines to outputs. Other pins may be connected to external circuitry for convenience of operation including the Associate LED pin (pin 15) and the Commissioning pin (pin 20).

6.5.2 Serial Buffers

The XBee modules maintain small buffers (*Figure 24*) to collect received serial and RF data, which is illustrated in the figure below. The serial receive buffer collects incoming serial characters and holds them until they can be processed. The serial transmit buffer collects data that is received via the RF link that will be transmitted out the UART.

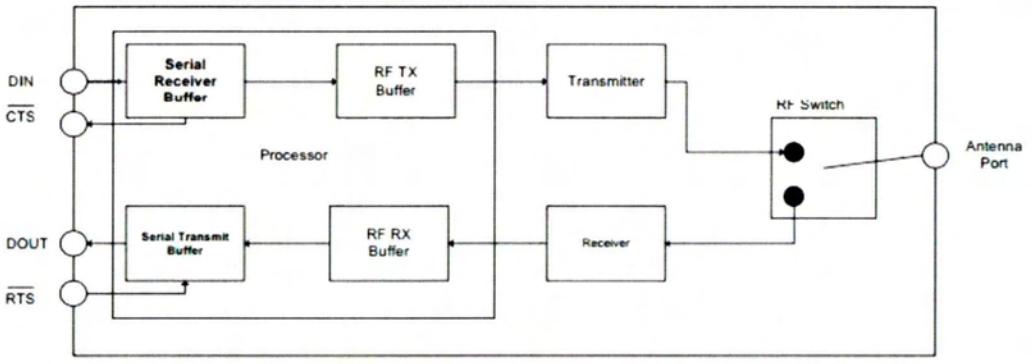


Figure 24. XBee's buffers

6.5.2.1 Serial receive buffer

When serial data enters the RF module through the DIN Pin (pin 3), the data is stored in the serial receive buffer until it can be processed. Under certain conditions, the module may not be able to process data in the serial receive buffer immediately. If large amounts of serial data are sent to the module, CTS flow control may be required to avoid overflowing the serial receive buffer.

6.5.2.2 Serial Transmit Buffer

When RF data is received, the data is moved into the serial transmit buffer and sent out the UART. If the serial transmit buffer becomes full enough such that all data in a received RF packet won't fit in the serial transmit buffer, the entire RF data packet is dropped.

6.5.3 Modes of Operation

6.5.3.1 Idle Mode

When not receiving or transmitting data, the RF module is in Idle Mode. The module can shift into the other modes of operation.

6.5.3.2 Transmit Mode

When serial data is received and is ready for packetization, the RF module will exit Idle Mode and attempt to transmit the data. The destination address determines which node(s) will receive the data. Prior to transmitting the data, the module ensures that a 16-bit network address and route to the destination node have been established.

6.5.3.3 Receive Mode

If a valid RF packet is received, the data is transferred to the serial transmit buffer.

6.5.3.4 Command Mode

To modify or read RF Module parameters, the module must first enter into Command Mode - a state in which incoming serial characters are interpreted as commands.

6.5.4 Device Types

A coordinator has the following characteristics: it

- Selects a channel and PAN ID (both 64-bit and 16-bit) to start the network
- Can allow routers and end devices to join the network
- Can assist in routing data
- Cannot sleep--should be mains powered
- Can buffer RF data packets for sleeping end device children

A router has the following characteristics: it

- Must join a ZigBee PAN before it can transmit, receive, or route data
- After joining, can allow routers and end devices to join the network
- After joining, can assist in routing data
- Cannot sleep--should be mains powered.
- Can buffer RF data packets for sleeping end device children

An end device has the following characteristics: it

- Must join a ZigBee PAN before it can transmit or receive data
- Cannot allow devices to join the network
- Must always transmit and receive RF data through its parent. Cannot route data.
- Can enter low power modes to conserve power and can be battery-powered

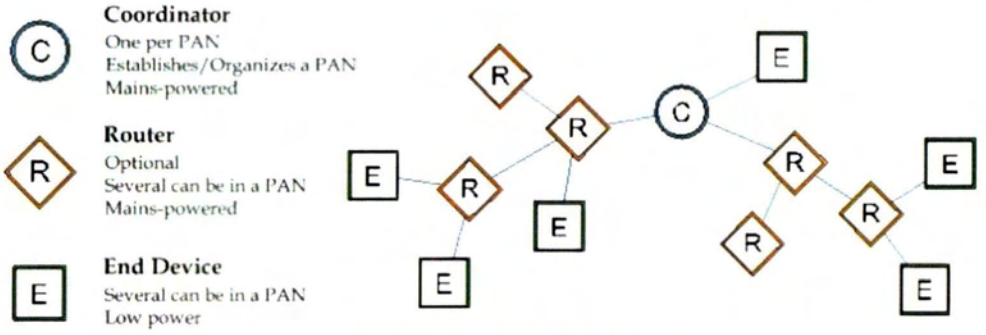


Figure 25. XBee's Device Types

6.5.4.1 PAN ID

ZigBee networks are called personal area networks or PANs. Each network is defined with a unique PAN identifier (PAN ID). This identifier is common among all devices of the same network. ZigBee devices are either preconfigured with a PAN ID to join, or they can discovery nearby networks and select a PAN ID to join.

6.5.4.2 Channel scanning

Routers and end devices must scan one or more channels to discover a valid network to join. When a join attempt begins, the XBee sends a beacon request transmission on the lowest channel specified in the SC (scan channels) command bitmask. If a valid PAN is found on the channel, the XBee will attempt to join the PAN on that channel. Otherwise, if a valid PAN is not found on the channel, it will attempt scanning on the next higher channel in the SC command bitmask. The XBee will continue to scan each channel (from lowest to highest) in the SC bitmask until a valid PAN is found or all channels have been scanned. Once all channels have been scanned, the next join attempt will start scanning on the lowest channel specified in the SC command bitmask. For example, if the SC command is set to 0x400F, the XBee would start scanning on channel 11 (0x0B) and scan until a valid beacon is found, or until channels 11, 12, 13, 14, and 25 have been scanned (in that order). SC parameter could also be specified for a single channel. In this case, the above procedure would not be followed and the devices will be coordinated in this unique channel.

6.5.5 XBee configuration

There are several ways to configure the XBee module. Digi proposes the Windows-based X-CTU tool for XBee configuration. However, if you need fast configuration or just to specify some important parameters, a more flexible and simple solution is to use the USB-serial interface to access to the XBee module with a serial tool such as "CoolTerm", "minicom" on Unix-based machines , "Zterm" on MAC computers or "HyperTerminal" on Windows-based systems machines.

6.5.5.1 CoolTerm

CoolTerm [19] is a simple serial port terminal application (no terminal emulation) that is geared towards hobbyists and professionals with a need to exchange data with hardware connected to serial ports such as servo controllers, robotic kits, GPS receivers, microcontrollers, etc. CoolTerm is an easy serial port terminal application in order to setup a basic communication with two XBee interfaces and to be more familiar with these modules. Using a serial connection from your computer, the text you type to one XBee will be wirelessly transmitted to the other XBee, which will send the text via serial to your other computer.

In order to connect XBee modules to pc via USB, you need USB explorers (*Figure 26*) and FTDI drivers so as to let the computer talk via serial to the board,

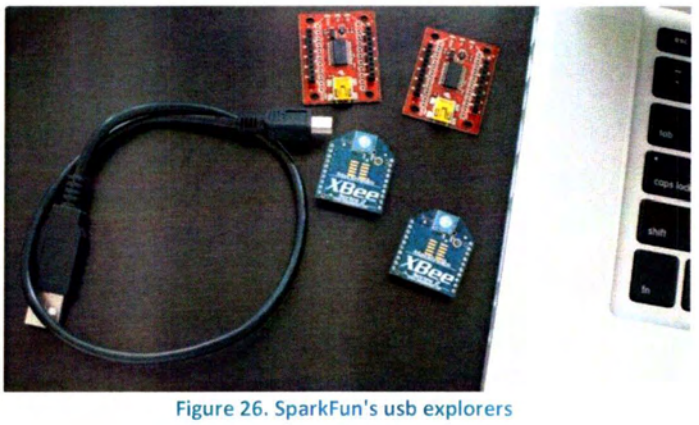


Figure 26. SparkFun's usb explorers

When you're working with XBees, you may need to update or change the firmware on the radios occasionally. For example, if you'd like to switch a ZigBee radio from router to coordinator or switch between API and AT modes, you'll need to upload the appropriate firmware to the radio. At this point there is a need to have 1 XBee with Coordinator At firmware and 1 XBee with Router AT firmware.

Every XBee radio has a 64-bit serial number address printed on the back (*Figure 27*). The beginning or "high" part of the address will be 0013A200, Digi's pre-assigned range of address space. The last or "low" part of the address will be different for every radio. For the radio on the image, it's 403B9E21.

Figure 27. 64-bit serial number address

Following the instructions from [19], we configure our XBee module using CoolTerm.

```
+++
OK
ATID 2001
OK
ATDH 0013A300
OK
ATDL 403B9E21
OK
ATID
2001
ATDH
13A300
ATDL
403B9E21
ATWR
OK
```

Figure 27. Commands example in CoolTerm

- **PAN ID:** PAN stands for Personal Area Network. This is a unique identifier for your network. XBees assigned to a particular PAN ID can only communicate with each other. This lets you set up separate networks in the same location.
- **Destination address high:** This represents the first half of the address we want to talk to. XBee radios can have a 64-bit address, so this is the higher 32-bit part of that address number. Since we don't need so many addresses, we'll set this to 0 and only use the low setting.
- **Destination address low:** This is the address we'll use to locate the other XBee. Make sure it matches the ATMY setting of the XBee you want to talk to.

On the Router AT side:

Follow the same steps as before to program the second radio. Set the destination address to the coordinator radio's address. If everything is set up properly, the text that you type in the serial terminal program on the first computer will be relayed to the second computer and appear on its serial terminal screen as well.

6.5.5.2 X-CTU

X-CTU is a Windows-based application provided by Digi. This program was designed to interact with the firmware files found on Digi's RF products and to provide a simple-to-use graphical user interface to them. Just like on the CoolTerm, there is a need to connect the xbees modules with usb explorers in order to establish communication between the modules and the PC through serial.

After opening X-CTU, at the first tab you have to select the device you wish to interact with (*Figure 28*).

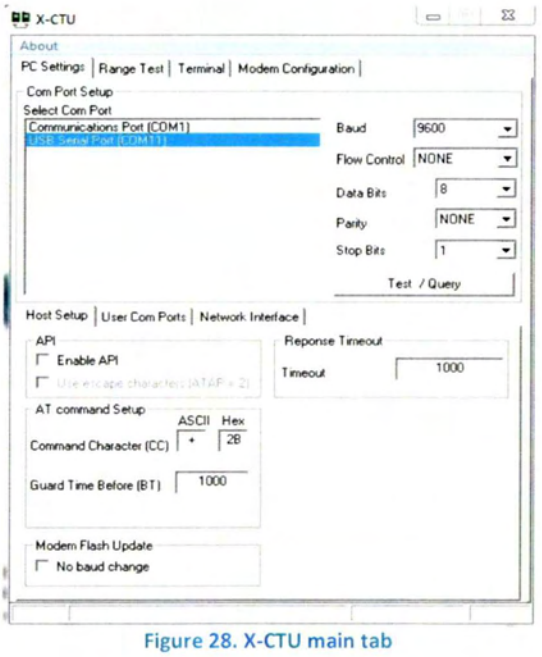


Figure 28. X-CTU main tab

The Test / Query button is used to test the selected COM port and PC settings. If the settings and COM port are correct, you will receive a response similar to the one depicted in *Figure 29* below.

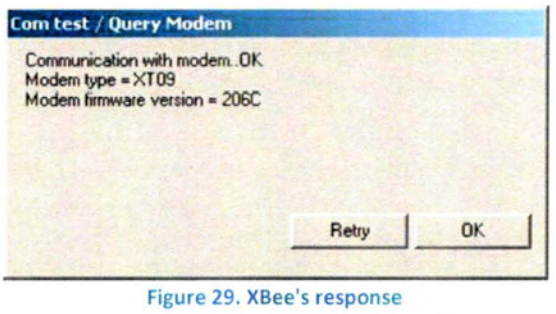


Figure 29. XBee's response

The Modem configuration tab (*Figure 30*) has to provide a Graphical User Interface with a radio's firmware in order to Read and Write firmware to the radio's microcontroller based on your selected parameters.

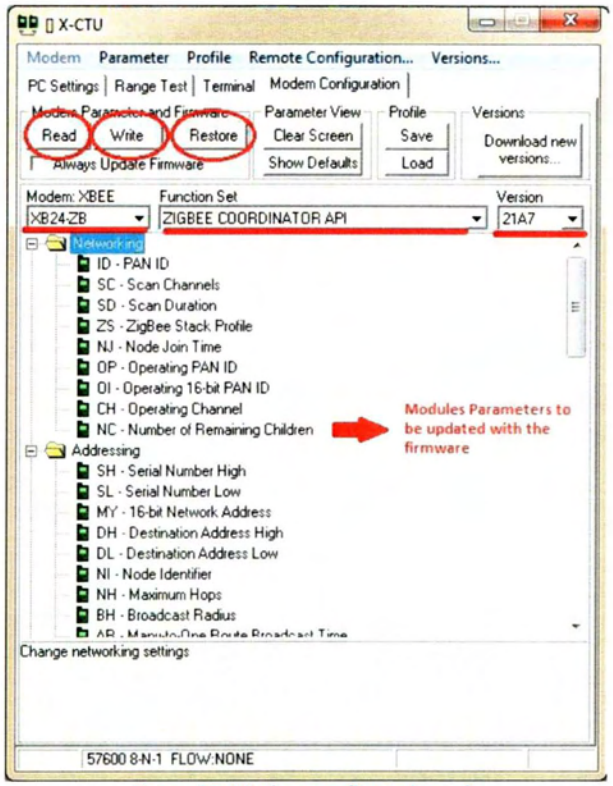


Figure 30. Modem configuration tab

- Read: in order to read the radio's firmware
- Write: in order to write the radio's firmware
- Restore: in order to restore the radio's default firmware
- Modem: XBee :in order to select your modem type
- Function Set: in order to select Coordinator, Router, End Device functionality.
- User-settable parameters: in order to change parameters to be updated with the firmware.

Based on the above example in which we use CoolTerm terminal application for configuring the XBee modules we have to do the following in order o achieve the same results using X-CTU:

- Open X-CTU
- Select the proper device we wish to update firmware to
- Go to Modem Configuration tab
- Click on read button so as to read the module's firmware
- Select one Coordinator AT and one Router AT at Function Set
- Change the values of ID, DH and Dl user-settable parameters(as the Coolterm example)
- Click write button in order to update firmware

6.5.6 XBee and Arduino

Configuring the XBee modules in the above way, even though it may establish communication between them, we can't get extra functionality on the modules. A good option is to combine the XBee modules with Arduino in order to obtain and take advantage of additional characteristics. This way, it is now possible to configure the XBee modules programming and requesting from the Arduino to provide the desired configuration. There is no need of USB explorers and applications like X-CTU anymore. Of course there is still need for some basic parameters initialization and firmware update the first time you use them. Thereafter the Arduino is able to promote any parameter configuration on the modules and save this configuration in order to be used by the modules. Another advantage of the above combination is that the Arduino can instruct now the xbee modules to send or receive a certain number of packets. Subsequently, Arduino can store various measurements and statistics in his memory. Good news is that it can control in this way the complete number and rate of sending packets. Also it can measure successful packets, latency and bandwidth of the network.

6.5.7 Arduino's XBee Library

There is an Arduino library for communicating with XBees in API mode [21], with support for both Series 1 (802.15.4) and Series 2 (ZB Pro/ZNet). This library Includes support for the majority of packet types, including: TX/RX, AT Command, Remote AT, I/O Samples and Modem Status.

The XBee class is the primary interface for communicating with an XBee radio and a gateway to the wireless network. The XBee class coordinates sending and receiving data, via packets. An instance of the XBee class is created as follows:

```
XBee xbee = XBee();
```

This class provides methods for sending and receiving packets with an XBee radio via the serial port. The radio must be configured in API (packet) mode (AP=2) in order to use this software. The code is designed to run on a microcontroller, with only one thread.

This software communicates with XBee radios through serial communication, so you will need to connect an XBee radio (typically a coordinator) to your computer's serial port. In other words you have to connect your XBee module on a microcontroller (SparkFun's pro Micro) and then the whole platform on your computer's serial port. The xbee.begin(BaudRate) method must be called in Arduino's setup() method to establish a serial connection to the arduino's rx,tx pins and the XBee module. The argument is baud rate. The default baud rate is 9600, unless it was changed it. For example, this opens a connection between arduino and XBee at 9600 baud:

```
xbee.begin(9600);
```

6.5.7.1 API mode

In API mode, we communicate with XBee radios by sending and receiving packets. The types of packets that can be sent and received depend on the type of XBee radio you are using: Series 1 or Series 2, and the firmware version. In general, there are two types of packets: Transmit and Receive. Transmit packets are sent to the XBee radio, and Receive packets are received from the XBee radio. All Transmit packets extend the XBeeRequest class, and similarly all Receive packets extend the XBeeResponse class, either directly or through intermediate classes. Each Transmit and Receive packet is identified by a unique API ID.

6.5.7.2 Sending Packets

Void send(XBeeRequest &request)

Sends a XBeeRequest (TX packet) out the serial port. The send method is Synchronous due to arduino has only one thread. This method matches a response to the request with the frame id. For that reason is a good idea to chose a unique frame id when using this method. Additionally this method is only allowed for requests that return a response. As a validation, the method checks the frame id of the request. If the frame id is not greater than zero, the request will not return a response and an exception is thrown.

6.5.7.3 Receiving Packets

The method call to getResponse() will block (wait) indefinitely until a response is received. For this reason, it is sometimes a good idea to use a timeout because if a packet is never received, you would be waiting forever. For example, some requests, such as Remote AT, do not return a response if the remote radio is off or out of range.

All methods for receiving packets return an instance of XBeeResponse, the super class of all Receive packets. This class must be "cast" into the appropriate subclass in order to access the packet specific data. To do this you can check the API ID to determine the appropriate class:

```
xbee.send(new AtCommand("NT"));

XBeeResponse response = xbee.getResponse();

if (response.getApiId() == ApiId.AT_RESPONSE) {
  // since this API ID is AT_RESPONSE, we know to cast to AtCommandResponse
   AtCommandResponse atResponse = (AtCommandResponse) response;

   if (atResponse.isOk()) {
      // command was successful
      System.out.println("Command returned " +
ByteUtils.toBase16(atResponse.getValue()));
   } else {
      // command failed!
   }
}
```

6.5.7.4 Packet Delivery Acknowledgement (ACK)

XBees provide a delivery confirmation, or ACK feature, that indicates if a packet was received by a remote XBee. This feature only works with unicast packets, not broadcast. To enable ACK, you must specify a frame ID greater than zero, and between 1 and 255, inclusive. By default, this software uses a frame id of 1. Secondly you must specify the packet as unicast (default setting).

Here's an example of sending a Series 1 packet and receiving the ACK:

```
// create a unicast packet to be delivered to remote radio with 16-bit address: B071,
with payload "Hi"
TxRequest16 request = new TxRequest16(new XBeeAddress16(0xb0, 0x71), new
int[] {'H','i'});

// send the packet and wait up to 12 seconds for the transmit status reply (you should
really catch a timeout exception)
TxStatusResponse response = (TxStatusResponse) xbee.send (request);

if (response.isSuccess()) {
      // packet was delivered successfully
} else {
      // packet was not delivered
      System.out.println("Packet was not delivered.  status: " + response.getStatus());
}
```

Note: While receiving a successful ACK response is a guarantee the packet was delivered, not receiving a successful ACK does not always indicate the remote radio did not receive the packet. There is a possibility that the remote radio received the packet, but the ACK response to the originating radio was not successful. This situation is unlikely however is more likely to occur when the radio is operating at the edge of its range.

6.5.7.5 AT and Remote Command

The AT command allows you to query and set the configuration of the XBee connected to your serial port. AT commands are defined by two characters and are supported by both series 1 and series 2 XBees . You can query the value of a command by sending the command without a value. For example:

```
// query the Serial Low Address
AtCommand  at = new AtCommand("SL")
```

To set the value of a command, specify the value as the second argument. The value must be either an int:

```
// set D2 digital input
AtCommand  at = new AtCommand("D2", 3);
```

or int (array):

```
// set PAN ID:
AtCommand  at = new AtCommand("ID", new int[] {0x1a, 0xaa}));
```

To execute the command and get a response, send the request to the radio:

```
// I choose 5 seconds as an arbitrary value for the timeout.  AT commands usually
respond very quickly.
AtCommandResponse response = (AtCommandResponse)
xbee.sendSynchronous(command, 5000);
```

The isOK() method will return true if the command was successful.

```
if (response.isOk()) {
// success
}
```

The Remote AT command allows to send an AT Command to a remote radio; where "remote" means a radio that has joined the network (same PAN ID and channel) of the serially connected XBee.

For series 2 radios, the MY address is read-only, as it is determined by the radio, so you only need to provide the 64-bit address of the remote radio.

```
// this is the SH + SL address of the remote radio
XBeeAddress64 remoteAddress = new XBeeAddress64(0, 0, 0, 0, 0, 0, 0, 0);
RemoteAtRequest request = new RemoteAtRequest(remoteAddress, "NI");
```

6.6 XBee communication using Arduino & on-the-fly channel configuration

We decide to use a platform which basic component is the SparkFun's pro Micro (*Figure 31*), [1]. The key issue of this platform is that SparkFun's pro Micro operates at 3.3V which is ideal for serial communication with XBee modules. Even more SparkFun's pro Micro can support simultaneously two serial communication ports, one for communication with a PC via USB and another one to communicate with the XBee module. At last SparkFun's pro Micro can be programmed in order to instruct the XBee module to operate the desired functionality.

Figure 31. Platform with XBee and pro Micro

As mentioned above, ZigBee based on coexistence techniques provided by IEEE 802.15.4 layers. Assuming that XBee radios are designed to operate within the ZigBee protocol they have the ability to use the Build-in Scanning and Reporting mechanism. This mechanism occurs when the network is formed for the first time. But, interference issues might occur during the network's operation. This fact can lead to collisions and as a result to poor network performance. Towards this direction a dynamic frequency allocation mechanism can provide a more optimize performance of the network.

The SC (*figure 32*) parameter is a 16-bit value with its bit representing a channel. The SC parameter need to be defined in order for XBees to use a particular channel or a range of channels. This parameter used in order to take place the Build-in Scanning and Reporting mechanism. On our implementation, for experimental purposes, the XBee modules are configured to use both a unique channel and a range of channels.

Channel*	SC	Frequency (MHz)	Bitmap	XBee ZNet 2.5/ZB	XBee ZNet 2.5/ZB PRO
B	1	2.405	0000 0000 0000 0001	x	
C	2	2.410	0000 0000 0000 0010	x	x
D	4	2.415	0000 0000 0000 0100	x	x
E	8	2.420	0000 0000 0000 1000	x	x
F	10	2.425	0000 0000 0001 0000	x	x
10	20	2.430	0000 0000 0010 0000	x	x
11	40	2.435	0000 0000 0100 0000	x	x
12	80	2.440	0000 0000 1000 0000	x	x
13	100	2.445	0000 0001 0000 0000	x	x
14	200	2.450	0000 0010 0000 0000	x	x
15	400	2.455	0000 0100 0000 0000	x	x
16	800	2.460	0000 1000 0000 0000	x	x
17	1000	2.465	0001 0000 0000 0000	x	x
18	2000	2.470	0010 0000 0000 0000	x	x
19	4000	2.475	0100 0000 0000 0000	x	
1A	8000	2.480	1000 0000 0000 0000	x	

Figure 32. XBee's SC parameter values

Arduino code have been developed for both the transmitter and receiver module. The transmitter module sends a number of packets, with a specific payload, repeatedly. The receiver module have been programmed to receive that packets and to send back Acknowledgements (ACKs) if the packet have been received successfully in a synchronous operation. Otherwise, it sends an error report. An additional functionality of the sender's code is to communicate with a Processing sketch in order not only to send his collected statistics about the network so as to be visualized, but also be able to obtain the best and less busy ZigBee's channel according to the CYWM6935 radio module recommendation(as mentioned on the CYWM6935 and Arduino section). It is able to gain this information and to form the network accordingly at any time. In other words an end- user can interact with the Processing sketch so as to prompt the XBee modules to be configured to the best channel of the moment according to the CYWM6935 radio module decision and with respect to the functioning of the

network. Last, some of the statistics being sent to Processing are throughput of the data, network's operating channel, total number of packets that was intended to send, number of successful received packets and the number of failed packets.

7 Demo setup

Combining all the above and for demonstration purposes we have set up a simple demo (*figure 33*) where each mote reports periodically its measurements to the Processing software running on PC.

Firstly, we establish communication between CYWM6935 platform and the Processing in order to have a live spectrum analyzer of the 2.4GHz band. Also, this platform, as mentioned on the CYWM6935 section, is able to send to the Processing the most uncongested ZigBee's channel of the 2.4 band.

Then we configure every XBee module using X-CTU and USB explorers. We configure one XBee as Coordinator API=2 and one as a Router API=2. We specify channel 13 as operating frequency channel. This take place setting the SC parameter=0x100. Also, we configure them with PADID=1AAA.

We programmed the Arduinos so as to communicate with Processing too. They have to send network's statistics to Processing and to listen from Processing the less busy ZigBee's channel in order to configure the XBee modules to operate over this channel. The sender sends 1000 packets of 80bytes payload each repeatedly in order to collect network's statistics.

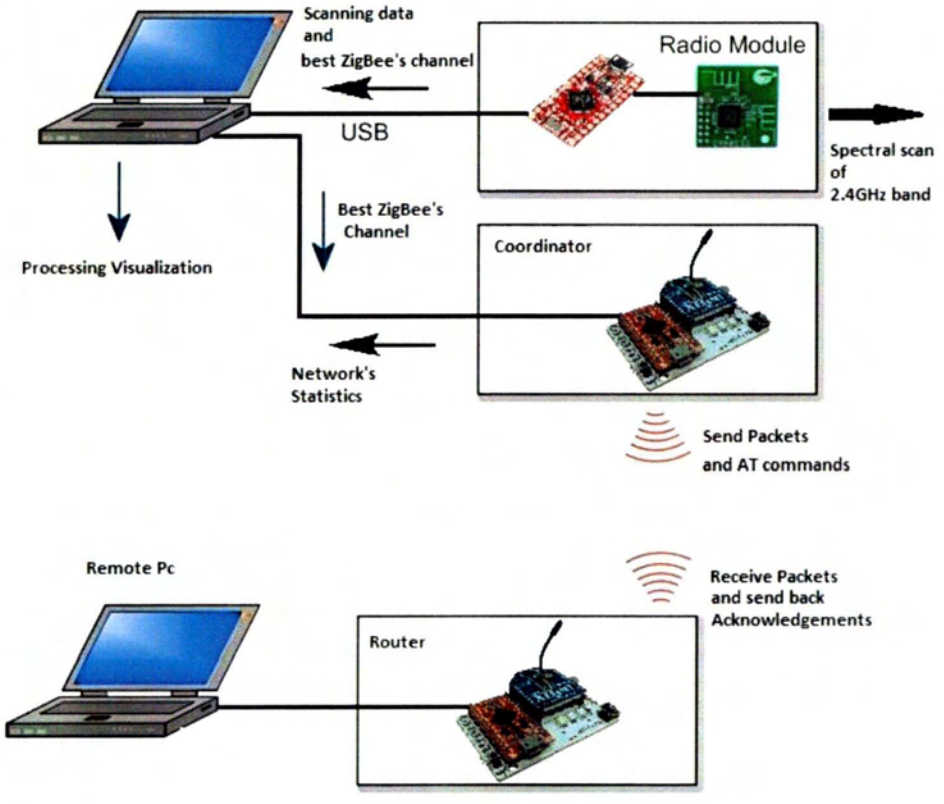


Figure 33. Demo setup

7.1 Experiments and measurements.

Figure 34 illustrates a screenshot taken by the developed software that demonstrates the congestion on each channel. We need to highlight that the channel 1 of the 2.4GHz band is utilized by a Wi-Fi node that performs a video streaming, from a Wi-Fi Router.

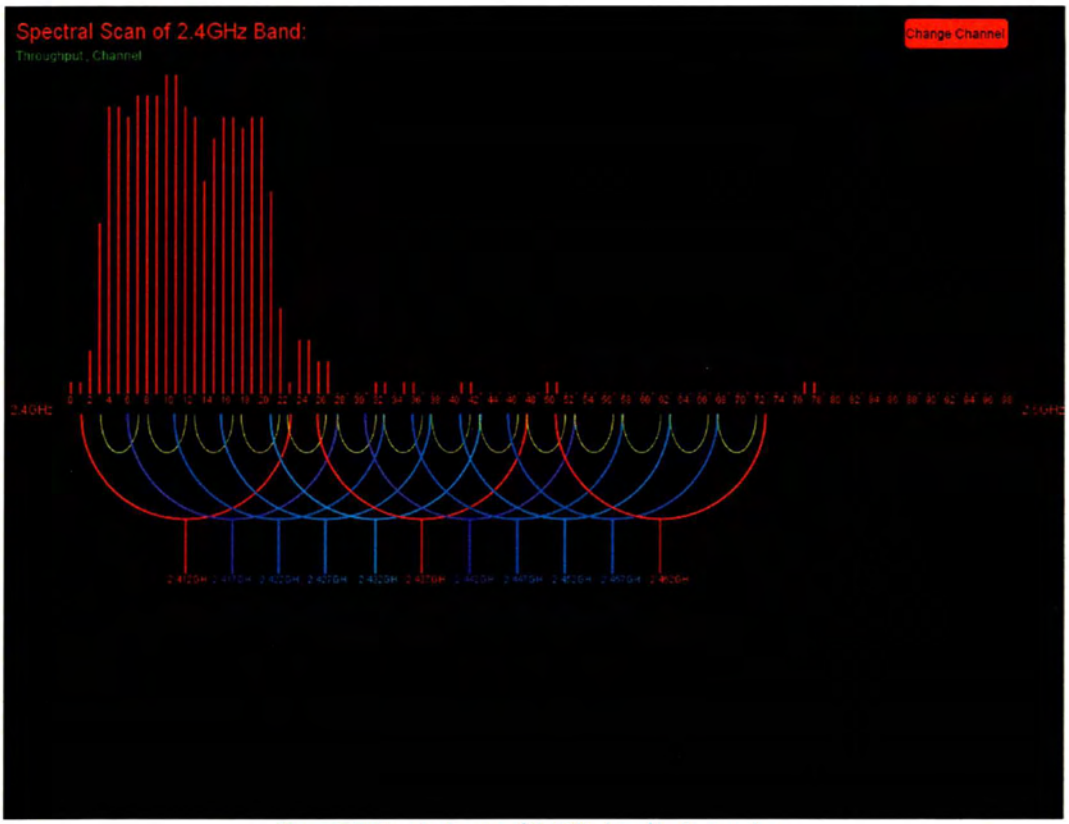


Figure 34. Spectral scan of 2.4GHz band in Processing

Based on artificial loads on Wi-Fi channels we can observe the differentiation between a congested and an uncongested channel.

Figure 35, illustrates the RSSI levels of a channel used by WiFi, another used by a ZigBee network as well as a channel where no transmissions are observed.

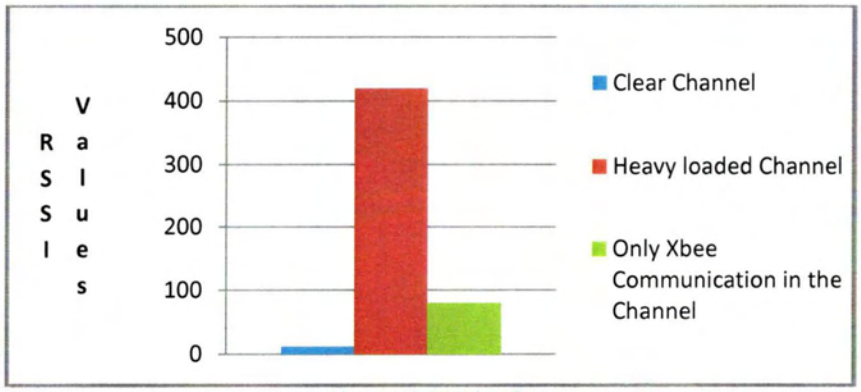


Figure 35. RSSI levels

These measurements are for a unique ZigBee channel. As we refer in the CYWM6935 section, our implementation is able to scan each 1MHz on each ZigBee channel and to report the signal strength in each frequency as a number typically up to 30, with zero representing no signal. We summarized these values for a single channel in conditions of heavy and zero load of the network. Also, we summarized these values when there is only XBee communication in this single channel.

Towards the direction of the mutual coexistence and interference between ZigBee (based on IEEE 802.15.4) and Wi-Fi (based on IEEE 802.11), from the chart below (*Figure 36*) we can see the influence wielded to the throughput of the data.

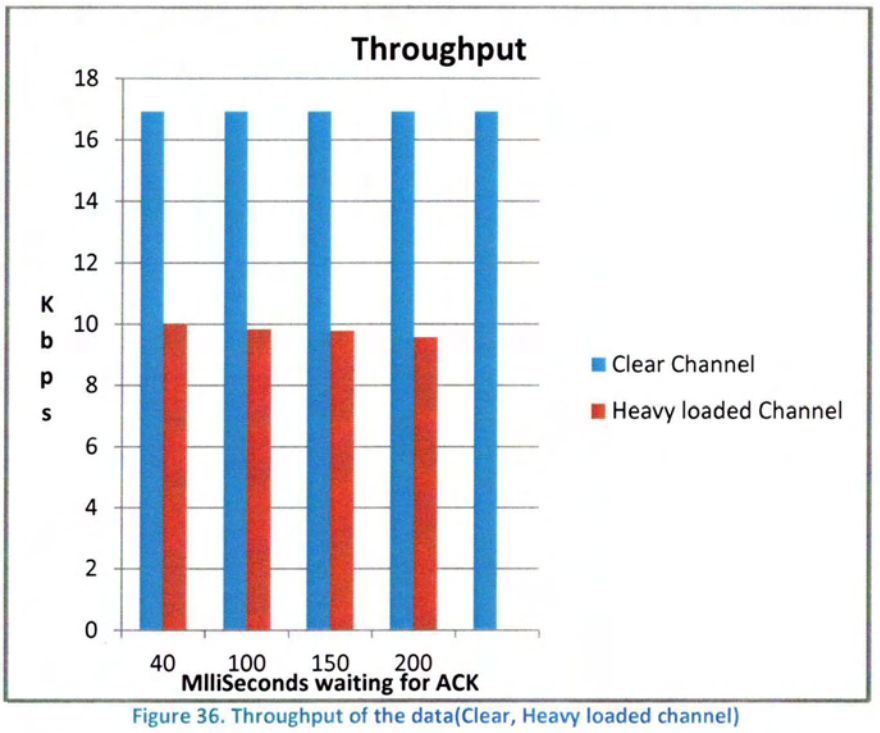


Figure 36. Throughput of the data(Clear, Heavy loaded channel)

Figure 36 shows an extract of experimental results obtained when placing Wi-Fi and ZigBee devices over a unique overlapped channel (Red lines) in addition to a scenario when placing ZigBee devices in an unique uncongested channel and as a result they have zero interference (Blue lines). The horizontal axis refers to the time that the sender is waiting to receive an Acknowledgement. This number affects, as is obvious, the sending rate of packets. The vertical axis refers to the data rate achieved in each scenario. As evidenced the interference from the Wi-Fi affects significantly the performance and throughput of data in a ZigBee network. Wi-Fi's channel is 6 and ZigBee channel is 13.

Why do we care about the time that the sender is waiting to receive an Acknowledgement?

According to our measurements, a single packet transmission with its own Acknowledgement takes 35-37 milliseconds in an uncongested channel. In a clear channel there is no reason to change this time, so we can set it at 40 milliseconds. But when the channel is heavy loaded, each node has to apply Carrier Sense Multiple Access-Collision Avoidance (CSMA-CA) algorithm. According to CSMA-CA, each

node listens the medium prior to transmit. If the energy found higher of a specific level, the transceiver waits during a random time (including in an interval) and tries again. This brings time cost both in sending a packet and an Acknowledgement. As a result assigning a small value, the sender may not receive all the Acknowledgements and thus it has an invalid view of delivered packages. On the other hand, assigning a big value leads to increased delay in case the packet is not successfully delivered.

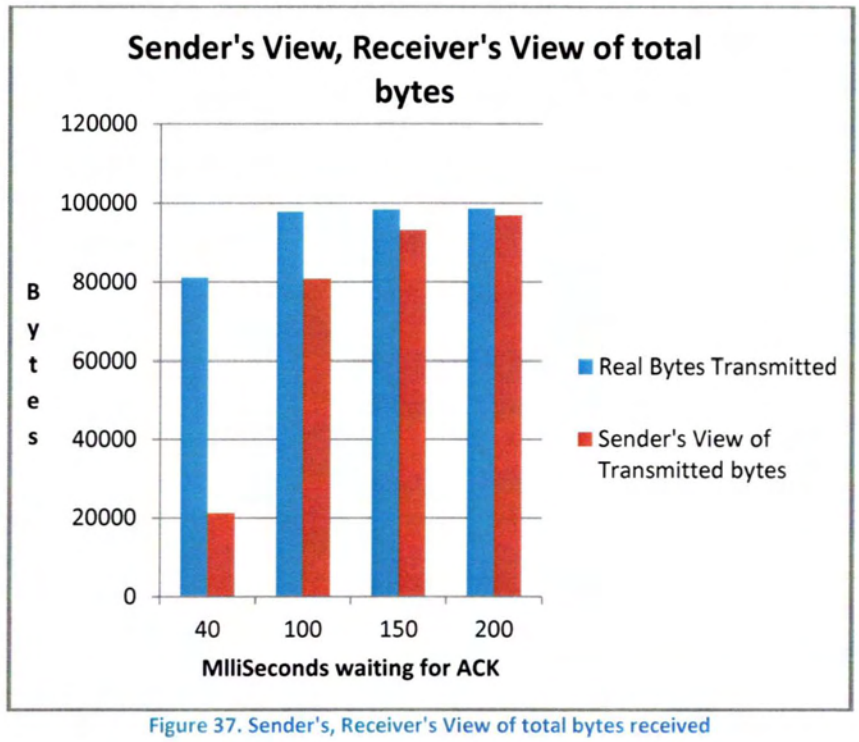


Figure 37. Sender's, Receiver's View of total bytes received

Figure 37 depicts the influence of this parameter. Another remarkable fact, as we can see from the chart, is that the higher the value, the greater the data dissemination. This is because the lower the sending frequency, the more probable the filling of receiver's buffer. This happens because the receiver can't directly settle all requests in a heavy loaded channel.

It is also worth to depict the number of occurred retransmissions both on a clear and a heavy loaded channel (*Figure 38*).

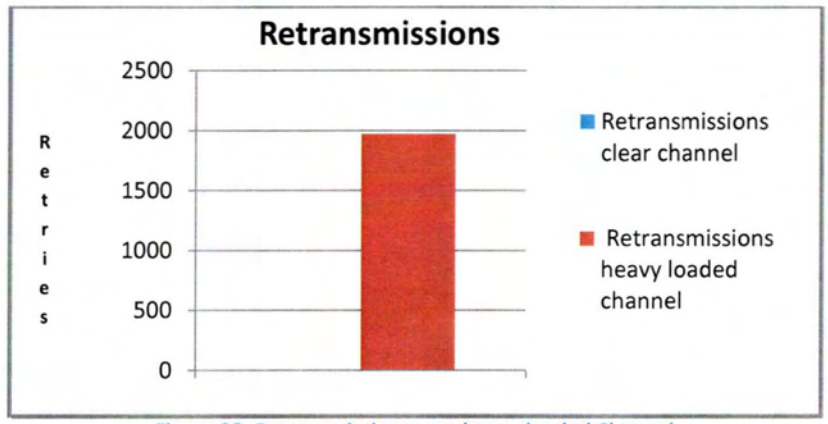


Figure 38. Retransmissions on a heavy loaded Channel

As data is transmitted from one node to the other, an acknowledgment packet (Ack) is transmitted in the opposite direction to indicate that the transmission was successfully received. If the Ack is not received, the transmitting device will retransmit the data, up to 4 times. For each retry of the device, the 802.15.4 MAC can execute up to 3 retries. This causes a lot of more data to be transmitted between nodes and results in poor network performance and data loss.

While all the above experiments were performed in a room, the distances between the nodes was small. Indicatively, distance between nodes was one meter and the distance between the interference source and the nodes was one meter too(Room scenario). Testing the performance of the network over longer distances, we defined the following scenario (Long Distance Scenario).

- Distance between nodes ten meters.
- Distance between interference source and nodes 10 meters.
- Also between nodes there is a wall(different rooms).

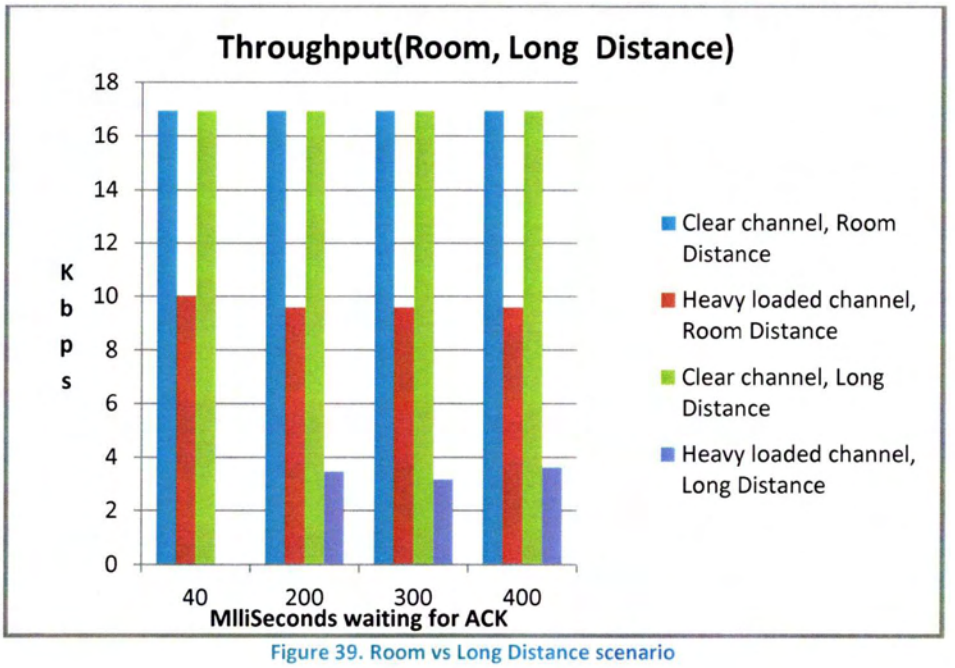


Figure 39. Room vs Long Distance scenario

As can be seen from *Figure 39* , we can achieve exactly the same throughput both on room and long distances in a clear channel. However, this can't be achieved when the channel is heavy loaded and as a result there is high interference in that frequency. So we lead to the fact that long distances strengthen the interference problem. One more noticeable fact is the number of time that the sender is waiting to receive an Acknowledgement (horizontal axis). In contrast to the room scenario, this number is increased in order the sender to have a valid view of delivered packages (*Figure 40*). As expected this brings higher costs of time on the network.

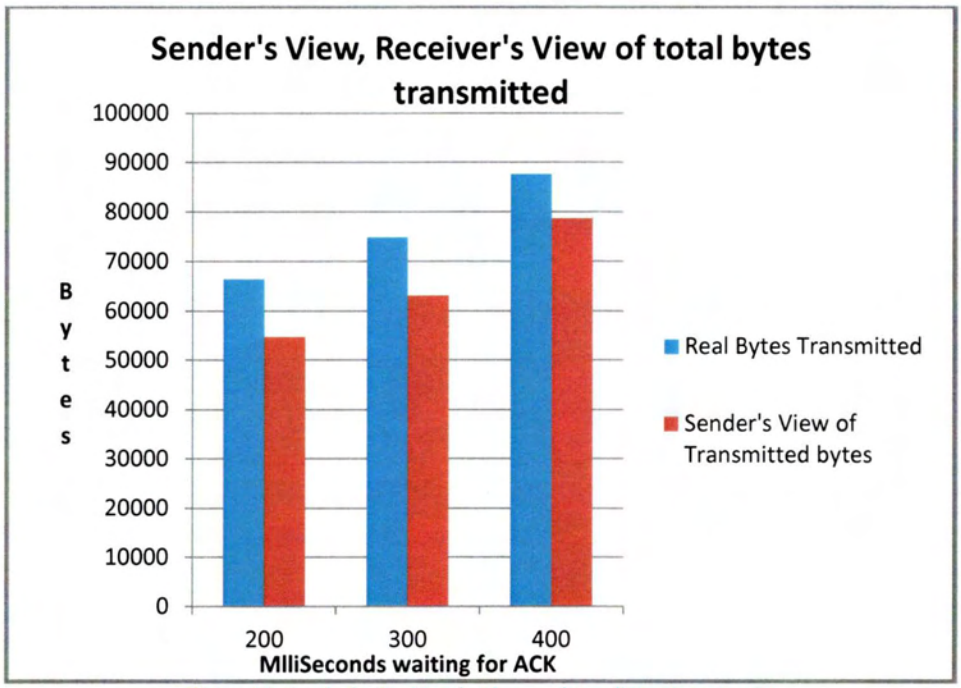


Figure 40. Sender's, Receiver's View on long distance scenario

Another important measure is the packet loss in each scenario. In the above chart (*Figure 41)* we can see the number of successful packets are transmitted in each scenario.

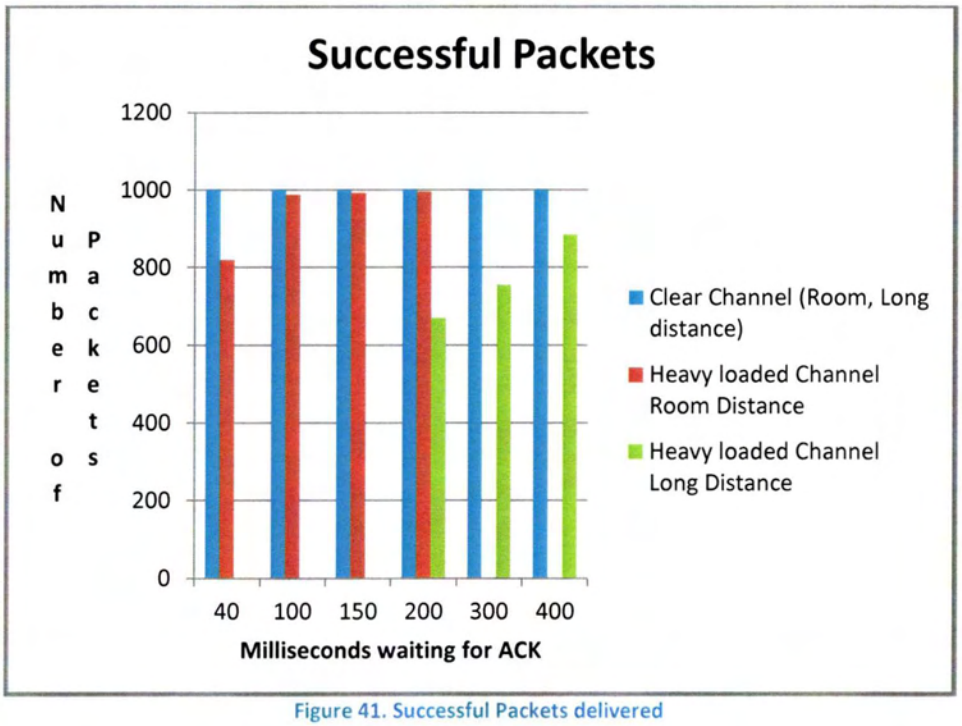


Figure 41. Successful Packets delivered

From what we can see, there is attained higher packet loss over long distances.
It is worth to mention that there is no reason to measure the "Heavy Channel Room Distance scenario" at 200, 300, 400 milliseconds, since we have already achieved to receive almost the whole number of the packets we transmitted at 200 milliseconds.

Applying this scenario to aforementioned time values, we achieve only to increase the delay of our network in the event that the packet is not successfully delivered.

Baud rate is the data rate in bits per second (baud) for serial data transmission. We use this parameter both on Computer-Arduino and Arduino-XBee communication. More specifically in Arduino-XBee communication, baud rate indicates the rate that data is transferred to the radio in order to be sent. Unsurprisingly, since the XBee module is fed with data from Arduino, this parameter can significantly affect the throughput of data between the radio modules. The higher the baud rate, the higher the throughput of data (*Figure 42*).

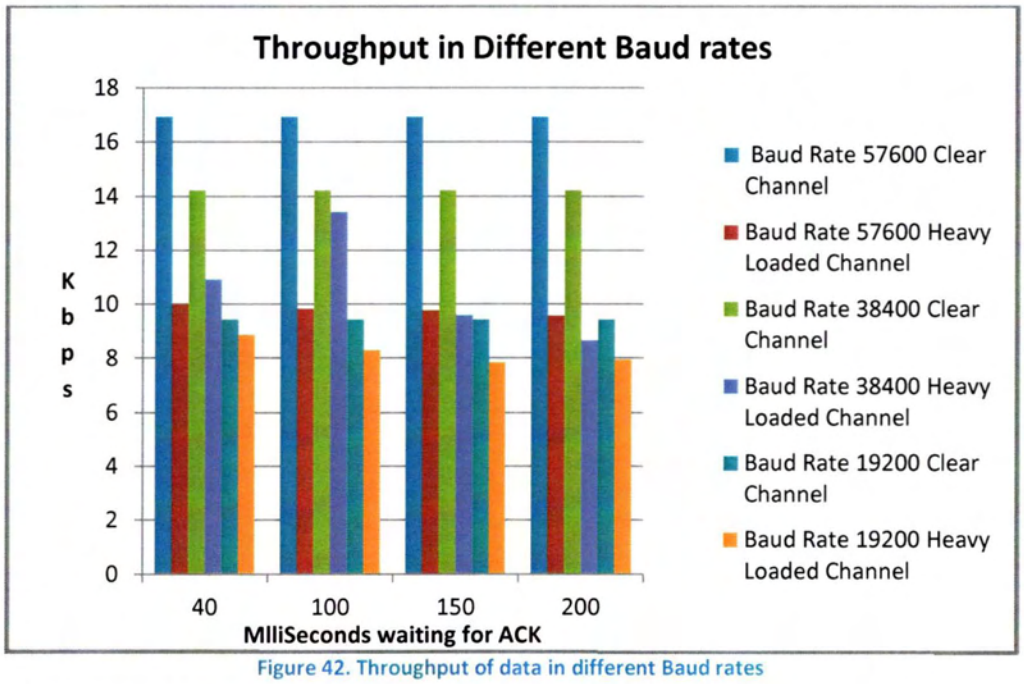


Figure 42. Throughput of data in different Baud rates

Unfortunately, we were unable to use bigger baud rates, because our SparkFun's pro Micro runs at 8MHz, and as a result there is no way to set a baud rate divisor that would get 115200 or bigger baud rates. According to the SparkFun's datasheet the error at 115200 is estimated to 8,5%.

The IEEE 802.15.4 PHY layer provides the ability to sample a channel, measure the energy, and report whether the channel is free from interference and thus clear to transmit. This information is then made available to higher layers so that devices using IEEE 802.15.4 radios have the possibility to select the best available channel for operation. This mechanism occurs when the network is formed for the first time. In order to achieve this on XBee, we can set the SC parameter accordingly so as to select from a range of channels. Based on this configuration, we seek to improve the mechanism of interference avoidance.

Configuring the XBee module to a range of specific channels (all ZigBee's channels in 2.4GHz band) and comparing with our implementation's results we conclude to the above measurements (*Figure 43*):

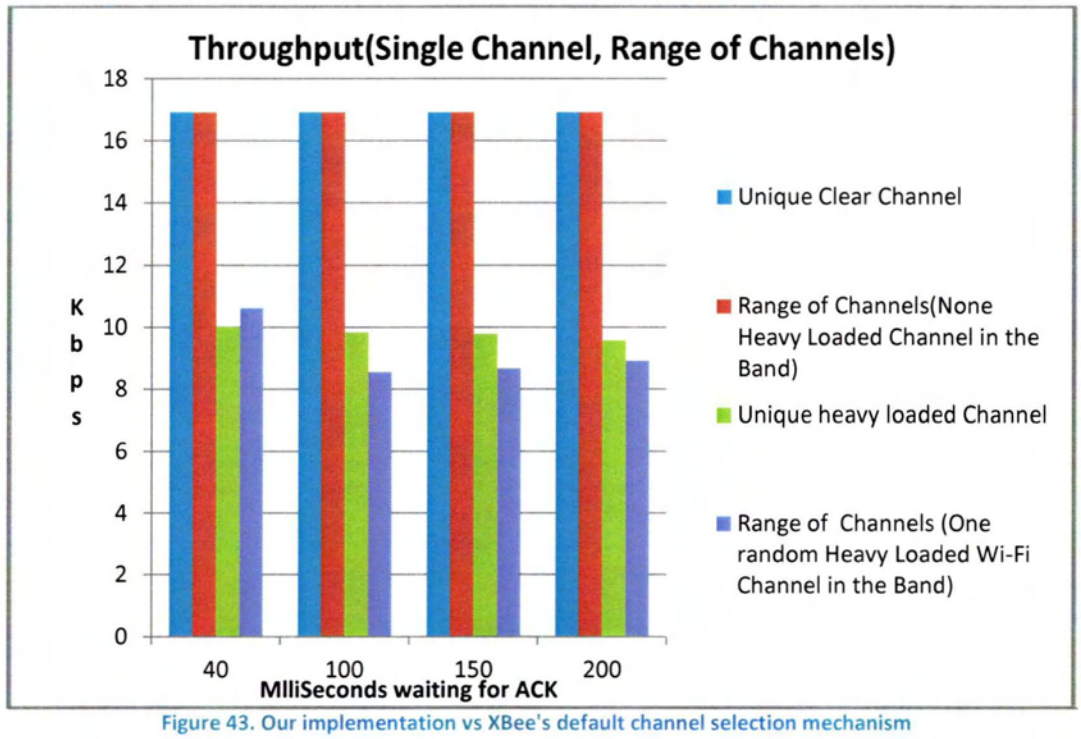


Figure 43. Our implementation vs XBee's default channel selection mechanism

From what we can see, we have the same performance on clear channels. This not observed on congested overlapped channels. More specifically when we select a unique congested channel(Green one) for our WSN, we have almost 10 Kbps throughput of data performance. Using our implementation we can quickly move our network to an uncongested frequency channel and to achieve 17 Kbps throughput of data. This, in contrast to the default interference avoidance mechanism, which achieves throughput of data at 9 Kbps on average on an interfere environment (purple).

On all of our experiments, we assume that a congested frequency channel is a heavy loaded frequency channel. As far as for an uncongested frequency channel, we may call it as a clear channel.

8 Conclusion & Future work

8.1 Conclusion

The present study aimed at better characterizing the effect of Wi-Fi transmissions on ZigBee performance of data throughput. We increasingly focused on the selection of an uncongested channel as well as in the transposition of the network's operation in this channel with respect to the network's functionality. Our results show better throughput of the data compared to the default XBee's interference avoidance mechanism.

8.2 Future Work

We plan to extend our work to the case where the microprocessor will features larger Processing power in order to achieve greater data feed to XBee radio modules and to summarize how the interference issue between IEEE 802.11 and ZigBee acts in these levels. Also, it would be beneficial to improve the channel selection algorithm. Finally, it is worth to evaluate our mechanism on networks with many nodes.

9 References

[1] Kazdaridis I. « Ανάπτυξη πλατφόρμας ασύρματων αισθητήρων και κατάλληλου λογισμικού διαχείρισης » http://www.inf.uth.gr/cced/wp-content/uploads/formidable/Master_Thesis_Report_Kazdaridis.pdf

[2] M. Petrova, *et al*, "IEEE 802.15.4 Low Rate - Wireless Personal Area Network Coexistence Issues," *Proc. IEEE WCNC'06*, Las Vegas, USA

[3] I. Howitt and J. A. Gutierrez, "Low-Rate Wireless Personal Area Networks - Enabling Wireless Sensors with IEEE 802.15.4," *Proc. IEEE WCNC'03*, vol.3, pp. 1481-1486

[4] S. Shin, *et al*, "Packet error rate analysis of IEEE IEEE 802.15.4 under IEEE 802.11b interference," *Proc. WWIC'05*, pp. 279-288

[5] A. Sikora, "Coexistence of IEEE 802.15.4 (ZigBee) with IEEE 802.11 (WLAN), Bluetooth, and Microwave Ovens in 2.4 GHz ISM-Band," *web document, http://www.ba-loerrach.de/stzedn/*

[6] http://electronicdesign.com/what-s-difference-between/what-s-difference-between-ieee-802154-and-zigbee-wireless

[7] http://sensor-networks.org/index.php?page=0823123150

[8] ZigBee- WiFi Coexistence , http://www.zigbee.org/LearnMore/KnowledgeBase.aspx?Contenttype=ArticleDet&Aid=143&CatID=

[9] http://en.wikipedia.org/wiki/IEEE_802.11#Layer_2_.E2.80.93_Datagrams

[10] http://www.atmel.com/Images/Atmel-42190-Coexistence-between-ZigBee-and-Other-24GHz-Products_AP-Note_AT02845.pdf

[11] http://en.wikipedia.org/wiki/Arduino

[12] https://www.sparkfun.com/products/12587

[13] http://processing.org/

[14] http://www.cypress.com/?docID=30630

[15] https://nurdspace.nl/Arduino_Radio_Spectrum_Analyzer_prototype_on_a_breadboard

[16] http://geoffg.net/ISM_Scanner.html

[17]http://translate.google.gr/translate?hl=el&sl=nl&tl=en&u=http%3A%2F%2Ffrack.nl%2Fwiki%2FArduino_Spectrum_Analyzer

[18] http://ftp1.digi.com/support/documentation/90000976_P.pdf

[19] http://examples.digi.com/get-started/basic-xbee-zb-zigbee-chat/

[20] http://www.digi.com/support/kbase/kbaseresultdetl?id=2125

[21] http://code.google.com/p/xbee-arduino/