Πανεπιστήμιο Θεσσαλίας, 2013-2014

Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών

Διπλωματική εργασία

Θέμα:

«Ανάπτυξη εργαλείων διαχείρισης σε Ασύρματα Δίκτυα Αισθητήρων»

«Development of Management Tools for Wireless Sensor Networks»

Συμεωνίδης Πολυχρόνης

Επιβλέπων καθηγητής:

Κοράκης Αθανάσιος (Λέκτορας Καθηγητής)

Συνεπιβλέπων καθηγητής:

Λάλης Σπυρίδων (Αναπληρωτής Καθηγητής)

Βόλος, Σεπτέμβριος 2014

Ευχαριστίες

Με την ολοκλήρωση της Διπλωματικής Εργασίας μου φέρω εις πέρας τις Προπτυχιακές σπουδές μου στο τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Πανεπιστημίου Θεσσαλίας.

Θα ήθελα να εκφράσω την ευγνωμοσύνη μου στον επιβλέπων της Διπλωματικής μου εργασίας, Λέκτορα του τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών κ. Αθανάσιο Κοράκη για την πολύτιμη καθοδήγηση που μου προσέφερε κατά την διάρκεια των σπουδών μου. Η παρουσία του στο τμήμα ήταν καταλυτική για την απόφασή μου να ασχοληθώ με τον τομέα των Δικτύων Η/Υ. Ακόμη, θα ήθελα να τον ευχαριστήσω για την δυνατότητα που μου προσφέρει να είμαι μέλος του NITlab και να αναπτύσσω συνεχώς τις γνώσεις μου πάνω στον τομέα των Δικτύων.

Επίσης, θα ήθελα να ευχαριστήσω θερμά τα παιδιά που αποτελούν την ομάδα του NITlab μέσω των οποίων το διάβασμα και η υλοποίηση διαφόρων projects ήταν μια ευχάριστη διαδικασία. Ιδιαιτέρως θα ήθελα να ευχαριστήσω τον υποψήφιο διδάκτορα του Τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών Καζδαρίδη Γιάννη για την απεριόριστη βοήθεια του και την καθοδήγηση του κατά την εκπόνηση της Διπλωματικής Εργασίας μου.

Τέλος, θα ήθελα να ευχαριστήσω την οικογένεια μου για την βοήθεια και τα εφόδια που μου παρείχε και τους φίλους μου για την υποστήριξη τους και την συμπαράστασή τους καθ' όλη τη διάρκεια των ακαδημαικών σπουδών μου.

Αφιερωμένο στην μητέρα μου και την αδερφή μου,
Αμαλία και Χρίστη

CONTENTS

ΠΕΡΙΛΗΨΗ

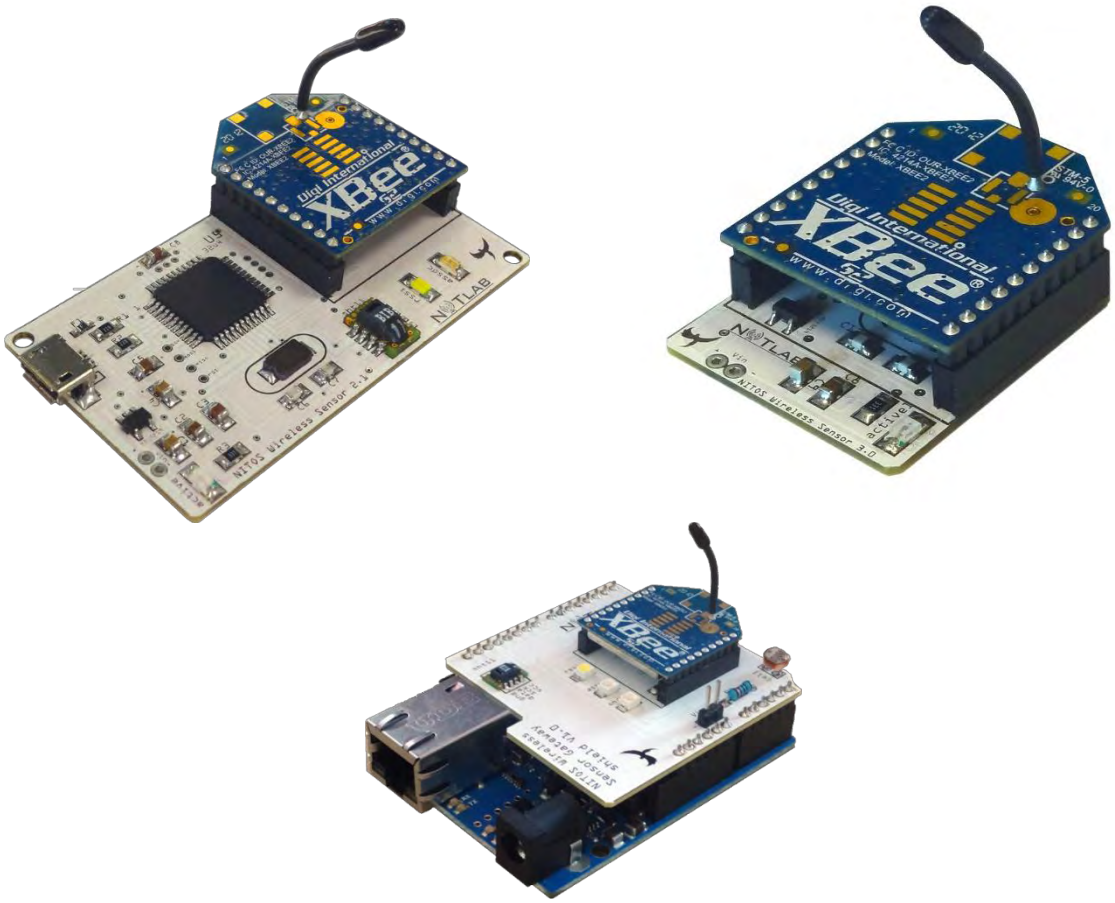
Οι ραγδαίοι ρυθμοί ανάπτυξης των Ασύρματων Δικτύων Αισθητήρων (ΑΣΔ) την τελευταία δεκαετία έχει βοηθήσει στην δημιουργία αυτόνομων και ελεγχόμενων υποδομών όπως Έξυπνα κτήρια, Έξυπνες Πόλεις κτλ. Αυτή η Διπλωματική Εργασία αναπτύσσει τις δυνατότητες της πρωτότυπης πλατφόρμας NITOS με την υλοποίηση διαφόρων εργαλείων λογισμικού και εξαρτημάτων υλικού οι οποίες επιτρέπουν τον βελτιωμένο έλεγχο και την καλύτερη παρακολούθηση της πλατφόρμας. Πιο συγκεκριμένα, στα πλαίσια αυτής της Διπλωματικής Εργασίας δημιουργήθηκε ένα εργαλείο τοπολογίας. Αυτό το εργαλείο λαμβάνει και απεικονίζει το συνολικό μονοπάτι επικοινωνίας και δρομολόγησης των ήδη ανεπτυγμένων ΑΣΔ και την ποιότητα του καναλιού της μεταξύ τους επικοινωνίας. Ακόμη υλοποιήθηκε ένα ισχυρό Gateway ΑΣΔ βασισμένο στην ευέλικτη Raspberry Pi πλατφόρμα συνδυασμένο με ένα XBee ασύρματο interface. Επίσης, προκειμένου να δημιουργήσουμε WSN clusters οι οποίοι υποστηρίζουν ευρυζωνικές backbone συνδέσεις εξοπλίσαμε το Raspberry Pi Gateway με USB dongles τα οποία επιτρέπουν επικοινωνία πάνω από δίκτυα 4^{ης} γενιάς (LTE/WiMAX). Επιπλέον αναπτύξαμε ένα έξυπνο web-based γραφικό εργαλείο το οποίο απεικονίζει real-time τις μετρήσεις των αισθητήρων του ήδη υπάρχοντος NITOS ΑΣΔ. Τέλος, μελετήσαμε και αναπτύξαμε μια μεθοδολογία μέσω της οποίας οι NITOS Ασύρματοι Κόμβοι μπορούν να κοιμηθούν προκειμένου να λειτουργούν για μεγάλες χρονικές περιόδους.

ABSTRACT

The rapid evolution of Wireless Sensor Networks (WSNs) over the last decade assists the formation of autonomous and controllable frameworks such as Smart-Buildings, Smart-Cities etc. This thesis expands the capabilities of the NITOS WSN prototype platform by developing several software tools and hardware components which allow enhanced monitoring and control of the platform. More specifically, in the context of this thesis a topology tool has been developed. This tool acquires and illustrates the overall communication path and routing of the deployed WSNs and their link quality. In addition, a powerful gateway node has been implemented using the versatile Raspberry Pi board along with an XBee wireless interface. Furthermore, in order to create WSN clusters that support broadband backbone connections we have equipped the Raspberry Pi gateway with USB dongles that enable communication over 4G networks (LTE/WiMAX). Moreover, we have developed an intuitive web-based graphical tool that depicts real-time sensor measurements of the deployed NITOS WSNs. Finally, we have studied and implemented a methodology through which NITOS WSN nodes can be set in sleep mode in order to operate unattended for long periods of time.

1 Introduction

Smart environments represent the next evolutionary development step in building, utilities, industrial, home, shipboard, and transportation systems automation. Like any sentient organism, the smart environment relies first and foremost on sensory data from the real world. Sensory data comes from multiple sensors of different modalities in distributed locations. In this thesis we present several implementations which assist in the monitoring and control of Wireless Sensor Networks (WSNs). In the second chapter we give an introduction on WSNs and their existing MAC and Network Protocols. In the third chapter we present the modules, platforms, architectures and standards used to build the monitoring tools. Finally in the fourth chapter we exhibit the aforementioned tools, hardware components and software features implemented in the context of this thesis.



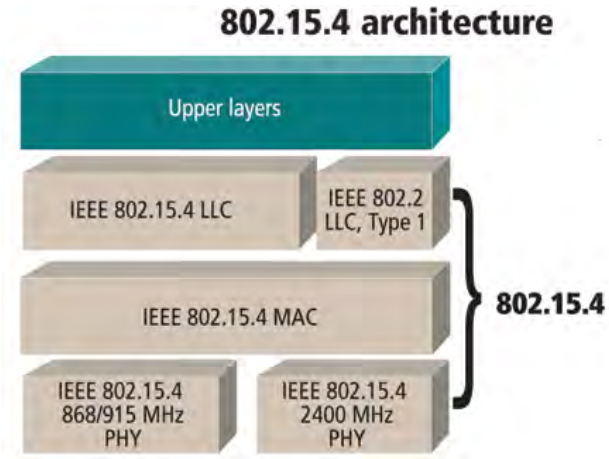
2 Wireless Sensor Networks and Existing MAC and Network Protocols

2.1 Wireless Sensor Networks

A wireless sensor network [1] is a collection of nodes organized into a cooperative network. Each node consists of processing capability (one or more microcontrollers, CPUs or DSP chips), may contain multiple types of memory (program, data and flash memories), has an RF transceiver (usually with a single omnidirectional antenna), has a power source (e.g. batteries and solar cells) and accommodates various sensors. A wireless sensor network (WSN) generally consists of a base station (or “gateway”) that can communicate with a number of wireless sensors via a radio link. Data is collected at the wireless sensor node, compressed, and transmitted to the gateway directly or, if required, uses other wireless sensor nodes to forward data to the gateway. The transmitted data is then presented to the system by the gateway connection. Creating wireless networks can be done using a variety of RF protocols, the most widely used protocol is the 802.15.4 standard.

2.2 IEEE 802.15.4 standard

The IEEE [2] 802.15.4 is a standard which specifies the physical layer and media access control (MAC) for low-rate wireless personal area networks (WPANs). Devices are conceived to interact with each other over a conceptually simple wireless network. The definition of the network layers is based on the OSI (Open System Interconnection) model although only the lower layers are defined in the standard, interaction with upper layers is intended, possibly using an IEEE 802.2 logical link control sub-layer accessing the MAC through a convergence sub-layer.



2.2.1 802.15.4 Physical layer

Physical layer is the initial layer in the OSI reference model used worldwide. The physical layer (PHY) ultimately provides the data transmission service, as well as the interface to the physical layer management entity, which offers access to every layer management function and maintains a database of information on related personal area networks. Thus, the PHY manages the physical RF transceiver and performs channel selection and energy and signal management functions. It operates on one of three possible unlicensed frequency bands:

- 868.0 - 868.6MHz: Europe - 1 channel
- 902.0-928.0MHz: North America - up to 10 channels
- 2400–2483.5 MHz: Worldwide – up to 16 channels

Supported Bit Rates:

- 868.0 - 868.6MHz: 20, 40, 100, 250 Kb/s
- 902.0-928.0MHz: 40,250 Kb/s
- 2400–2483.5 MHz: 250 Kb/s

2.1.2 802.15.4 MAC Layer

This standard defines a communication layer at the 2nd level of the OSI model. The medium access control (MAC) enables the transmission of MAC frames through the use of the physical channel. Besides the data service, it offers a management interface and itself manages access to the physical channel and network beaconing. It also controls frame validation, guarantees time slots and handles node associations. Finally, it offers hook points for secure services.

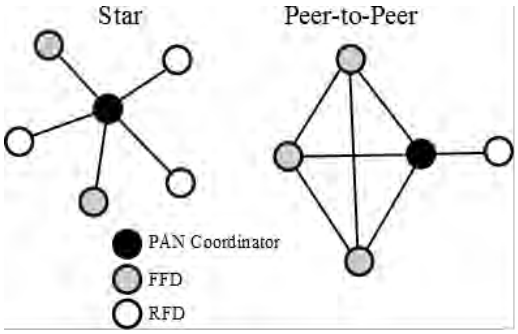
2.1.3 802.15.4 Network model

2.1.3.1 Node Types

The standard defines two types of network nodes. The first one is the full-function device (FFD). It can serve as the coordinator of a personal area network just as it may function as a common node. It implements a general model of communication which allows it to talk to any other device: it may also relay messages, in which case it is dubbed a coordinator (PAN coordinator when it is in charge of the whole network). On the other hand there are reduced-function devices (RFD). These are meant to be extremely simple devices with very modest resource and communication requirements; due to this, they can only communicate with FFDs and can never act as coordinators.

2.1.3.2 Topologies

Networks can be built as either **peer-to-peer** or **star** networks. However, every network needs at least one FFD to work as the coordinator of the network. Networks are thus formed by groups of devices separated by suitable distances. Each device has a unique 64-bit identifier, and if some conditions are met short 16-bit identifiers can be used within a restricted environment. Namely, within each PAN domain, communications will probably use short identifiers.



Peer-to-peer (or point-to-point) networks can form arbitrary patterns of connections and their extension is only limited by the distance between each pair of nodes. They are meant to serve as the basis for ad hoc networks capable of performing self-management and organization. Since the standard does not define a network layer, routing is not directly supported, but such an additional layer can add support for multi-hop communications. Further topological restrictions may be added; the standard mentions the cluster tree as a structure which exploits the fact that an RFD may only be associated with one FFD at a time to form a network where RFDs are exclusively leaves of a tree, and most of the nodes are FFDs. The structure can be extended as a generic mesh network whose nodes are cluster tree networks with a local coordinator for each cluster, in addition to the global coordinator.

A more structured star pattern is also supported, where the coordinator of the network will necessarily be the central node. Such a network can originate when an FFD decides to create its own PAN and declare itself its coordinator, after choosing a unique PAN identifier. After that, other devices can join the network, which is fully independent from all other star networks.

2.1.4 On top of IEEE 802.15.4

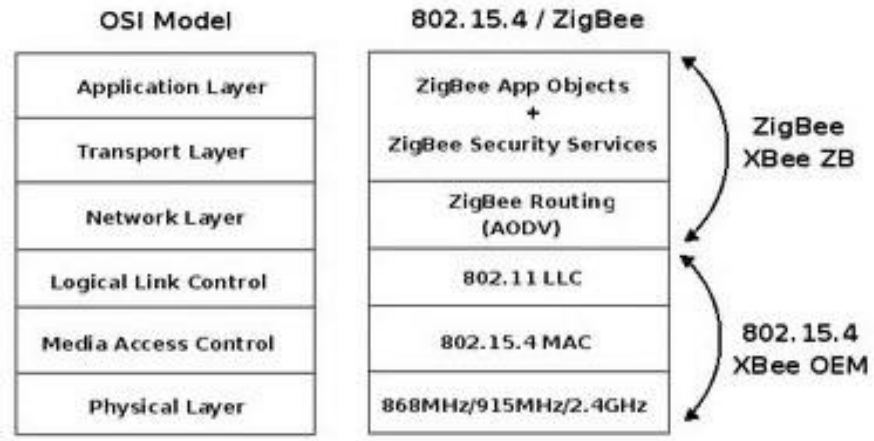
There are several protocols which use IEEE 802.15.4 [4] as its MAC layer. The most known is **ZigBee** [5], although the protocol is the basis for several other specifications such as:

- **Wireless HART** [6]**:** It is the wireless version of the HART protocol which is the most used in the automation and industrial applications which require real time. It uses Time Synchronized Mesh Protocol (TSMP). A "time coordinator" node is required in order to assign the time slot to all the motes.
- **ISA - SP100** [7]**:** It also centers in the process and factory automation. It is being developed by the Systems and Automation Society (ISA) and tries to be an standard for this kind of projects.
- **6LoWPAN** [8]**:** As the may point out it is the implementation of the IPv6 stack on top of 802.15.4 to let any device be accessible from and to the Internet.
- **MiWi** [9]**:** It is designed for low data transmission rates and short distance
- **Mesh protocols**: A lot of different mesh networking protocols have been and are being implemented by companies over the 802.15.4 MAC layer. One in particular is Digi's [11] **DigiMesh™**. DigiMesh™ is a proprietary peer-to-peer networking topology for use in wireless end-point connectivity solutions. The nature of its peer-to-peer architecture allows DigiMesh™ to be both easy to use and equipped with advanced networking features, including support for sleeping routers and dense mesh networks.

Reference guide to 802.15.4 description: [3].

2.3 ZigBee Routing Protocol

This standard defines a communication layer at level 3 and up in the OSI model. Its main purpose is to create a network topology (hierarchy) in order to let a number of devices communicate and set extra communication features such as authentication, encryption, association and application services in the upper layer. It was created by a set of companies which form the ZigBee Alliance.



2.2.1 Understanding ZigBee

ZigBee offers basically four kinds of different services:

- **Extra Encryption services** (application and network keys implement extra 128bits AES encryption)
- **Association and authentication** (only valid nodes can join to the network).
- **Routing protocol**: AODV [11], 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 dedicated to organize the network. The first thing a node (router or end-device) joining the network has to do is to make a request to the coordinator for a network address (**16-bit**), as part of the **association** process. All the information in the network is routed using this address and not the 64-bit MAC address. In this step authentication and encryption procedures are performed.

Once a node has joined the network it can send information to its siblings through the routers which are always awake awaiting for packets. When the router gets the packet and the destination is in its radio of signal, the router first determines if the destination end-device is awake or sleeping. In the first case the router sends the packet to the end device, however if it is sleeping, the router will buffer the packet until the end device node wakes up and asks for buffered packets.

2.2.2 ZigBee as a mesh protocol

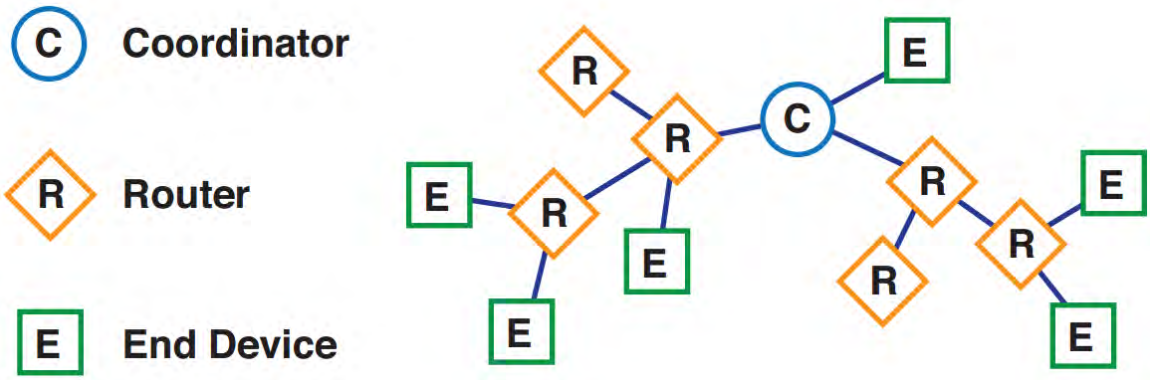
ZigBee networks can operate as Mesh networks. There are three kinds of nodes in a ZigBee network:

- **Coordinator**: is the "master" device, it governs the entire network.

- **Routers**: they route the information which sent by the end devices.
- **End devices**: (motes): they are the sensor nodes, the ones which acquire the information from their accommodated sensors.

Coordinator and routers cannot be battery powered, motes can. ZigBee creates **star topologies**. There are some basic rules:

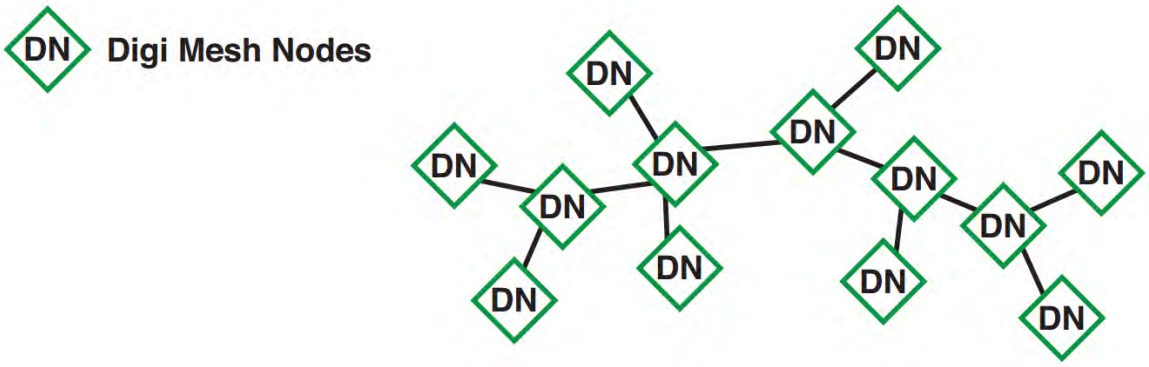
- The end devices connect to a router or a coordinator.
- The routers can connect among them and with the coordinator.
- The routers and coordinators cannot sleep. They have to save in their buffer the packets which are destined for the end devices.
- The end devices can sleep.



The concept "Mesh Network" relays in the Ad hoc communications, also called peer to peer (P2P). This means all the devices in the network can communicate with each other directly. They have to be able to discover each other and send broadcast messages to all the siblings. They have to be able to create networks like the one represented in the image below.

ZigBee creates star network topologies, not mesh ones. To create a completely mesh network such as the one showed in the image below all the nodes have to have the same **role**, all of them have to be *"end devices + routers"* so that they can route their sibling information and sleep when no action is required (conserving energy).

The **DigiMesh™** [10] protocol (over 802.15.4) sets a completely distributed network where all the nodes can communicate using P2P datagrams.



Reference guide to ZigBee protocol description: [1].

2.3 DigiMesh™ Networking Protocol

As mentioned above DigiMesh™ is a proprietary P2P networking topology for use in wireless end-point connectivity solutions. The nature of its peer-to-peer architecture allows DigiMesh™ to be both easy to use and equipped with advanced networking features, including support for sleeping routers and dense mesh networks. Overhead associated with the protocol and data payload is optimized for network performance and addressing is made simple so you spend less time defining your network, and more time on your application.

2.3.1 Understanding DigiMesh™

DigiMesh™ was developed to address the needs of a broad category of wireless end-point connectivity applications. DigiMesh™ is an ideal solution for setups that require:

- Robust mesh networking (no Parent/Child dependencies)
- Support for advanced mesh networking, including dense networks
- A power-optimized protocol with support for sleeping routers for power-sensitive or battery-dependent networks
- An easy-to-use protocol that simplifies mesh networking (no need to define and organize coordinators, routers or end-nodes)
- The ability to deploy wireless solutions in both 900 MHz & 2.4 GHz

3 Hardware devices used to implement WSN tools

In order to be able to implement the tools of this thesis several hardware modules had to be used. These modules are presented below.

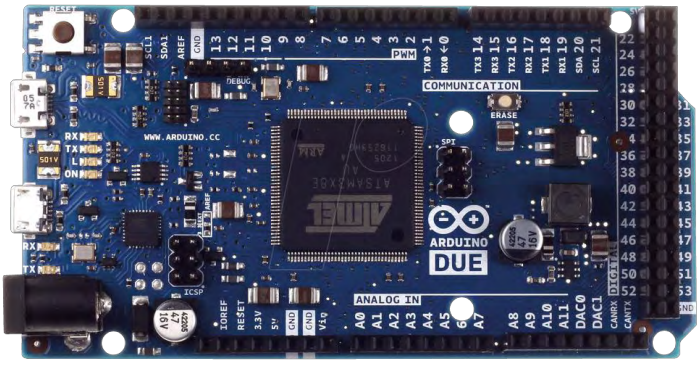
3.1 Arduino

The most common open-source electronics platform is Arduino [14]. Arduino is a single-board microcontroller, the hardware consists of an open-source hardware board designed around an 8-bit Atmel AVR microcontroller or a 32-bit Atmel ARM. The Arduino platform is low-cost and can be used in a variety of projects. 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. Most boards include a 5 volt linear regulator and a 16 MHz crystal oscillator. What differentiates the Arduino from other platforms is that an Arduino's microcontroller is 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.

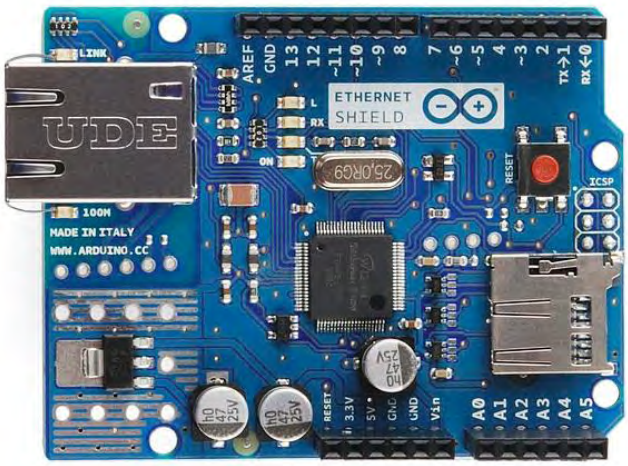
The Arduino board can easily be programmed through a cross-platform Arduino Software IDE

The Arduino platforms used in the context of this thesis are:

- Arduino DUE [16] It is the first Arduino board based on a 32-bit ARM [15] core microcontroller. It is a powerful board that has a CPU Clock at 84MHz, 96 Kbytes of SRAM and 512 Kbytes of Flash memory for code.



- Arduino Ethernet Shield [17]: This Ethernet Shield allows an Arduino board to connect to the internet. It is based on the WizNet W5100 Ethernet chip. It supports up to 4 simultaneous socket connections. The Ethernet Shield has a standard RJ-45 connection with an integrated line transformer and Power over Ethernet (PoE) enabled. Also there is an onboard micro-SD card slot, which can be used to store files for serving over the network.



3.2 NITOS Wireless Sensor Mote

The prototype NITOS wireless sensor mote [19], is comprised of open-source and configurable modules. NITOS mote features the ATmega32u4 [18] microcontroller running at 8MHz and operating at 3.3V. The aforementioned microcontroller is fully compatible with the Arduino platform that enables ease of software development and provides compatibility with several commercial sensing modules. Moreover, the platform is equipped with an XBee [22] radio interface that enables communication with the respective gateway. The XBee module is a tiny device ideal for setting up mesh networks and has a defined rate of 250 kbps. This module uses the IEEE 802.15.4 stack which is the basis for the Zigbee protocol. Apart from the Xbee module, NITOS mote can also feature a WiFi wireless interface in order to communicate with WiFi gateways. The developed mote currently features specific sensing modules, an air temperature and humidity sensor, a light intensity sensor and a human presence sensor. Various types of sensing modules and actuators can be further integrated exploiting existing Arduino software that implements several existing communications protocols. The firmware can be easily uploaded through the on-board USB connection.

3.3 XBee

The XBee radios are a set of RF modules designed to operate under the IEEE 802.15.4 networking protocol. They are developed by Digi International and provide low-cost wireless connectivity to WSN nodes. These modules provide high-throughput and low latency communication. Communication is achieved through a UART.

In this thesis there have been used two types of XBee modules.

- The XBee 802.15.4 (Series 1) [25] and its variant the XBee PRO 802.15.4 (Series 1) which are devices that operate at 2.4GHz and provide wireless communication based on the IEEE 802.15.4 standard. Their specifications are listed below:

	XBee 802.15.4 (Series 1)	XBee PRO 802.15.4 (Series 1)
Power output	1mW North American & International version	63mW North American 10mW International
Indoor/Urban range	Up to 30m	Up to 90m
Outdoor/RF line-of-sight range	Up to 90m	Up to 1.6km
Transmit current	45 mA (@ 3.3 V) boost mode 35 mA (@ 3.3 V) normal mode	215 mA (@ 3.3 V)
Receive current	50 mA (@ 3.3 V)	55 mA (@ 3.3 V)
Power-down sleep current	<10 µA	<10 µA

- The XBee ZB (Series 2) [26] and its variant the XBee PRO ZB (Series 2) which are devices that also operate at 2.4GHz and provide wireless communication based on the ZigBee protocol. They offer interoperability with other ZigBee devices. Some of their specifications are shown below:

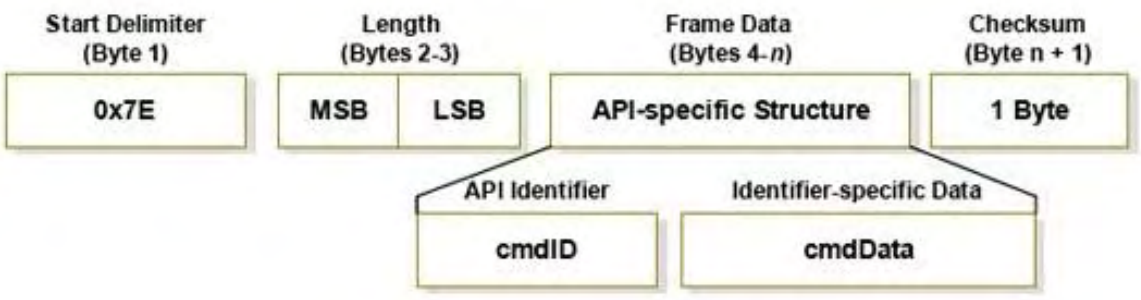
	XBee ZB (Series 2)	XBee PRO ZB (Series 2)
Power output	2mW North American & International version	63mW North American 10mW International
Indoor/Urban range	Up to 40m	Up to 90m
Outdoor/RF line-of-sight range	Up to 120m	Up to 1.5km
Transmit current	40 mA (@3.3 V)	220 mA (@ 3.3 V)
Receive current	38 mA / 40 mA boost mode @ 3.3V	62 mA (@ 3.3 V)
Power-down sleep current	<1 uA	4uA

The XBee modules can be configured through Digi's XCTU program. The XCTU is a multi-platform application designed to enable developers to interact with the XBee RF module. This interaction includes firmware updates in order to change operating modes (e.g. from ZigBee Coordinator to ZigBee Router) and to manage and configure the RF devices.

XBees have two modes of operation:

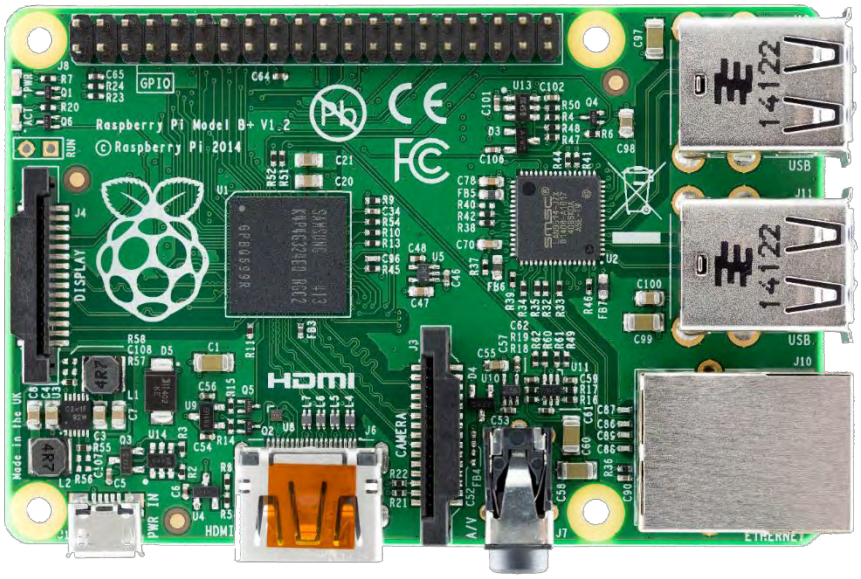
- **AT Command Mode (Transparent Mode):** The XBees that operate in AT Command Mode can be configured through simple AT commands issued through the serial UART. The XBee module enters the command mode when the characters "+++" are sent to the XBee. Afterwards the XBee replies with "OK" and the XBee is in command mode. An example of an AT command is "ATID" which returns/sets the XBees PAN ID.
- **API Mode:** API (Application Programming Interface) mode is a frame-based method for sending and receiving data to and from a radio's serial UART. These frames are called ZigBee Device Objects [27]. The API is an alternative to the default transparent mode. The API allows the programmer the ability to:
    - Change parameters without entering command mode
    - View RSSI and source address on a packet by packet basis
    - Receive packet delivery confirmation on every transmitted packet

    A typical API packet format is:



3.4 Raspberry Pi

The Raspberry Pi [28] is a low cost ($25-$35), small, single-board computer that was first released on February 2012. It is a highly versatile platform that was designed to have a wide variety of capabilities. It can be used as a basic computer, as a DIY platform or even as a WSN Gateway. The Pi features an ARM CPU, HDMI Output, 256-512MB RAM, USB hosting, Ethernet port and several GPIO Pins. There are three models available in the market, model A, model B and model B+.



In our implementation we used the Raspberry Pi Model B+ which features:

- 4 USB ports
- 40 GPIO pins
- 512MB RAM
- 700MHz ARM CPU

4 Implementations

The main motivation of my thesis was to implement certain tools for the NITOS WSN Platform that would facilitate the monitoring and control of the platform. The tools developed are the NITOS WSN Topology Tool, which is a tool that reports the Topology of an existing WSN and the NITOS WSN Graphic Depiction Tool, which - as the name may already reveal – is a web-based tool used to inform the user of the current measurements acquired by the WSN mote sensors in graphical way. During the elaboration of my thesis certain needs for hardware components and software features emerged. The hardware component implemented is a powerful WSN Gateway that can be used to implement more complex deployments and support more features than the restricted, in terms of processing power and memory, Arduino platform. The platform used for this implementation is the Raspberry Pi. In addition to the implementation of the WSN

Gateway we upgraded the Raspberry Pi WSN Gateway by adding a new feature that cannot be supported by an Arduino WSN Gateway. This feature is the support for a 4G communication backbone for the WSN. Another software upgrade is the integration of coordinated cyclic sleep on the NITOS WSN City Mote. This feature allows for a WSN mote that can operate autonomously for long periods of time solely by using a rechargeable Li-Po battery as a power source and a solar panel for battery charging.

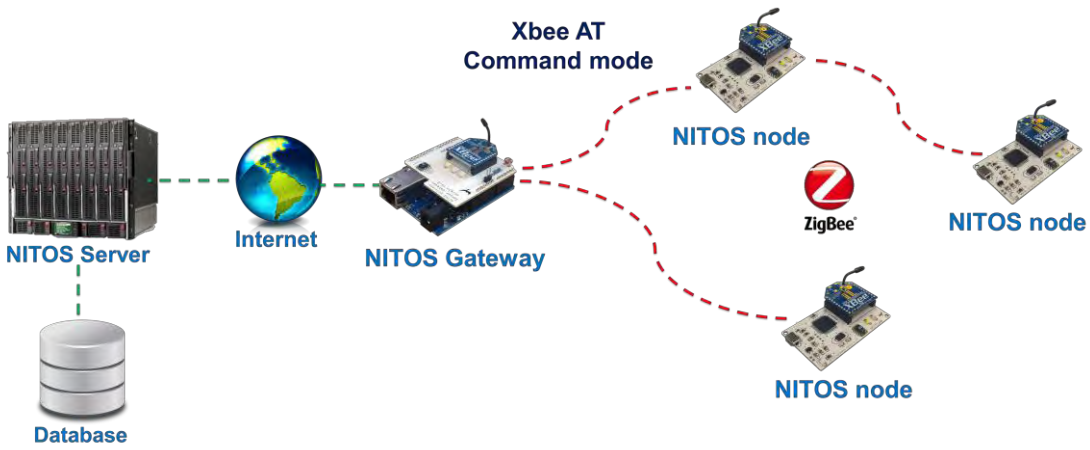
The implementations mentioned above are presented in more detail in the forthcoming chapter.

4.1 Topology Tool

The NITOS WSN Topology Tool is a monitoring tool designed to report and display the network topology of an existing WSN. The tool acquires the LQI Table (Link Quality Indicator) and the Routing Table of each node. The acquisition is accomplished by sending ZigBee Device Objects (ZDOs) through the XBee wireless interface of a dedicated Arduino-based device.

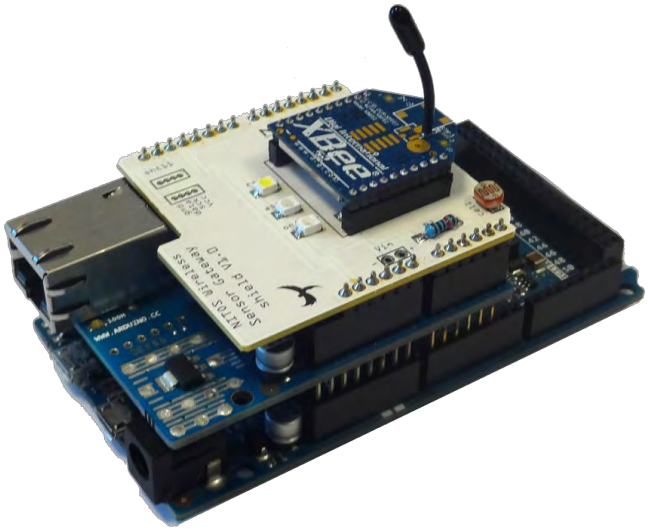
4.1.1 Existing WSN Topology

The tool was implemented based on the characteristics of the deployed NITOS Indoor WSN testbed. The NITOS Indoor WSN consists of several WSN motes and a WSN Gateway based on the Arduino platform. The gateway node is connected through an Ethernet network to the NITOS Server. Each NITOS WSN mote is equipped with a microcontroller, several sensors and an XBee wireless interface. The wireless interface is configured to operate in AT Command mode. In order to acquire the topology information of the WSN we must use two types of ZDOs (0x0031 & 0x0032), and to be able to send the ZDOs we need an XBee in API mode. Due to the fact that we don't want to modify the existing topology and implementation of the WSN testbed we implemented a new dedicated device. This device acts as an extra router that acquires the LQI and Routing tables of the network and then reports them back to the server through an Ethernet connection.



4.1.2 Hardware

The hardware device constructed for the topology tool is based on the Arduino DUE, which is a powerful board that can host and execute more complex code implementations such as the NITOS WSN Topology Tool. Attached to the Arduino DUE is an Arduino Ethernet Shield in order to run a web server on the device. And last the device is equipped with an XBee S2 configured in API mode.



4.1.3 Hardware configuration

The implementation of the NITOS WSN Topology Tool required several hardware modifications in order to work properly. The main modification was to increase the Hardware Serial Buffer size, which is the buffer where the information that arrives on the serial port is stored temporarily, from 64 bytes to 256 bytes. This configuration had to be made because the ZDO packets sent and received by the XBees when in API mode have a size of more than 64 bytes. Without this modification the Arduino board that receives the information from the XBee through the UART would lose these packets.

The modification done in the path:
"**arduino-1.5.6-r2\hardware\arduino\sam\cores\arduino\RingBuffer.h**"
is shown below:

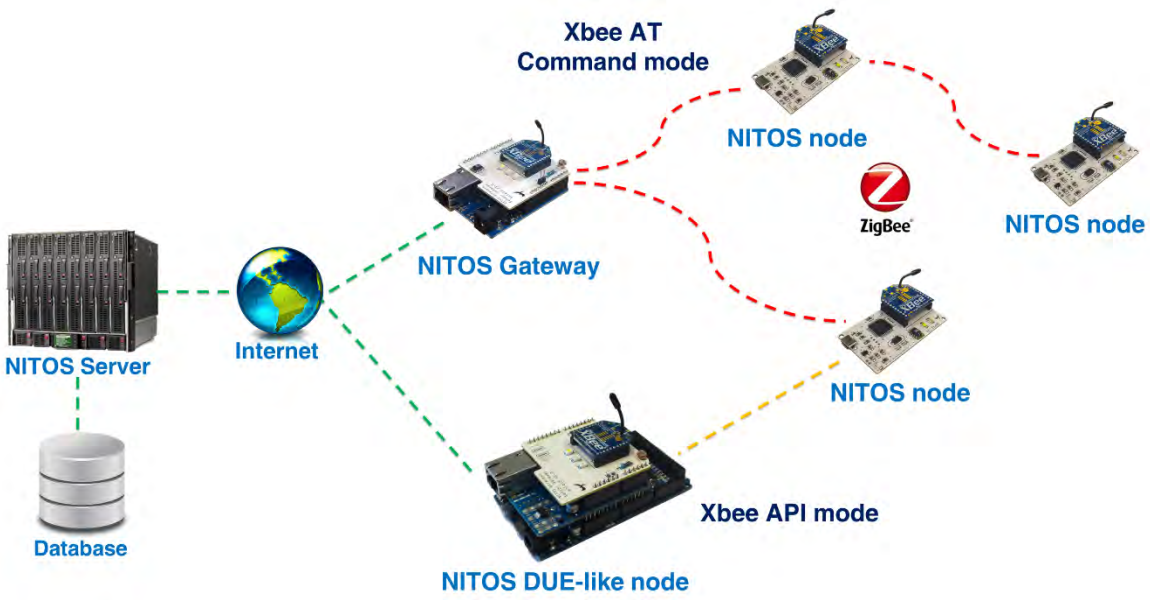
```
#define SERIAL_BUFFER_SIZE 256
```

Another hardware configuration was to set the MAC address and IP address of the Web Server that runs on the Arduino platform. And the last modification was to upload new

ZigBee Router firmware on the XBee S2 module that is able to communicate in API mode and set the network ID of the XBee in order to join the existing network. The configurations for the XBee were executed through Digi's X-CTU program.

4.1.4 Implementation – Software

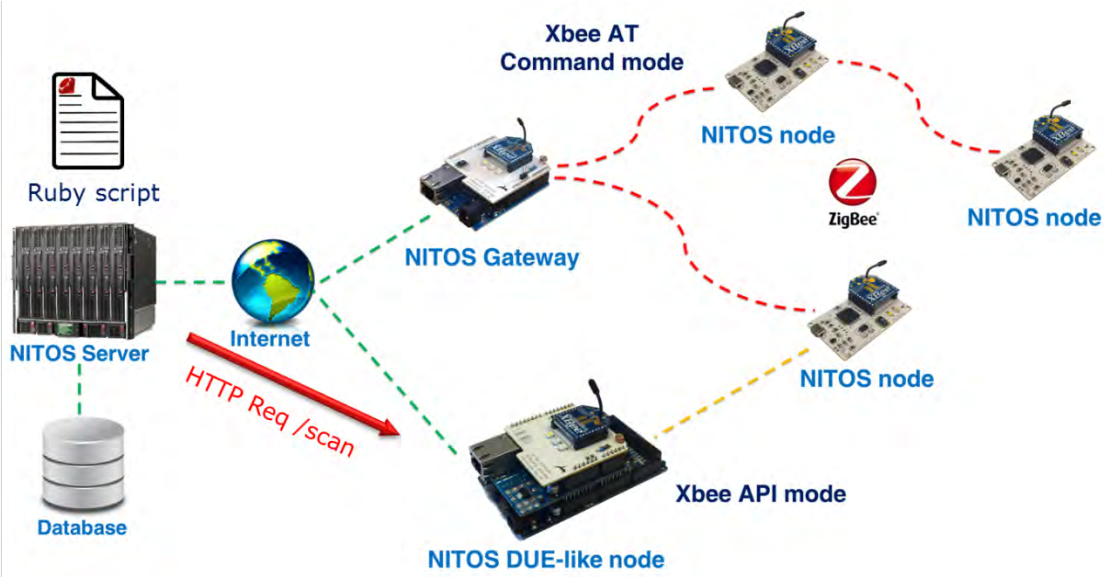
After the construction of the dedicated device and the hardware configurations the new NITOS Indoor WSN architecture is as shown in the image below. The device joins the network and communicates with the NITOS server through its own Ethernet link. On the device we uploaded code that is able to acquire the LQI and Routing tables of every device in the network and report it when asked to.



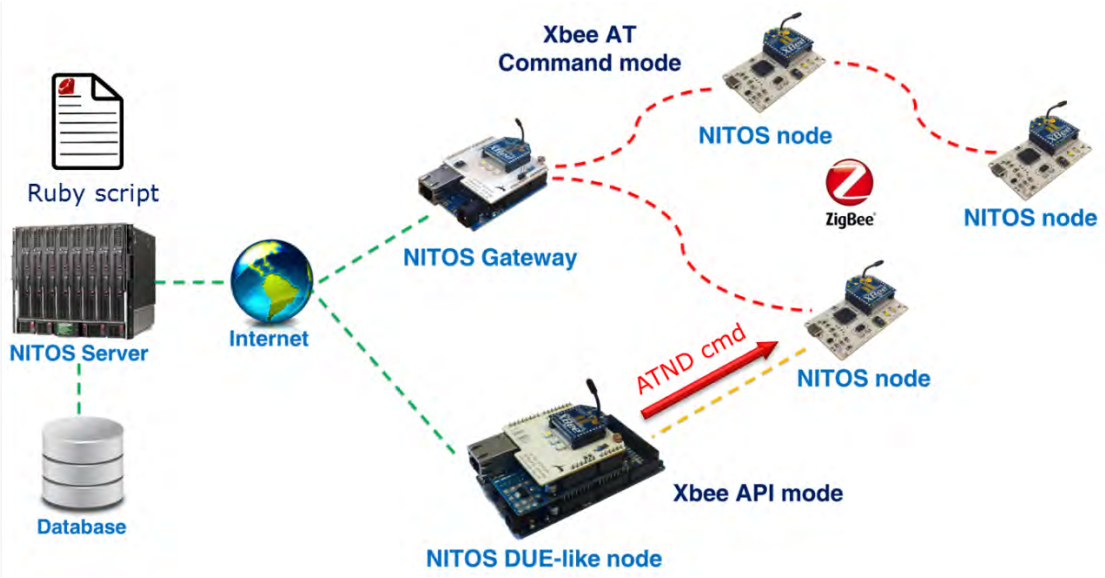
The acquisition of the WSN topology commences periodically of on demand by a **Ruby script** that runs on the NITOS Server and is executed in several sequential steps.

```ruby
require 'open-uri'
require 'nokogiri'
require 'sequel'

DB = Sequel.mysql('sensors_city_map', :host=>'#########', :user=>'########', :password=>'#########')

loop do

  begin

  contents = open('http://10.64.44.115/scan') {|io| io.read}
  puts contents
  sleep 20

  doc = Nokogiri::HTML(open("http://10.64.44.115/getdata"))
  res = doc.xpath('//xbee')

  #add to database
  #DB = Sequel.mysql('nitlab', :host=>'83.212.100.171', :user=>'test', :password=>'pass')
  #DB = Sequel.mysql('sensors_city_map', :host=>'10.64.86.170', :user=>'sensors_user', :password=>'$sensors$')

  xbees = DB[:xbees]
  puts xbees.delete

  res.each do |xbee|
    xbees.on_duplicate_key_update(:LQIIndex,:LQITable).multi_insert(
    [{ addr16: xbee.xpath('addr').text.strip, RoutingIndex: "0", :RoutingTable => "00300", LQIIndex: xbee.xpath('lqiindex').text.strip, LQITable: xbee.xpath('lqitable')

    puts xbee.xpath('addr').text.strip
    puts xbee.xpath('lqiindex').text.strip
    puts xbee.xpath('lqitable').text.strip
  end

  DB.disconnect
  rescue Errno::ECONNREFUSED,Errno::EHOSTUNREACH,Timeout::Error, Errno::EINVAL, Errno::ECONNRESET, EOFError,
      Net::HTTPBadResponse, Net::HTTPHeaderSyntaxError, Net::ProtocolError => e
    puts "Exception: #{e}"
    next
  end
sleep 20
end
```

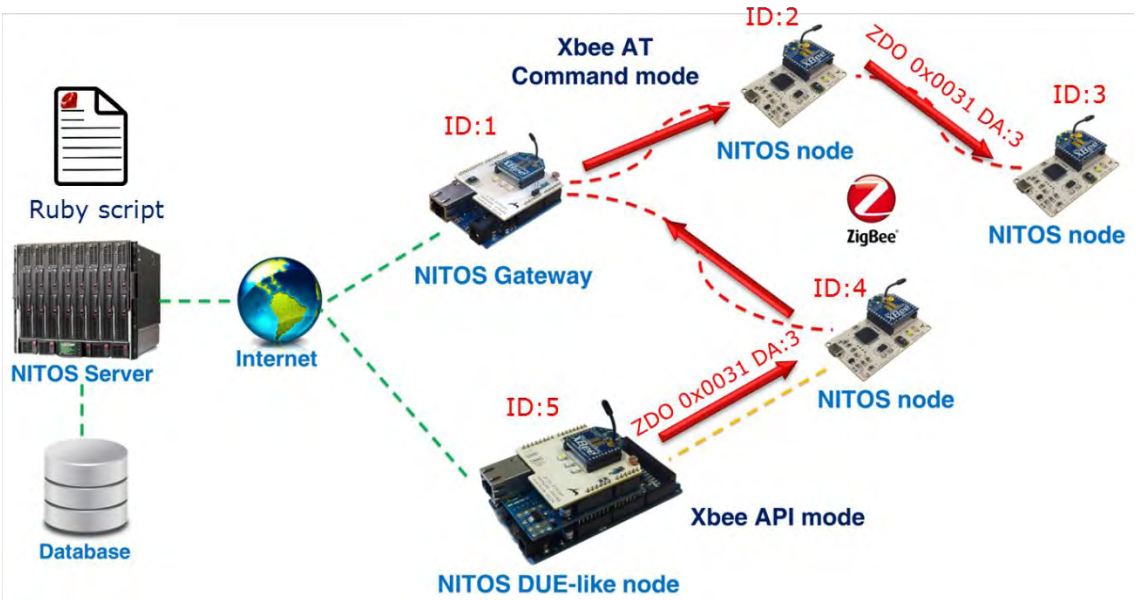
In the first step the script issues an HTTP Request to the NITOS DUE-like node with the identifier “/scan”. Then the device sends an HTTP Reply “Scan Initiated”.



In the second step the dedicated device broadcasts an ATND (Node Discovery) command in the form of a ZDO frame. Then the device awaits for a response from the nodes. The response contains the 64-bit and 16-bit addresses of every node in the network. The device receives the responses and stores them locally.

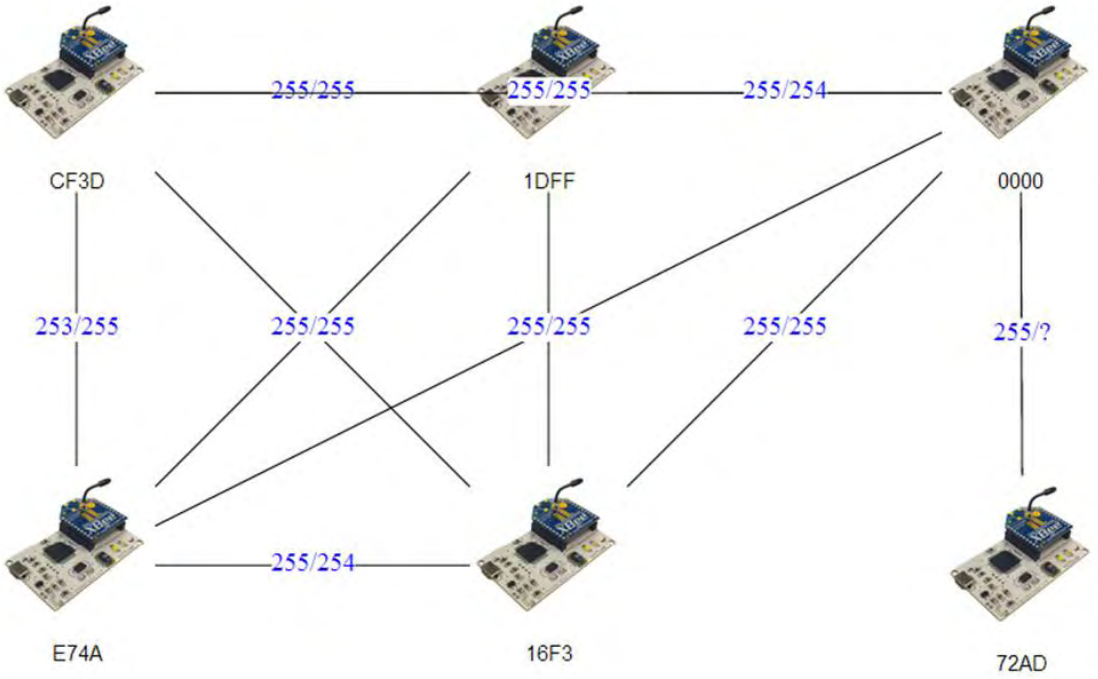


After the response reception the device issues one by one Management LQI Request (0x0031) ZDOs, which is the ZDO that acquires the LQI table from the node, to every node in the network and the nodes respond with the raw information in the form of a Management LQI Response (0x8031) ZDO. The routing table is acquired after the LQI table through a Management Rtg Request (0x0032) ZDO and the response is sent in a Management Rtg Response (0x8032) ZDO. The device stores the information received in order to send them back when the data are requested through another HTTP Request issued to the device from the server that bears the identifier "/getdata".

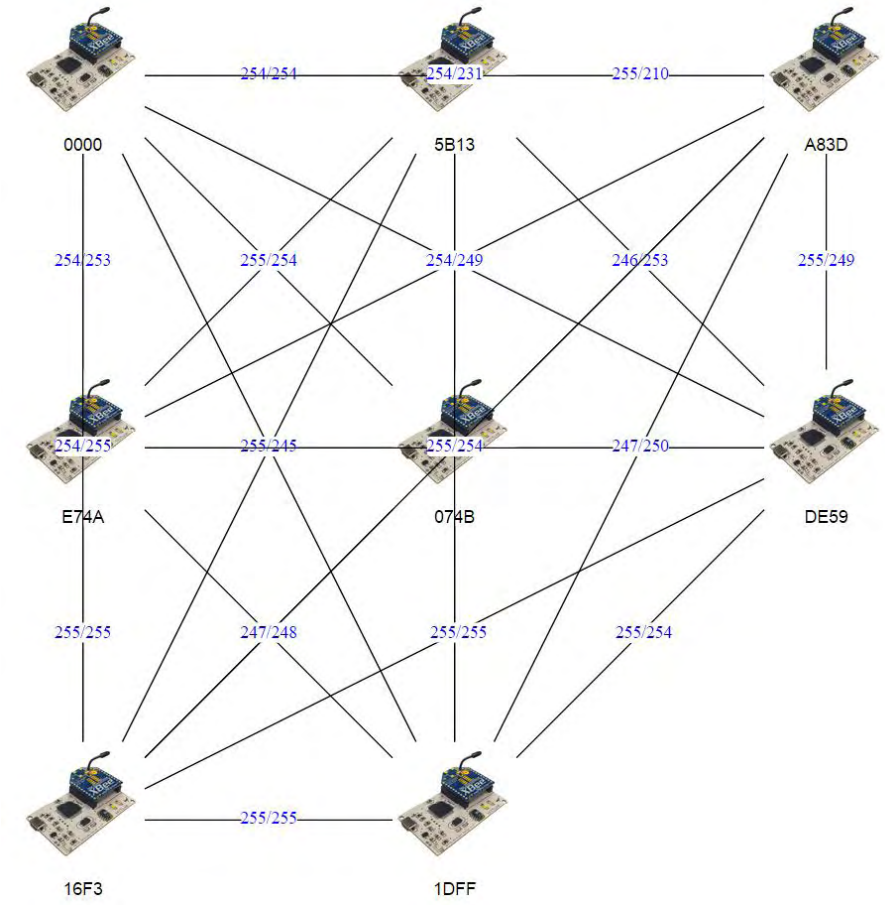


Subsequently, the data are stored in a MySQL server on the NITOS Server and are depicted using an elegant User Interface which supports movable objects. The UI is constructed using PHP and a JavaScript [31] Diagraming framework called JointJS [29].

The visual outcome of the of the topology acquisition using the NITOS WSN Topology Tool is shown in the picture below:



A second screenshot that displays a more complex WSN topology:



4.2 WSN Graphic Depiction Tool

The WSN Graphic Depiction Tool is a Web based GUI designed to reduce the gap that lies between the users and the information that exists in a WSN and more specifically the NITOS Indoor WSN. Rather than illustrating the data as obtained by a WSN Gateway in their original format, which may be difficult and unpleasant to a user, the WSN Graphic Depiction Tool presents that data in a user-friendly Tool that consists of several tabs. The WSN Graphic Depiction Tool displays real-time measurements and graphs from data obtained from different sensors that feature on the NITOS WSN motes.

These sensors are:

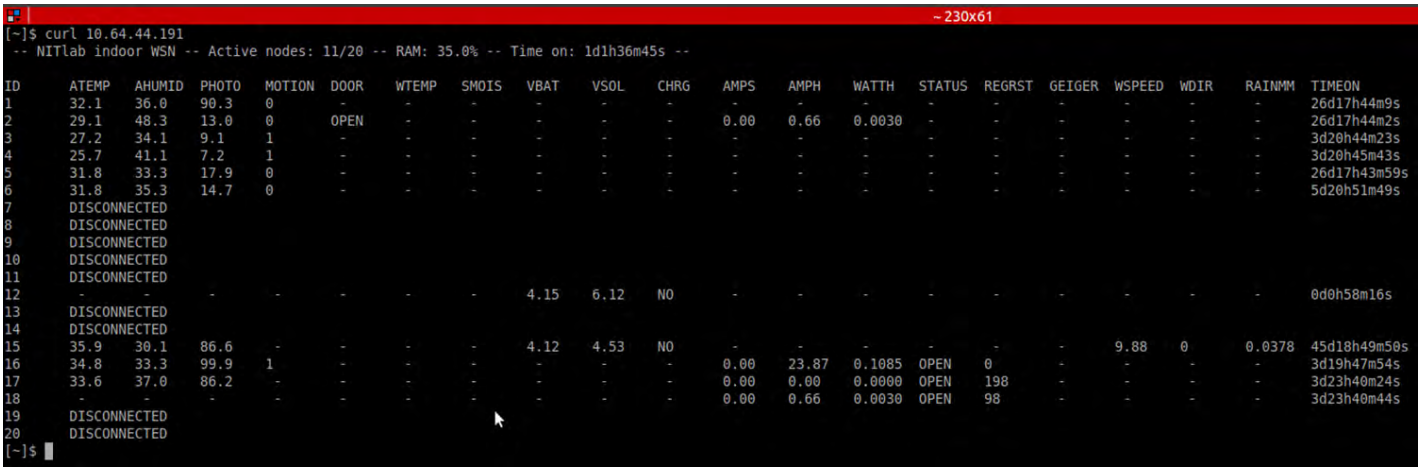
- Temperature, Humidity, Luminosity
- Human Presence
- Radiation
- Energy Consumption
- Plant soil moisture, soil temperature
- Camera

4.2.1 Hardware Setup Used

The hardware used it mainly consists of the NITOS Indoor WSN that hosts a wide variety of sensors and several devices already implemented and deployed in NITlab. Apart from the NITOS Indoor WSN , which we presented in the previous section, these devices are an Arduino compatible camera and a pilot WSN deployment formed by a prototype developed in NITlab, the NITOS City Mote.

4.2.2 Software Implementation

The software implementation process is quite similar to the development of a web-platform. The tool was implemented on the NITOS server using some of the latest technologies introduced to the web-development community. These technologies, which will be presented more thoroughly in the forthcoming section, allow for a smooth user experience and seamless real-time data refreshing. The data are collected from a WSN Gateway through HTTP Requests which are issued from a Ruby script that runs on the NITOS Server. The image below shows the raw data displayed by the WSN Gateway.

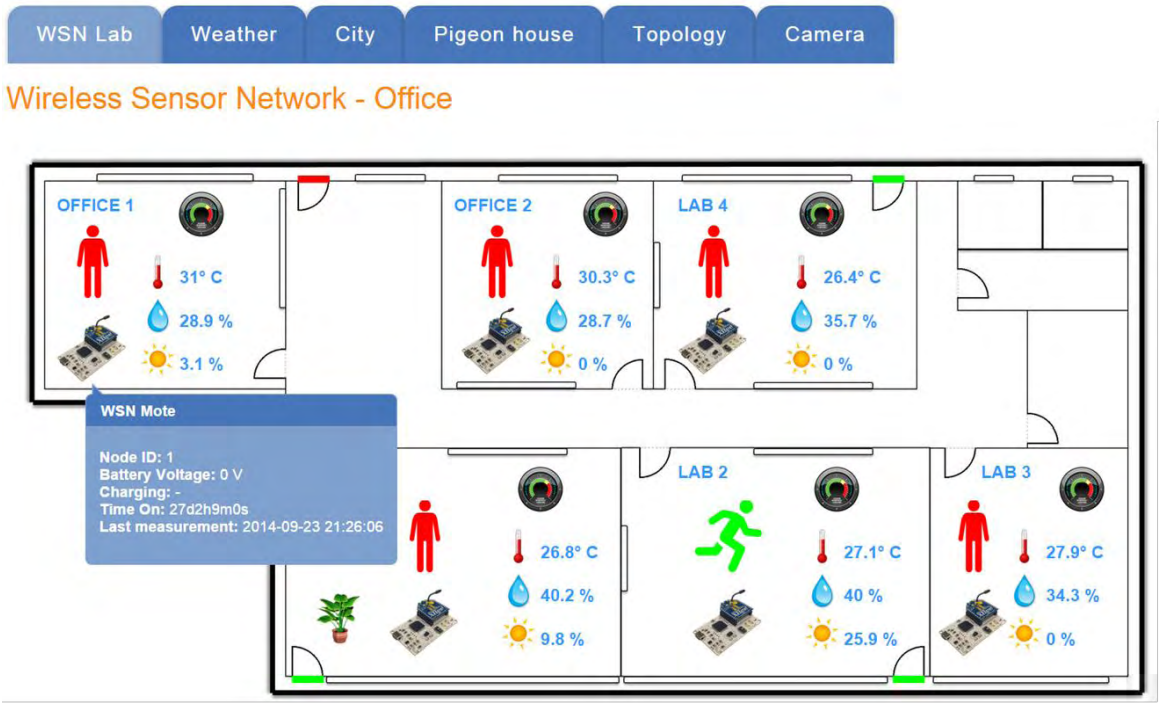


Afterwards the information are stored in a MySQL database and the tool displays the information in a web browser.

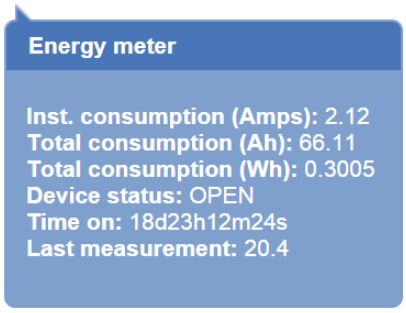
There are tabs to separate the depiction of different implementations such as the measurements of the NITOS Indoor WSN, the NITOS WSN, the NITOS WSN City Map and the Camera. The WSN Graphic Depiction tool is shown in the pictures below.

In the WSN Lab tab we display the measurements collected from the WSN. These measurements are the temperature, humidity, luminosity, human presence and door status (open or closed).

In addition, there are multiple clickable objects that display in a pop-up window specific information of the device clicked, such as the energy monitoring devices, the WSN Motes and some standalone sensors.

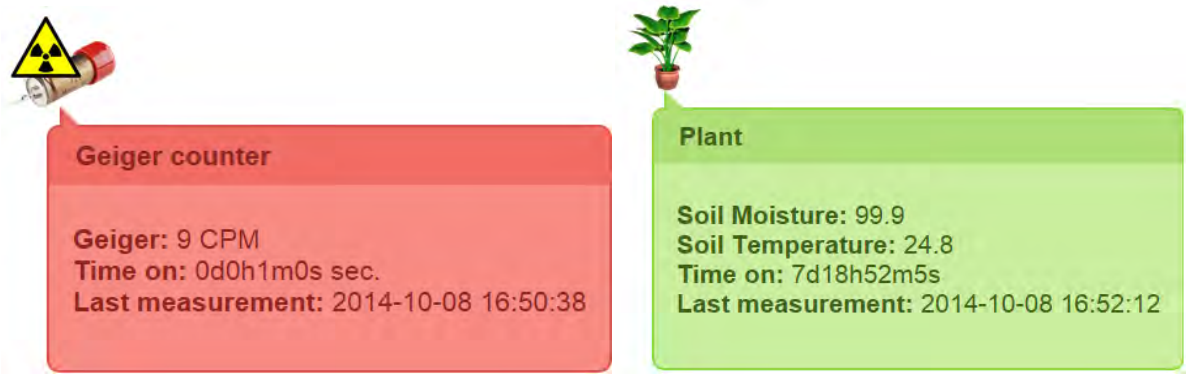


In the energy monitoring devices we display the instant consumption in Amperes, the total consumption in Ampere Hours, the total power consumption in Watt Hours and the device status (open or closed).

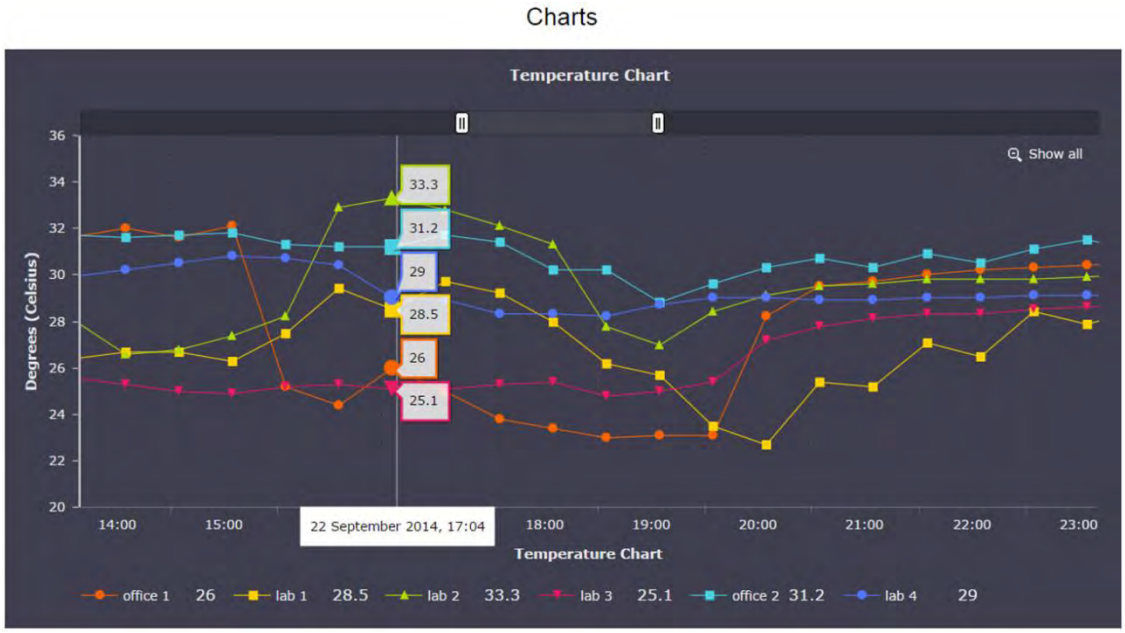


In the WSN Mote pop-up we display the Node ID, Battery Voltage and whether the battery is charging if the device has a battery attached, the time the device has been on-line and the time of the last collected measurement.

The standalone sensors consist of a Geiger counter and a sensor mounted on a plant dedicated to measure soil moisture and temperature.



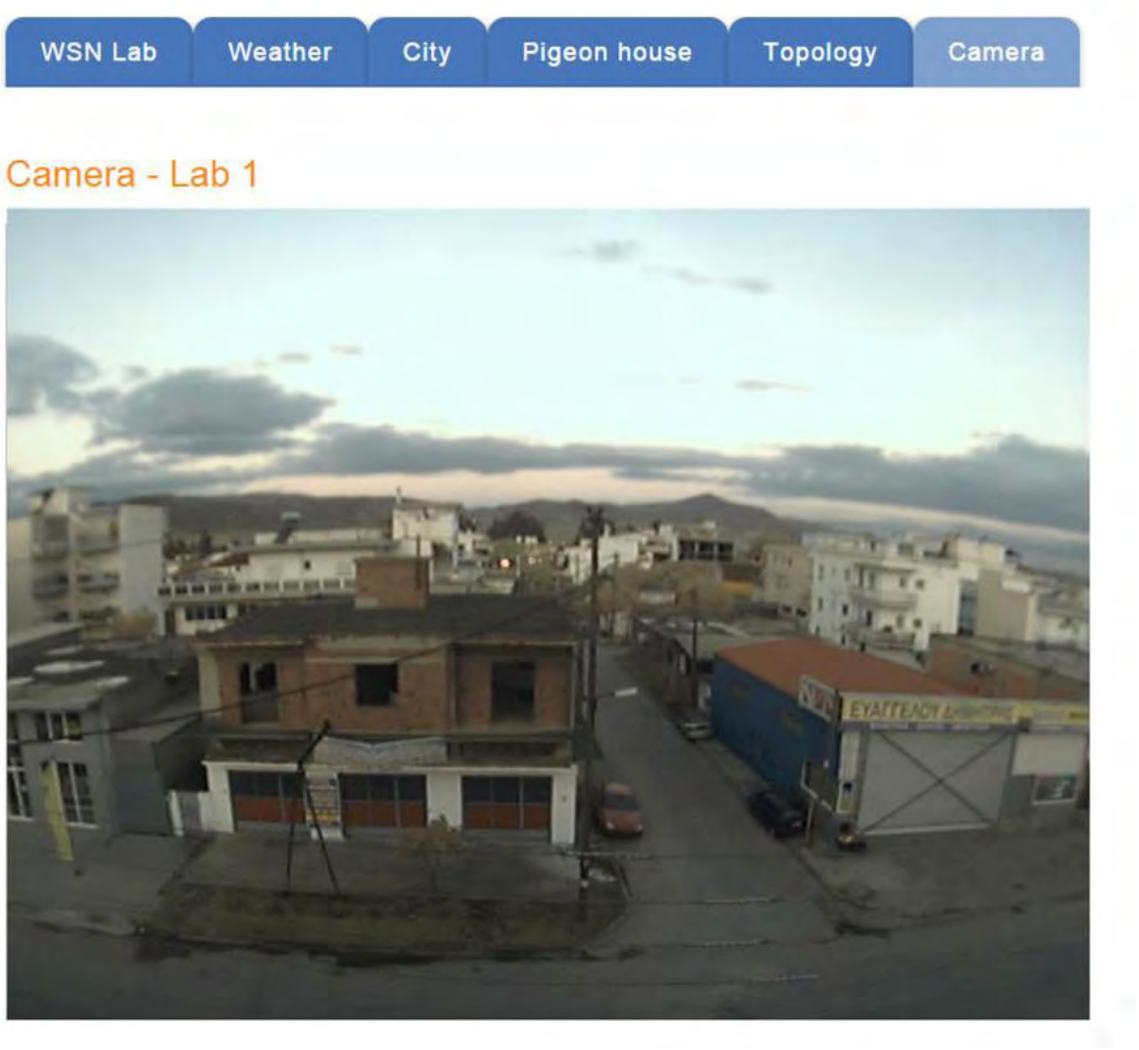
Furthermore, in the same tab there are charts that depict the measurements the temperature, humidity, luminosity and human presence of each room in the WSN Lab.



Moreover, the City tab displays a map and some markers to display an experimental deployment of the NITOS City WSN. The markers, when clicked, display the information collected from each WSN mote.

City Map

In addition, the Camera tab displays an image which is periodically taken from a Camera module interfaced with an Arduino Board. After the picture is taken a Ruby script collects the image, stores it in the NITOS Server and it is illustrated through the WSN Graphic Depiction Tool.



Also featured in the Topology tab is the WSN Graphic Depiction Tool which displays the data collected from the aforementioned Topology Tool.

4.2.2.1 Software Frameworks Used

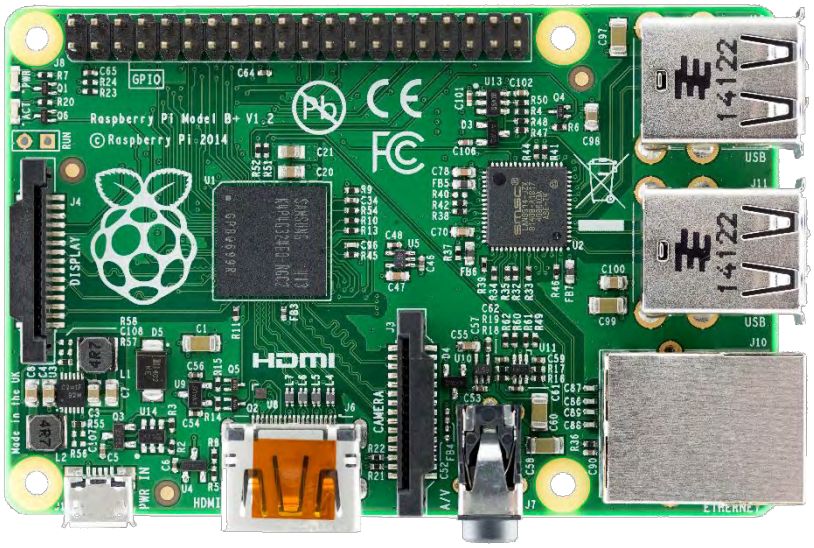
The NITOS WSN Graphic Depiction Tool uses a wide variety of frameworks for the illustration of the information collected. More specifically, the tabs are displayed using HTML5 [30] and pure CSS3 [33]. In the WSN Lab tab we used the HTML5 SVG to draw and position the images, a JavaScript framework called qTip which is a jQuery plugin used to make the objects clickable and display information in a pop-up window. In addition, we used AJAX [35] to refresh the content of the web-page without refreshing the whole image. The charts were implemented using an open-source JavaScript library called ChartJS [32].

4.3 Raspberry Pi WSN Gateway

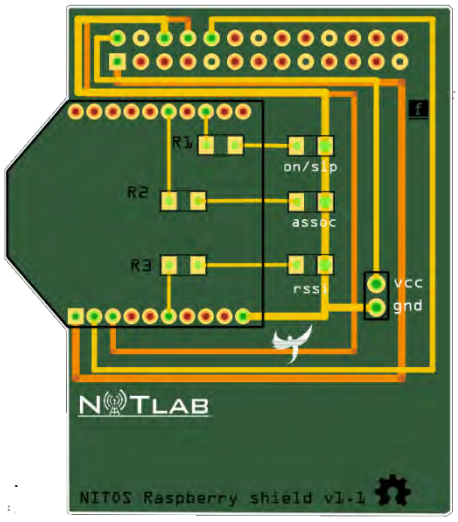
The requirement for a WSN Gateway that features higher capabilities, is more versatile and can integrate more features than a limited Arduino-based Gateway, lead towards the implementation of the Raspberry Pi WSN Gateway. The Raspberry Pi platform was chosen because it has a small form factor, similar to the Arduino board, it features 4 USB ports that can host different devices, it embeds an Ethernet port, it provides several General Purpose Input Output (GPIO) Pins and it is low cost (roughly 25-30$). The Raspberry Pi has 2 pins for RX-TX communication. Using those pins we can achieve a UART communication with a wireless interface.

4.3.1 Hardware

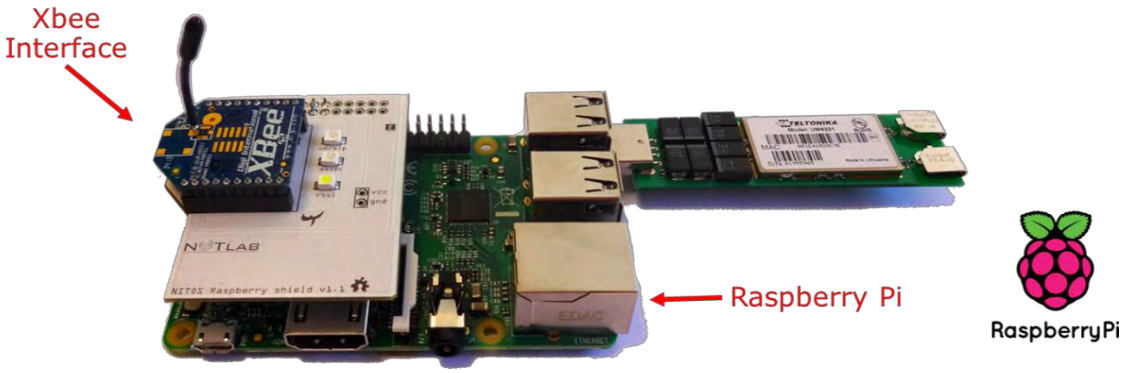
The Raspberry Pi model that we used is the latest Model B+. The B+ supports up to 4 USB connections, has more GPIO pins and consumes less power than its predecessors.



For the needs of the implementation we designed a custom-made XBee shield using the Fritzing Program [20] which is a software tool used to design Printed Circuit Boards (PCBs).



The wireless interface of the WSN Gateway is based on the XBee module. For this implementation we used an XBee Series 2.



Raspberry-like Gateway node

4.3.2 Implementation

The Raspberry Pi needed software configuration in order communicate with an XBee through a serial port [36]. The configuration steps are described below:

First of all, we uploaded to the Pi's microSD a system image called Raspbian (it is a free operating system optimized for the Raspberry Pi and based on Debian). Afterwards, we needed to edit the **/boot/cmdline.txt** file:

```
sudo cp /boot/cmdline.txt /boot/cmdline.txt.bak # Backup file
sudo vi /boot/cmdline.txt
```

Then remove all references to ttyAMA0 (console and kdgboc) so the file looks similar to

```
dwc_otg.lpm_enable=0 rpitestmode=1 console=tty1 root=/dev/mmcblk0p
rootfstype=ext4 rootwait
```

this:

Subsequently we edited the **/etc/inittab**:

```
sudo cp /etc/inittab /etc/inittab.bak # Backup file
sudo vi /etc/inittab
```

By commenting out the following line:

```
2:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100
```

Then we reboot the Raspberry Pi, connect the custom made XBee shield to the Pi and attach an XBee S2 configured to operate as a WSNs Coordinator. We can test the configuration by using Minicom.

```
sudo minicom -b 9600 -o -D /dev/ttyAMA0
```

To imitate the function of an Arduino WSN Gateway (Receive data from the motes and run a web-server to display the information) we ported from an already existing WSN Gateway Arduino code to Python. We wrote a Python script to read the serial pipe and to store data to a file.

For the web server we used a Ruby script which implements a REST API in order to serve the file when an HTTP Request is issued.

4.4 Raspberry Pi Gateway supporting 4G communication backbone

One of the many features the Raspberry Pi WSN Gateway can integrate is the support for a 4G communication backbone. In the context of this thesis we implemented the 4G communication backbone using the widely known technologies of WiMAX [38] and LTE [40].

The communication was achieved through USB dongles attached to the Raspberry Pi WSN Gateway. The dongles used are a Teltonika [39] UM6250 WiMAX USB Dongle and a Huawei [41] E392 LTE USB Dongle.

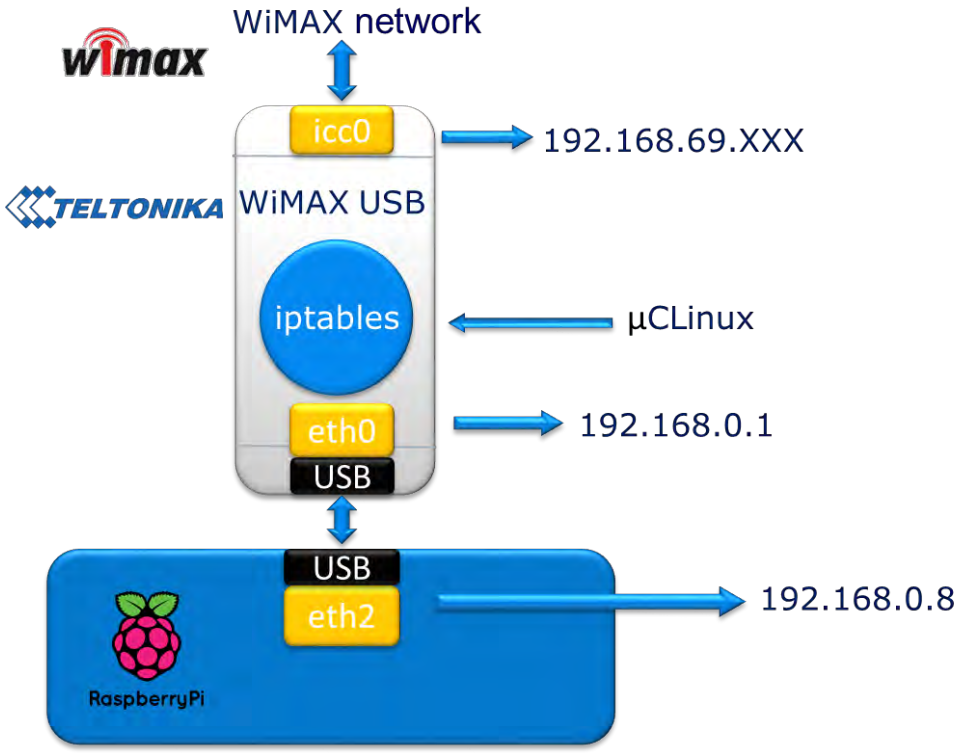
Huawei LTE USB Dongle

Teltonika WiMAX USB Dongle

The USB Dongles did not work right out of the box. Certain modifications were required to be made.

4.4.1 WiMAX

The WiMAX dongle installed on the Pi are provided by Teltonika with model UM6250. These devices are using a small Linux client running on them which is configured to serve the WiMAX network to the Linux OS as a simple Ethernet device. When we connect the Teltonika WiMAX USB Dongle to the Pi there is no need to install new drivers on the Pi, because the drivers are already stored in the dongle. The dongle creates a new Ethernet interface over the USB connection and has another interface called icc0 which is the interface that receives WiMAX traffic from the WiMAX base station. We have to assign the 192.168.0.X IP address to the Ethernet interface and then use the telnet command to connect to the USB dongle using the 192.168.0.1 address.



The WiMAX dongle is capable of receiving incoming connections because it is running a tiny Linux distribution called µCLinux. After the connection has been initiated we configure the iptables of the WiMAX dongle in order to swap the destination header of packets that are destined for the icc0 interface with a new destination, the Ethernet interface. The next step is to route the traffic of the WSN Gateway through the Ethernet interface that is created from the WiMAX USB Dongle

After those steps we have internet connection through the WiMAX Base Station.

The series of commands are shown below:

```
ifconfig tel0 192.168.0.46/24 up
telnet 192.168.0.1 700
/bin/iptables -t nat -A PREROUTING -d 192.168.55.XX -j DNAT --to 192.168.0.YY
route del default gw dev tel0
route add -net 192.168.55.0/24 gw 192.168.0.1 dev tel0
```

4.4.2 LTE

The configuration process for the LTE USB Dongle set-up are different. The LTE Dongle features a slot for a SIM card which means that the LTE connection specifics are taken care of by the USB Dongle.

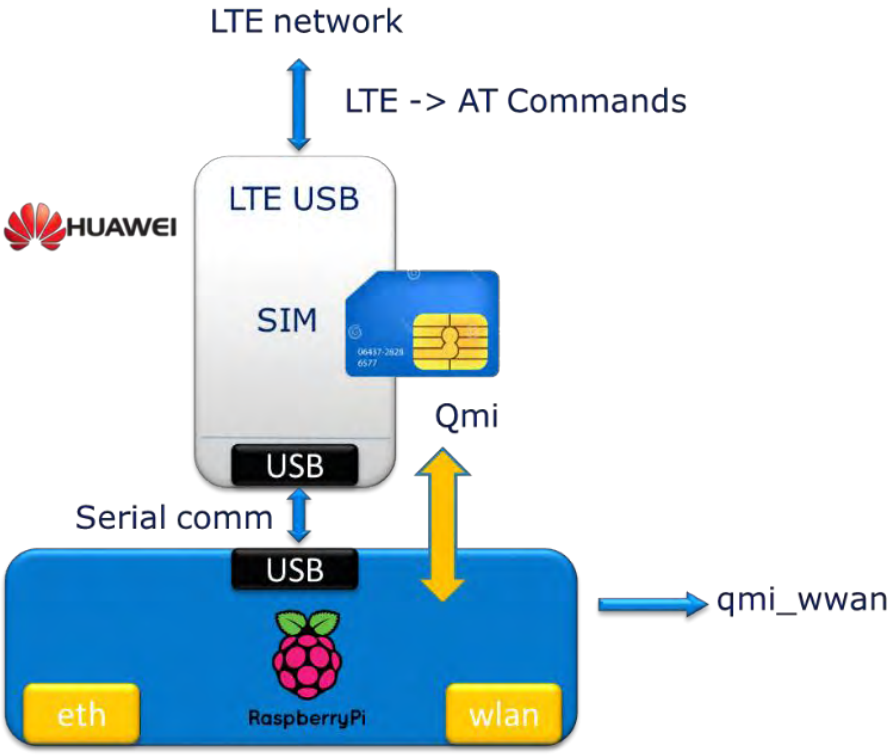
When we connect the LTE USB Dongle to the Raspberry Pi we must install drivers for the device. These drivers are called **qmi_wwan** and if they are not present in the system we can install them with a simple **sudo apt-get install** command.

The USB Dongle connects to the Raspberry Pi using a serial port. In order to have communication with the LTE network we need to send some AT commands to the LTE USB Dongle. We simply open a serial communication port using e.g. Minicom and we send these commands over serial.

The sequence of the AT commands sent are:

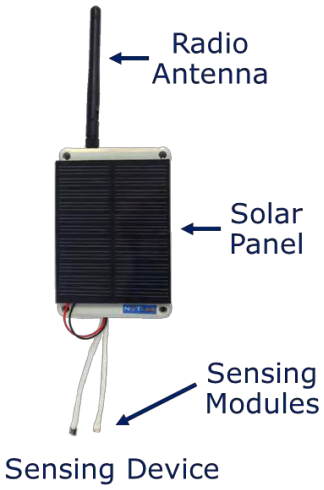
```
AT+CGDCONT=1,1,"default"
AT^NDISDUP=1,1,"default"
AT+CGATT=1
```

When connected, we set a specific IP address, which is assigned from the Evolved Packet Core (EPC), in the internal WWAN interface of the Dongle and then the connection is up and running.



4.5 Sleep mode integration in NITOS WSN City Mote

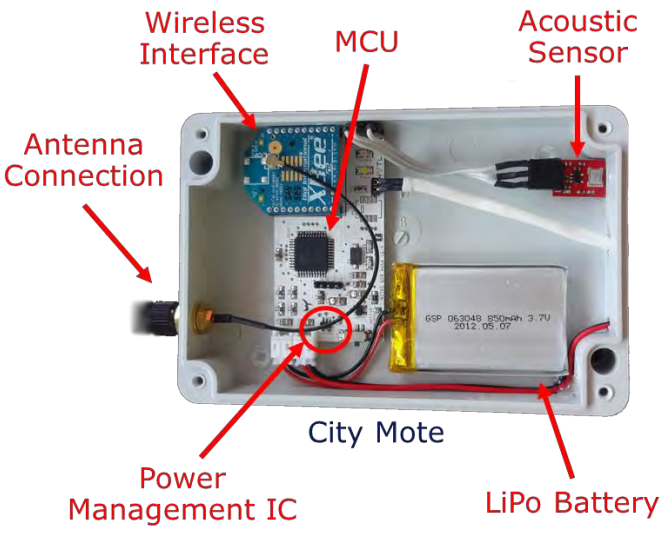
NITlab has deployed a small-scale pilot WSN in Volos City, consisting of 15 nodes deployed around the NITlab facility that reports air temperature humidity and noise pollution.



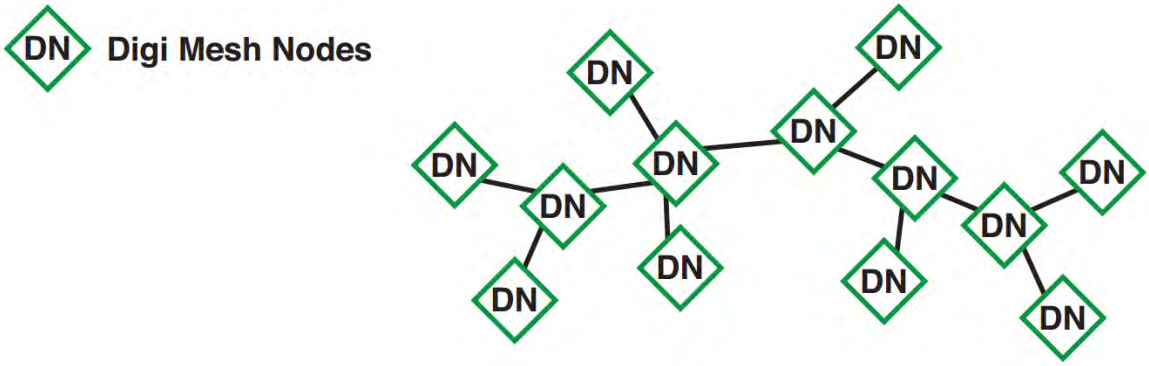
In the context of this thesis we added an extra feature to the existing prototype NITOS WSN City Mote. This extra feature is the integration of a sleeping mechanism that allows every node in a mesh network to sleep in order to conserve energy. The feature is of high importance because it allows a WSN that is based on WSN Motes which are battery powered and are charged using solar panels to operate autonomously for long periods of time.

4.5.1 Hardware

The hardware on which the sleeping mechanism was developed is the NITOS WSN City Mote. The city mote among the standard features of a Wireless Sensor Mote (Microprocessor, Wireless Interface, a variety of sensors) embeds a Li-Po battery which is



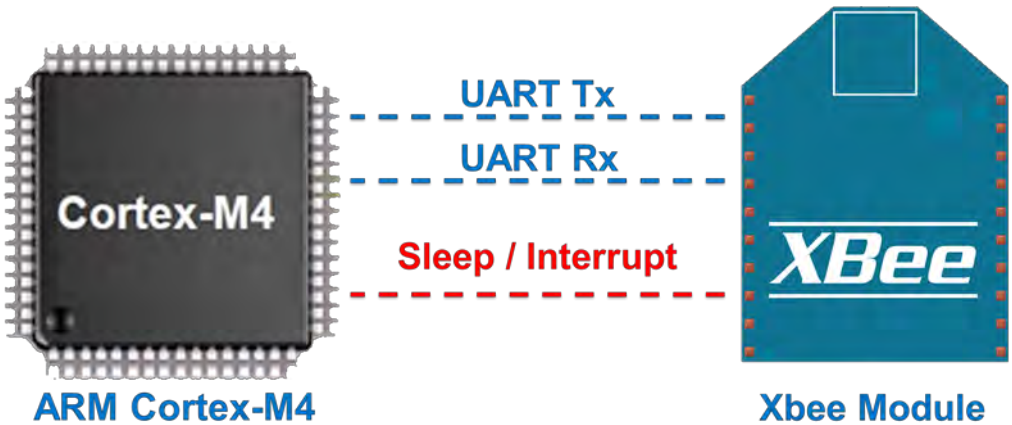
recharged by a solar panel. The Microprocessor is an ARM Cortex-M4 and the Wireless interface is comprised of an XBee Series 1 module. This module can operate in the DigiMesh™ mode which allows for a Mesh network with sleeping routers. The network topology is as shown in the picture below:



Where the XBee which is mounted on the WSN Gateway acts as a coordinator to the entire network, which means that every node sends the measurements it collects to the WSN Gateway.

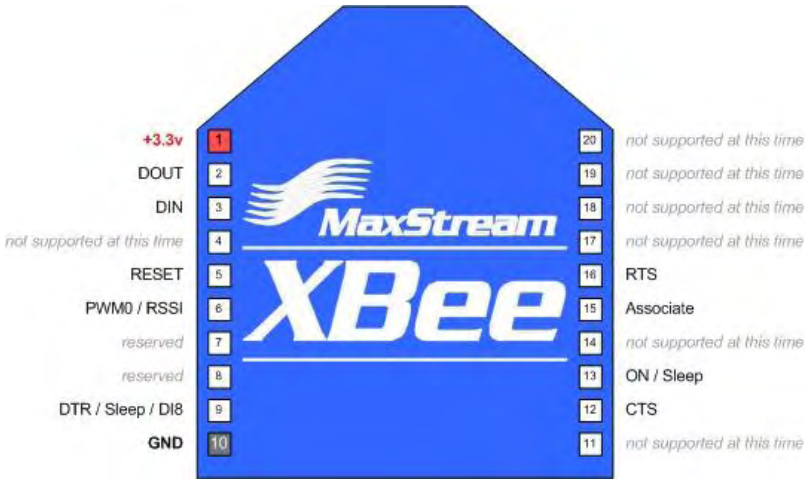
4.5.2 Implementation

In order to implement this feature we configured the XBee module to operate in a Coordinated Cyclic Sleep Mode. This mode is supported by the XBee modules and can make the XBee modules of an entire DigiMesh network to sleep and wake up at the same time. The sleep coordination is taken care of by a sleep coordinator which is assigned statically or elected by the modules. The sleep coordinator broadcasts the sleep and wake time values to the other XBee modules. Furthermore when the XBee sleeps it has a dedicated pin that can be sensed to determine whether the module is awake or asleep.



MCU - XBee Communication

This pin is pin number 13 on the XBee pinout. This pin is HIGH when the xbee is operating and LOW when the XBee is asleep. Another pin, the CTS pin has exactly the inverted output of the ON/Sleep pin. When the XBee is asleep this pin is HIGH and when it wakes up the pin is set to LOW.

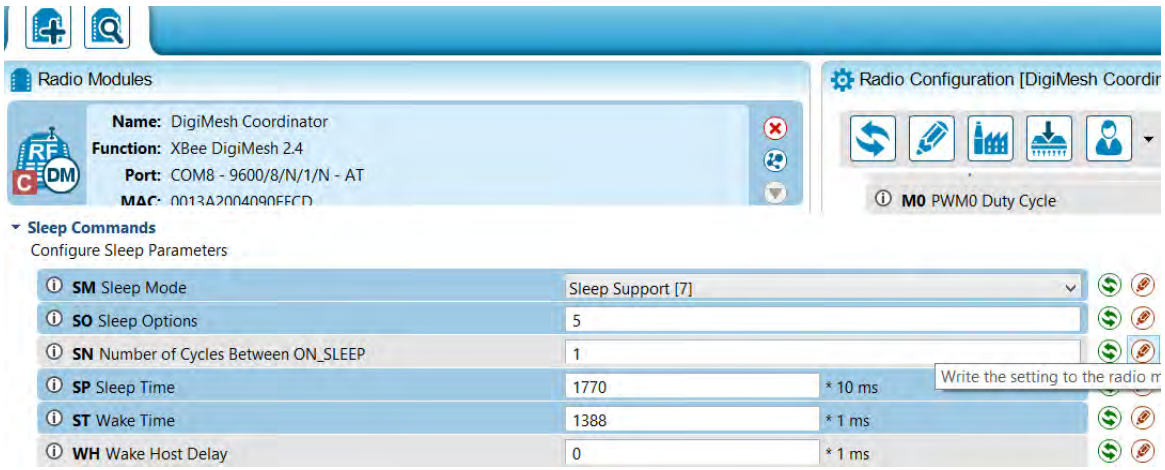


To enable the microprocessor to enter a low power sleep mode we used an Arduino library called LowPower_Teensy3 [42]. This library is designed to operate with the Teensy 3.0 and supports the sleep mode that we wanted to implement. In addition, this library gives us the capability to downclock the ARM microprocessor in order to conserve power whenever the full processing power of the ARM Cortex-M4 is not needed. The Teensy 3.0 is a single board microcontroller which bears the Cortex-M4, the same as the NITOS WSN City Mote. Because the LowPower_Teensy3 library wakes the ARM microprocessor when the interrupt is set to 0, we attached the CTS pin to a pin on the ARM Cortex-M4 microcontroller, which we set as an interrupt pin in order to wake the microprocessor and started sampling this pin to determine if the XBee is awake or asleep. The microcontroller senses when the XBee has gone to sleep and enters a low power sleep mode.

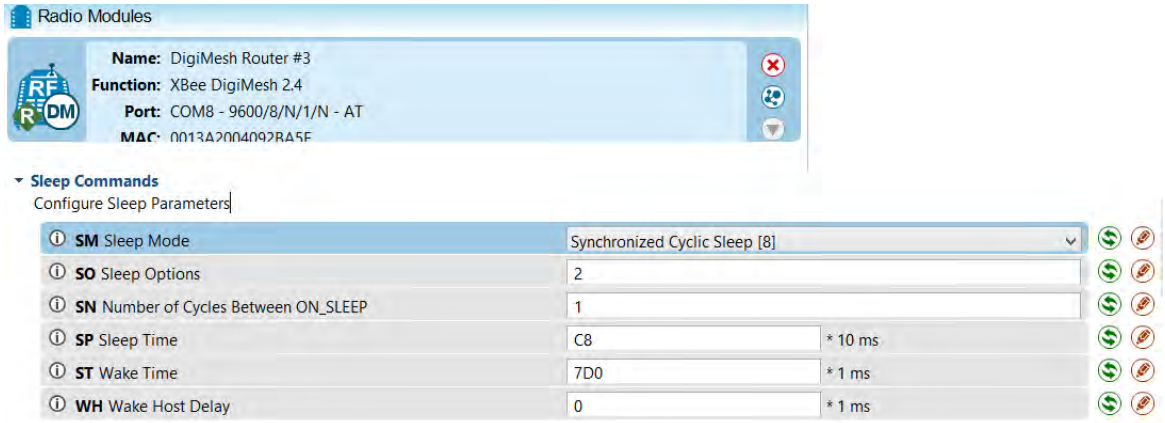
4.5.2.1 XBee configurations

In order to have a sleeping capable network we had to configure the XBee S1 settings [43]. These settings are done through the XCTU program. We had to configure the XBee which is mounted on the WSN Gateway as a sleep coordinator. The WSN Gateway is always awake and is the best candidate to be a sleep coordinator.

In the coordinator we set the **SM** (Sleep Mode) variable of the **XBee to Sleep Support (7)** which means awake but sleep aware, the **SO** variable to 0x05. We also have to set the **SP** (Sleep Time) to our preferred sleep time for the entire network and the **ST** (Wake Time) to the time we want our network to be awake. These values (SP and ST) are set in a hexadecimal representation.



In every other XBee in the network which acts as a router (sleeping devices) we have to set the only the **SM** variable to **Synchronized Cyclic Sleep (8)**. The Sleep Time and Wake Time values are sent to the Routers from the Sleep Coordinator.



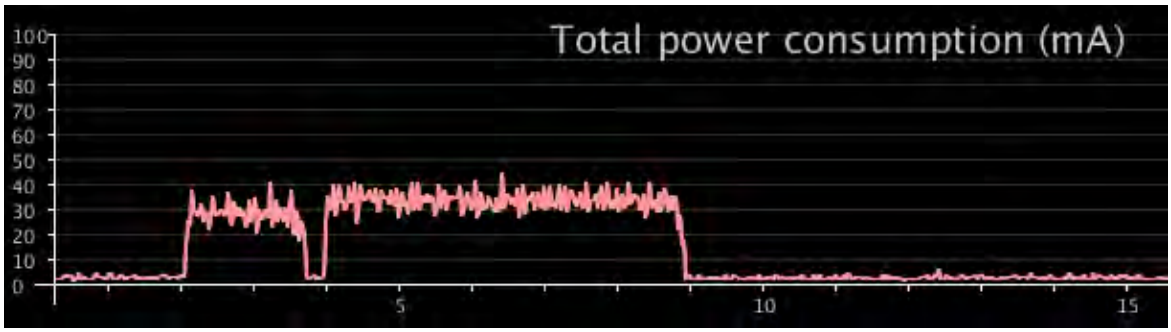
After these settings we will have a network that sleeps for SP*10 ms and operates for ST*1 ms.

4.5.3 Power Consumption Measurements

In addition to the coordinated sleep mode we collected some power consumption measurements from the NITOS WSN City Mote that are presented below:

Clock speed	Awake (ARM + XBee)	Asleep
2 MHz	52 mA	0.01 mA
4 MHz	53.5 mA	0.01 mA
8 MHz	56.2 mA	0.01 mA
16 MHz	58.9 mA	0.01 mA
24 MHz	65 mA	0.01 mA
48 MHz	69.5 mA	0.01 mA
96 MHz (overclocked)	74.2 mA	0.01 mA

We measured the power consumption of the NITOS WSN City Mote. The first burst indicates that the mote has been awaken in order to acquire temperature measurements, while the second one shows the fire of the XBee timer that awakes the entire mote. In this last case the XBee receives the collected temperature value from the microprocessor and transmits it through the established mesh network to the Gateway node.



References

[1] Wireless Sensor Networks: https://www.cs.virginia.edu/~stankovic/psfiles/wsn.pdf
[2] Institute of Electrical and Electronics Engineers : http://www.ieee.org/index.html
[3] 802.15.4 vs ZigBee : http://www.sensor-networks.org/index.php?page=0823123150
[4] IEEE 802.15.4: http://en.wikipedia.org/wiki/IEEE_802.15.4
[5] ZigBee protocol : http://www.zigbee.org/
[6] Wireless HART : http://www.hartcomm2.org/index.html
[7] ISA-SP100 : http://www.isa.org/MSTemplate.cfm?MicrositeID=1134&CommitteeID=6891
[8] IETF IPv6 LoWPAN : http://ietfreport.isoc.org/ids-wg-6lowpan.html
[9] MiWi: http://www.microchip.com/pagehandler/en-us/technology/personalareanetworks/technology/home.html
[10] Digi Mesh : http://www.digi.com/technology/digimesh/
[11] AODV Routing Protocol : http://en.wikipedia.org/wiki/AODV
[12] Digi : http://www.digi.com/
[13] X-CTU program : http://www.digi.com/products/wireless-wired-embedded-solutions/zigbee-rf-modules/xctu
[14] Arduino : http://www.arduino.cc/
[15] ARM: http://www.arm.com/
[16] Arduino DUE: http://arduino.cc/en/Main/arduinoBoardDue
[17] Arduino Ethernet: http://arduino.cc/en/Main/arduinoBoardEthernet
[18] Atmega 32U4 Microcontroller : http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Dev/Arduino/Boards/ATMega32U4.pdf
[19] NITOS Wireless Sensor Mote: http://nitlab.inf.uth.gr/NITlab/index.php/hardware/sensors/nitos-wireless-sensor-platform
[20] Fritzing Software : http://fritzing.org/
[21] Arduino Ethernet Board : http://arduino.cc/en/Main/ArduinoBoardEthernet
[22] Atmel Corporation : http://www.atmel.com/
[23] XBee: http://en.wikipedia.org/wiki/XBee
[24] XBee radio interfaces : http://www.digi.com/xbee/
[25] XBee Series 1 802.15.4: http://www.digi.com/products/wireless-wired-embedded-solutions/zigbee-rf-modules/point-multipoint-rfmodules/xbee-series1-module#specs
[26] XBee Series 2 ZigBee: http://www.digi.com/products/wireless-wired-embedded-solutions/zigbee-rf-modules/zigbee-mesh-module/xbee-zb-module#specs
[27] Zigbee Device Objects: http://ftp1.digi.com/support/images/APP_NOTE_XBee_ZigBee_Device_Profile.pdf
[28] Raspberry Pi: http://en.wikipedia.org/wiki/Raspberry_Pi
[29] JointJS: http://jointjs.com/

[30] HTML 5: http://en.wikipedia.org/wiki/HTML5
[31] JavaScript: http://en.wikipedia.org/wiki/JavaScript
[32] ChartJS: http://www.chartjs.org/
[33] CSS: http://en.wikipedia.org/wiki/Cascading_Style_Sheets
[34] Google Maps API: https://developers.google.com/maps/
[35] AJAX: http://en.wikipedia.org/wiki/Ajax_(programming)
[36] Raspberry Pi and XBee: http://michael.bouvy.net/blog/en/2013/04/02/raspberry-pi-xbee-uart-serial-howto/
[37] NITOS WiMAX: http://nitlab.inf.uth.gr/NITlab/index.php/testbed/wimax-experimentation/simple-wimax-tutorial
[38] WiMAX: http://en.wikipedia.org/wiki/WiMAX
[39] Teltonika: http://www.teltonika.lt/en/
[40] LTE: http://en.wikipedia.org/wiki/LTE_(telecommunication)
[41] Huawei: http://www.huawei.com/gr/
[42] Low power teensy library: https://github.com/duff2013/LowPower_Teensy3
[43] DigiMesh™ Sleep settings: http://www.digi.com/wiki/developer/index.php/Sleep_Settings_within_DigiMesh