SUP'COM

Higher School of Communications of Tunis

PISTE Project 2018

Wireless Sensor-Based IoT Application for Monitoring and Controlling Agro-Environmental Parameters

Authors

Aymen Hamrouni
Dhiaeddine Alioui
Ghassen Allouche
Bechir Nahali
Zoubeir Ibidhi
Houcine Gouadria
Yesmine Bejar
Abdedeyem Zelfani
Firas Guiza
Emna Seddik

Supervisors

Leila Najjar
Maymouna Ben said
Zakia Jellali
Leila Nasraoui
Riadh Abdelfateh
Sofiane Cherif

Abstract

This report describes the required functionalities for the realization of the project "Wireless Sensor based-IoT Application for Monitoring and Controlling Agro-Environmental Parameters". This work is a part of the project PISTE of SYSTEL specialty. It is an integration project where students used their knowledge in various fields of telecommunications to achieve a concrete application. In this project, the team is interested in realizing an IoT application allowing the control of environmental parameters for better regulation of the agricultural irrigation process.

Contents

List of Figures

Introduction

1

Contents

1.1 Motivation

As part of project PISTE for the second year of the engineering cycle at SUP'COM, students have the opportunity to carry out a project allowing them to acquire new knowledge and thus obtain new skills related to their future job.

This project aims to develop a monitoring and control solution for agricultural infrastructures. The expected benefit of this project is to minimize unnecessary farmer movement and design optimized irrigation processes. This procedure enables to improve the management of the use of water resources by minimizing the waste of water. In such context, our team involves designing and developing an IoT application based on wireless sensor networks (WSN) for the acquisition and control of agro-environmental data.
This application allows the collection of the state information of the agricultural environment and more specifically on soil moisture, luminosity and temperature parameters using wireless sensor devices deployed in the agricultural area to be controlled. After sensing step by the sensor nodes, the fusion center (sink) sends the collected measurements to a cloud IoT platform where the analysis phase is carried out for a remote management of irrigation.

1.2 Report organization

The remainder of this repport is organized as follows. Chapter 2 is dedicated to a project organization by presenting the team and the general architecture of the application. Hereafter, we focus on presenting and explaining the different phases of the IoT chain.
In chapter 3, we investigate the phase of data acquisition and transmission through WSN. More precisely, we start by presenting the considered experimental platform. Then, in order to vehicle the sensed data by sensor devices towards a specific

destination an overview on the considered routing protocols will be investigated. At the end of this chapter, we exhibit the adopted scenarios by introducing the experimental environment for the realization of each scenario.
Chapter 44 addresses the phase of border router where the received data at the sink node is forwarded to the Internet via the gateway. In our realization, a Raspberry Pi 3 is used to ensure this transition. Also, the used platform and the database structure will be discussed.
In chapter 5, we focus on WSN collected data analysis. In this part, the fuzzy algorithm will be exploited for intelligent irrigation. The chapter 6 is dedicated to the phase of application development. In this context, the used services in the considered platform and the software choice will be discussed for the user dashboard development.

Project organization

2

Contents

2.1 Team presentation

We are ten students in the 2nd year of the engineering cycle at the higher school of communication of Tunis SUP'COM in the specialty Telecommunication Systems **Systel**. We were split into five groups with five sub-projects.

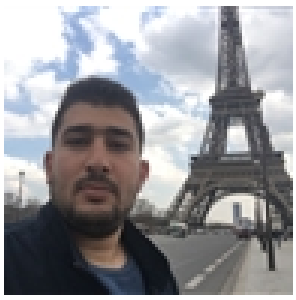
Sub-project 1

Aymen HAMROUNI

Houcine GOUADRIA

- Measure agro-environmental parameters from IoT nodes and send data to the gateway.
- WSN implementation with multi-hop relaying using Rime protocol stack.
- C programming wih Contiki OS.

Sub-project 2

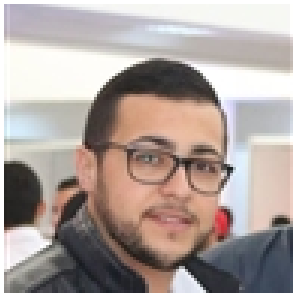


Dhiaeddine ALIOUI

Yasmine BEJAR

- Measure agro-environmental parameters from IoT nodes and send data to the gateway.
- WSN implementation with multi-hop relaying using uIP protocol stack.
- C programming wih Contiki OS.

Sub-project 3

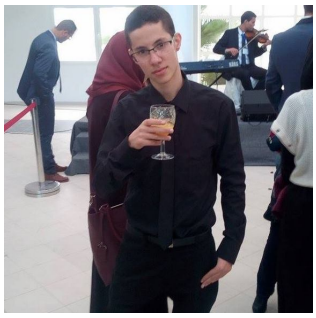


Ghassen ALLOUCHE

Abdedeyem ZELFANI

- Serialize data received from IoT nodes and post to MySQL database.
- Application development with Node-Red.
- Python, MySQL.

Sub-project 4

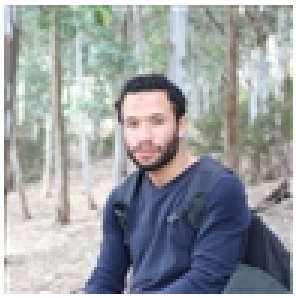


Bechir NAHALI

Firas GUIZA

- Apply the fuzzy logic algorithm on agro-environmental parameters to determine the flow rate and duration.
- Update the database.
- Matlab, R, Python.

Sub-project 5

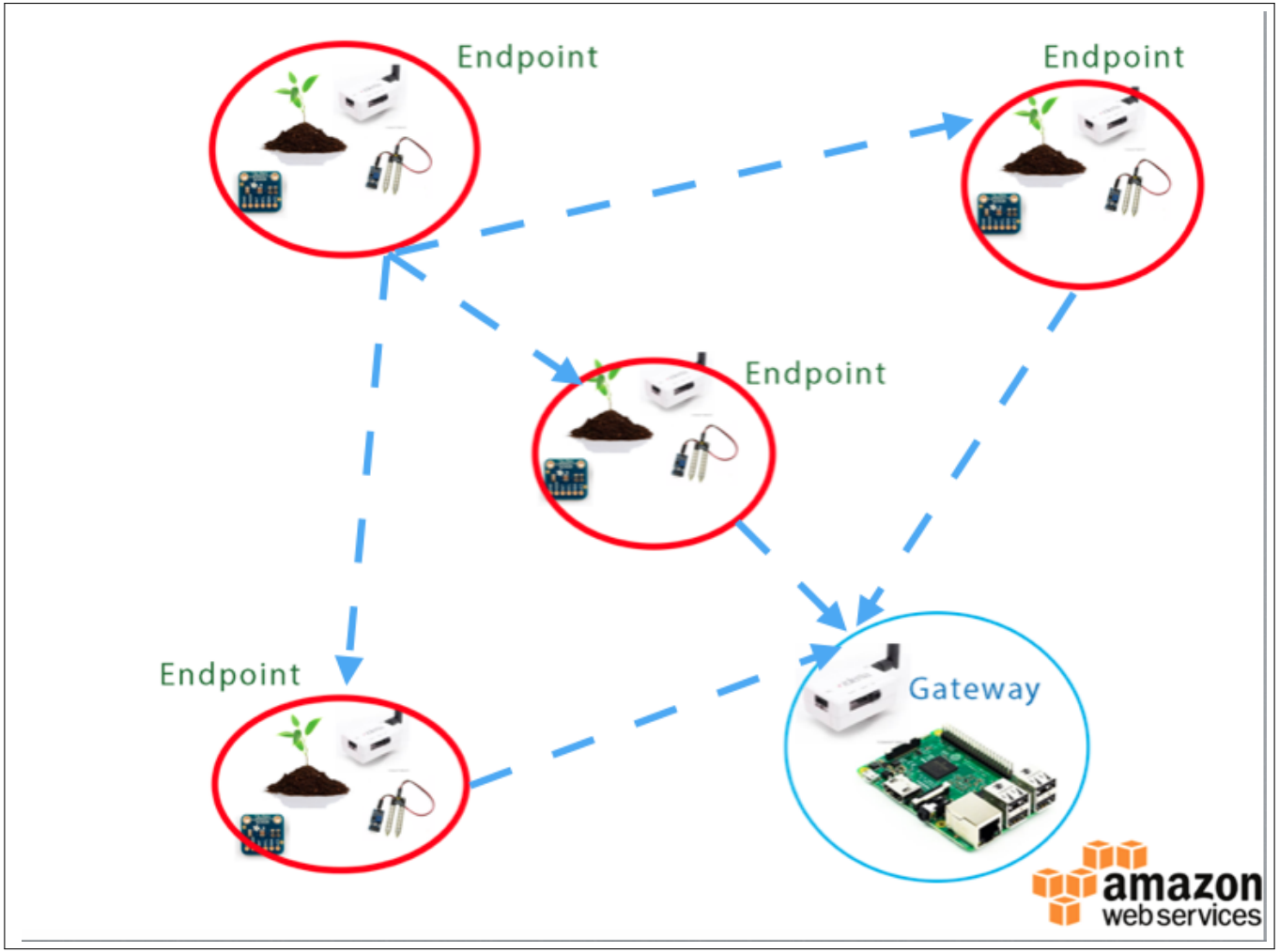


Zoubeir IBIDHI

Emna SEDDIK

- Web site development.
- Virtual server deployment and management : Amazon EC2 instance.
- PHP, CSS, JavaScript, HTML.

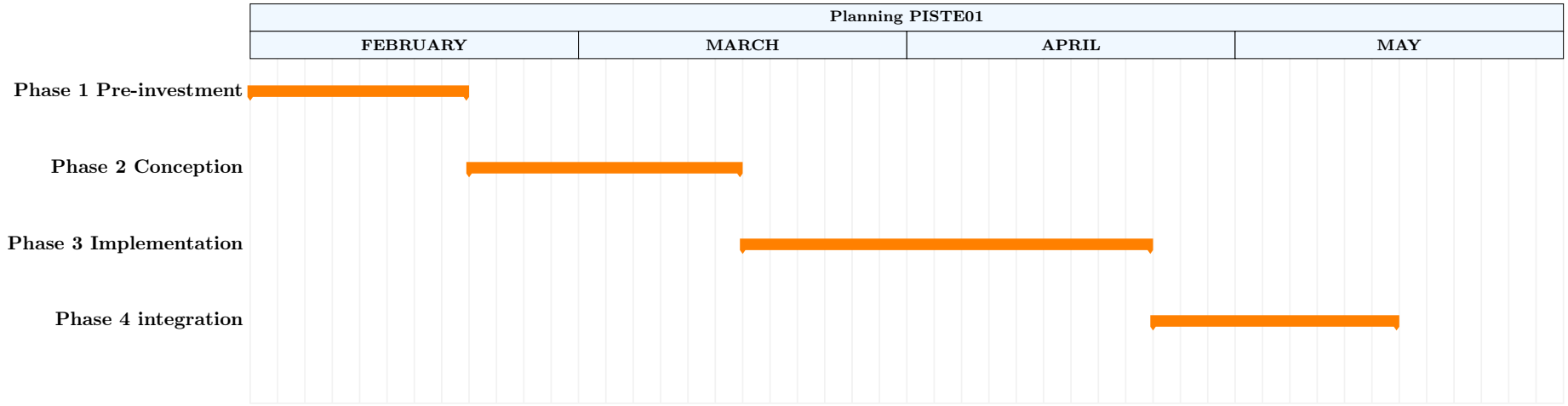
2.2 General architecture



2.3 Project management

The project was planned in a duration of 14 weeks. Tasks were organized into four phases :

1. Pre-investment phase
2. Conception phase
3. Implementation phase
4. Integration phase



3 WSN Data Acquisition and Transmission

Contents

The developed IoT application is based on Wireless Sensor Networks (WSN). As a first phase of this application, we validated the functionality of two following steps:

a. Measurement of environmental parameters such as soil moisture, light and temperature from various sensor nodes.

b. Transmission of the acquired data to a sink node.

3.1 Operating environment : context and technology

In this section, we discuss the 6LowPAN technology and present the experimental scenario with the different used materials for the application realization.

3.1.1 6LowPAN Network

A 6LoWPAN (IPv6 over Low-Power Wireless Personal Area Networks) is a wireless communication networking technology, consisted of a set of equipment with few resources (low power, short range and low cost), allowing the sending and the reception of IPv6 packets via the IEEE 802.15.4 communication protocol. As the full name implies, it provides the connection of low power sensor devices using IPv6 addressing [3].
The physical layer of IEEE 802.15.4 uses frequencies of 868 Mhz, 915 Mhz, and 2.4 Ghz with date rates of 20 kbps, 40 kbps, and 250 kbps respectively.

3.1.2 Experimental platform

Zolertia Z1 mote

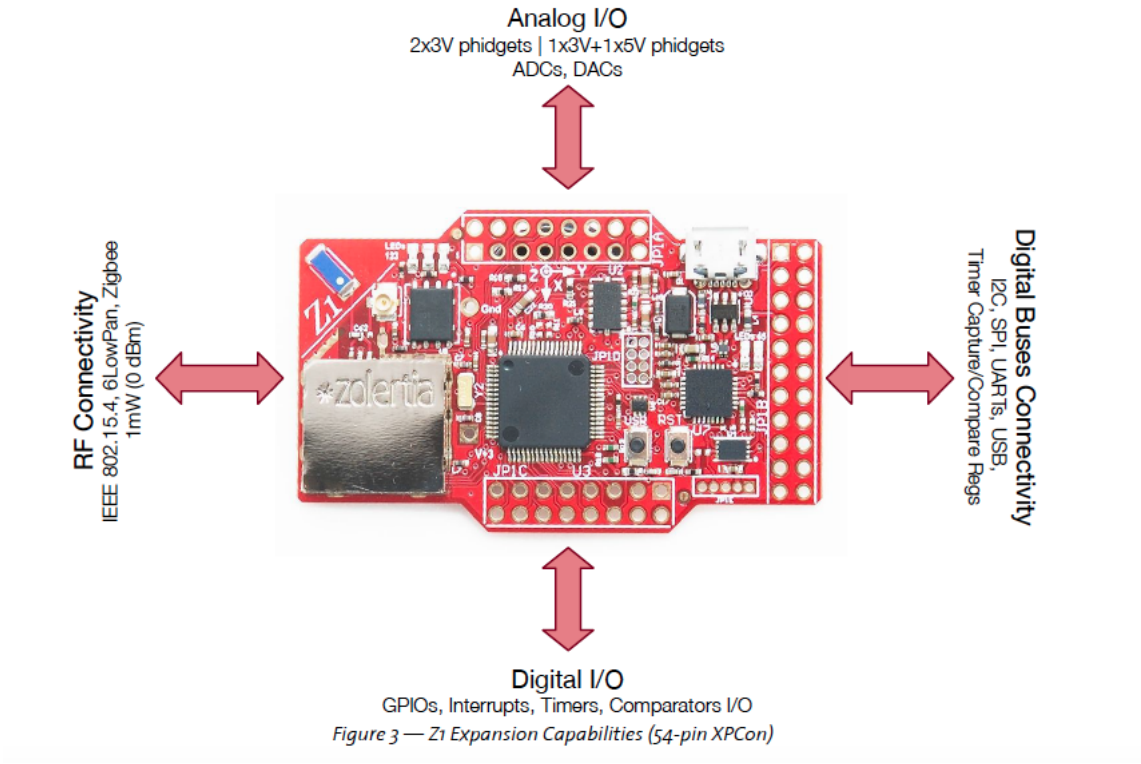


Figure 3.1 – Z1 mote structure [10]

The Z1 is a low power wireless module compliant with IEEE 802.15.4 and Zigbee protocols intended to be used for Wireless Sensor Networks. The main features and specifications of Zolertia Z1 module are listed in the table below.

Feature	Specification
CPU	MSP430F2617 MCU from Texas Instruments RISC 16-bit Architecture
Memory	8KB RAM and 92Ko Flash.
Radio	CC2420 :2.4GHz, 250Kbps
Supported OS	TinyOS, Contiki,OpenWSN, RIOT
Power	3V , 5V ($\mu - USB$)

Table 3.1 – Zolertia Z1 specifications

Z1 sensors

A Z1 mote has 2 internal sensors, and using the external ports, can be connected to a variety of external sensors.

a. **Internal sensors**

- Temperature Sensor : The internal temperature sensor in the Z1 mote is the **Tmp102** sensor from Texas Instruments. This sensor is integrated with the Z1 mote using the I2C interface. It can read the temperature range of $-40°C$ to $+125°C$.
- Accelerometer Sensor : The internal accelerometer in the Z1 mote is the adxl345 from Analog Devices Inc. This sensor is integrated with the z1 mote using the I2C interface.

b. **External sensors** The Z1 mote has several ways to connect sensors.

a. **Analog sensors** The North Port is meant to connect the Z1 module to analog sensors via Phidget connectors $3V$ and $5V$ (Figure 3.1) .
Analogue sensors require being connected to an ADC (Analogue to Digital Converter) to translate the analogue reading to an equivalent digital value in millivolts. When using the Phidget port $5V$, we should take into account the following notes :

* The $5V$ Phidget requires $USB + 5V$ power net.
* **The** $5V$ **phidget** ($ADC0$, $ADC3$) **has** $5:3$ **relationship** : the ADC input is implemented via a resistor divider :

$$V_{in}(\text{Sensed value : pin 5, 15}) = \frac{\text{ADC output (AD0, AD3)} \times 5}{3} \tag{3.1}$$

Features	MSP430 Port#	Pin Name	Pin#		Pin Name	MSP430 Port#	Features
Phidget powered @5V, ADC input has resistor divider to allow 5V inputs		ADCGND	1	2	ADCGND		Phidget @3V
		USB+5V	3	4	Vcc+3V		
	P6.3	ADC3*	5	6	ADC7	P6.7	
	P6.4	ADC4	7	8	ADC6/DAC0	P6.6	
	P6.2	ADC2	9	10	ADC5/DAC1	P6.5	
Phidget powered @5V, ADC input has resistor divider to allow 5V inputs		ADCGND	11	12	ADCGND		Phidget @3V
		USB+5V	13	14	Vcc+3V		
	P6.0	ADC0*	15	16	ADC1	P6.1	

Table 3 — JP1A Pinout description

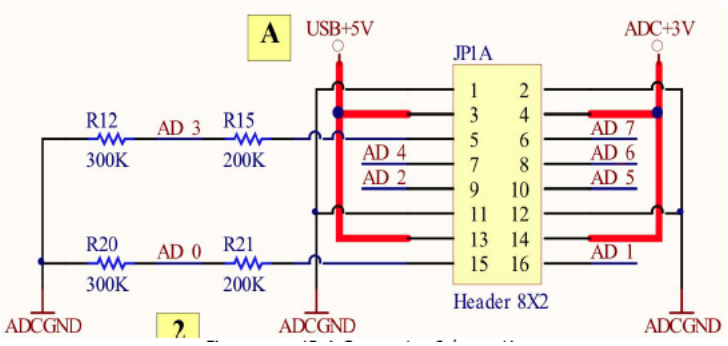


Figure 3.2 – Pinout and connector schematic

b. **Digital sensors** are connected to the Z1 module through the ziglet port.

Contiki OS

We here used the contiki as operating system for the Z1 motes. Contiki is an open source OS which connects tiny low-cost, low-power micro-controllers to the Internet. Contiki supports both IP version 4 (IPv4) and IP of IPv6 (IPv6). Furthermore, contiki provides a 6LowPAN packets over IEEE 802.15.4 networks [5].

3.2 Routing Protocols

The Contiki OS has two communication stacks namely uIP and Rime. uIP is a TCP/IP stack that makes it possible for Contiki to communicate over the Internet and use the UDP or TCP protocols at the transport layer. Rime is a lightweight communication stack designed for low power radio communication with very simple layers.

3.2.1 Rime protocol stack

The Rime stack supports both single-hop and multi-hop communication primitives. The multi-hop primitives do not specify how packets are routed through the network. Instead, as the packet is sent across the network, the application or upper layer protocol is invoked each node is capable to choose randomly the next-hop neighbor (an addressing table is available through the announcement message).
In our experienced scenario, we focused on multihop topology.

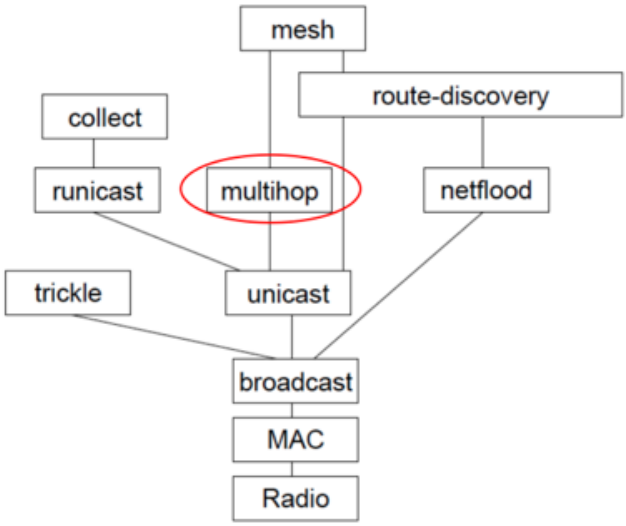


Figure 3.3 – Multihop with Rime

3.2.2 RPL protocol stack

Principle

- Routing Protocol for Low power and Lossy Networks (RPL) is an IPv6 routing protocol designed for low power and lossy networks. The main goal of RPL is to allow constrained devices to be connected via the Internet. The specifics of RPL is that it can build connection between nodes based on a directed Acyclic graphs (DAG). The DAG defines a tree structure that allows designing the default route between nodes.
- RPL defines a set of control messages that allow exchanging information associated to DODAG. The four types of RPL control messages are:
  - DODAG Information Object (DIO): initiated by the LBR (LowPAN Border Router) and retransmitted in multicast by its neighbors.

Transport	UDP
Network	IPv6 / RPL
Adaptation	6LoWPAN
MAC	CSMA / link-layer bursts
Radio Duty Cycling	ContikiMAC
Physical	IEEE 802.15.4

Figure 3.4 – uIP protocol stack

  - Destination Advertisement Object (DAO): used to propagate destination information upwards along the DODAG.
  - DODAG Information Solicitation (DIS): makes it possible for a node to solicit a DODAG information object (DIO) from a reachable neighbor.
  - Destination Advertisement Object Acknowledgement (DAO-ACK): it sent by a DAO recipient in response to a DAO message.

- RPL is based on the Objective Function (OF) to construct roots. The OF defines how a node translates one or more metrics into a Rank value. A node that provides the least Rank from a list of candidate nodes is selected as best parent. The Rank represents the location of nodes in DODAG. Moreover, the OF is separately implemented into the core of RPL specification which enables RPL to be adaptable to different optimization according to some criteria. These criteria allow choosing best parent based on a set of routing metrics. These metrics can be node metrics as hop count and energy or link metrics as throughput, latency, link quality level and number of expected transmission count (ETX). In this way, two objective functions are considered. The first one is the objective function zero (OF0) that uses the minimum hop count as a criterion for selecting best parent. The second one is the minimum rank with modified objective function (MRHOF) with ETX metric to select the optimal route toward sink node.

3.3 Realization : experimental environment

We considered a communication between sensor nodes Z1. More precisely, we adopted a multipoint-to-point topology where we use one Z1 as a sink node and multi-sender Z1. The following sensors were used :

- TMP102, the internal sensor of Z1 : measures temperature.
- Soil moisture sensor SEN-13322 : measures soil humidity.

- Phidgets Light Sensor (1143) : measures luminosity.

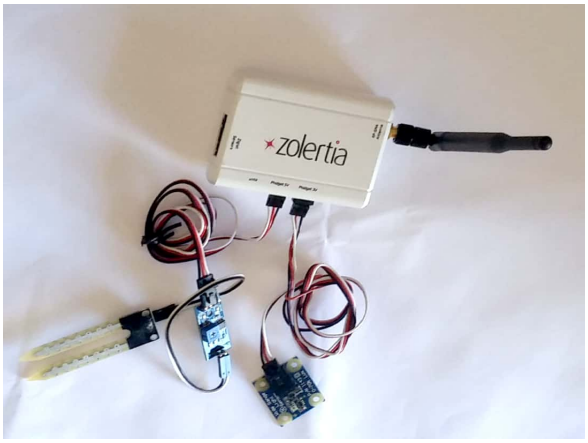


Figure 3.5 – The sender node Z1

Each Z1 sender senses, measures and sends periodically to the sink three agro-environmental parameters : temperature, soil humidity and luminosity.
Two case scenarios were implemented :

- multihop with Rime.
- multihop with RPL.

In the following sections, we give a short description of the used Contiki functions to implement the desired network.

3.3.1 Data Acquisition : sender side

Data structure

The sensed data by the sender nodes is defined using a C structure with 4 fields of 2 byte size : node identifier, temperature, humidity and luminosity. The node identifier is used to associate the sensed data to the corresponding node.

```
struct sensed_data{
uint16_t NodeID ;
uint16_t Humidity ;
uint16_t Luminosity;
uint16_t Temperature;};
/* Declaration of a variable data_to_be_sent  as type sensed_data  */
static struct sensed_data  data_to_be_sent;
static struct sensed_data  *data_to_be_sent_Ptr = &data_to_be_sent;
```

Listing 3.1 – Data structure

Node Identifier

In Contiki, each device has a Node ID which is used to generate the mote's MAC address and the IPv6 addresses [4]. The Node ID can be modified by the user.
To access the Node ID in contiki, we use the function `node_id` from the library `node-id.h`.

```
/*  Header file*/
#include "node-id.h"
/* Node ID reading */
 data_to_be_sent_Ptr->NodeID=node_id ;
```

Listing 3.2 – Code fragment for Node ID assignation

Temperature

Sensor We used the Z1 platform on-board digital temperature sensor **TMP102**.

Code The associated library in Contiki is located in `"plateform/z1/dev/tmp102.h"`. The main functions are listed in the table below.

`void tmp102_init(void);`	Init the sensor : ports, pins, registers,...
`void tmp102_write_reg(uint8_t reg,uint16_t val);`	Write to a register
`uint16_t tmp102_read_reg(uint8_t reg);`	Read the sensor register (temperature)
`uint16_t tmp102_read_temp_raw();`	Read the temperature in raw format
`int8_t tmp102_read_temp_simple();`	Read only the integer part of temperature
`int16_t tmp102_read_temp_x100();`	Read only the integer part of temperature multiplied by 100

Table 3.2 – tmp102.h main functions

To read the sensed data (temperature), we need to make the following steps :

1. Include the driver header.
2. Initialize the temperature sensor.
3. Request a reading.

```
/*  Header file*/
#include "dev/tmp102.h"
 /*  Init the temperature sensor*/
tmp102_init();
/*  Temperature reading*/
data_to_be_sent_Ptr-> Temperature=tmp102_read_reg(TMP102_TEMP);
```

Listing 3.3 – Code fragment for temperature reading

Humidity

Sensor We used a basic low-cost soil moisture sensor. This sensor has an analogue output that can be connected to Z1 module via the Phidget ports $3V$ or $5V$.

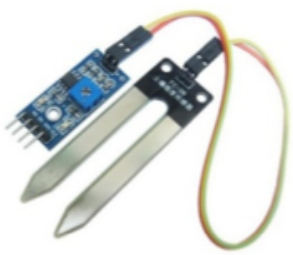


Figure 3.6 – Soil moisture sensor

The soil moisture sensor consists of two probe's pads which are used to measure the Volumetric content of water. The two probe's pads allow the current to pass through the soil and then it gets the resistance value to measure the moisture value [2].

- Wet soil : with higher volume of water, conductivity increases and the resistance will decrease. Therefore the output voltage will be less.
- Dry soil : poor conductivity and so the resistance increases and the output voltage will be high.

Since we have to power our sensing motes by batteries ($3.3V$), we connected the humidity sensor to the Phidget connector $3V$.

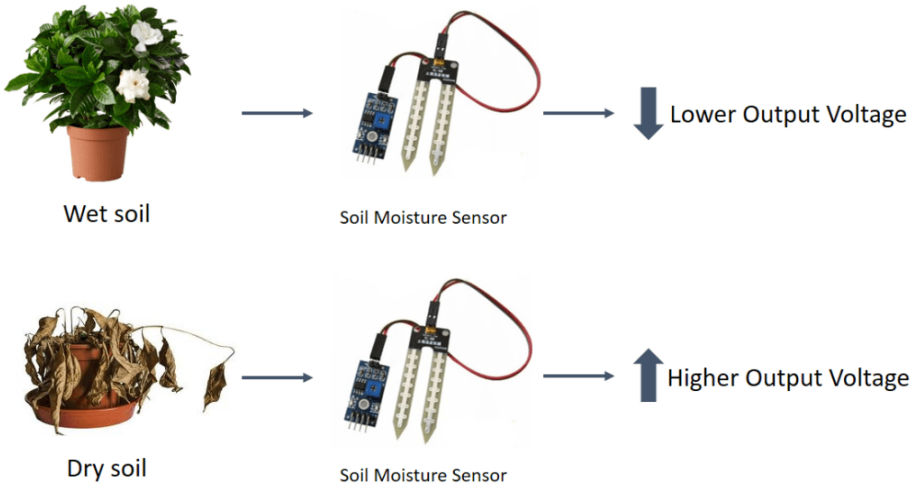


Figure 3.7 – Humidity and voltage

Humidity reading The measured soil conductivity and the humidity are inversely related. To get the correct humidity reading we note the extreme values of the measured data in two fields : dry and wet soil. We get the following :

Max Raw value (Dry soil)	4095
Min Raw Value (Wet soil)	1700

The measured values present the raw values of humidity which is the output of the ADC register (12 bits). Given the `max` and `min` values, we established the following formula to retrieve the humidity in percentage from a measured $\text{Humidity}_{\text{RAW Value}}$.

$$Humidity_{\texttt{Value}} = 100 - \frac{Humidity_{\texttt{RAW Value}} - Humidity_{\texttt{Min RAW Value}}}{Humidity_{\texttt{Max RAW Value}} - Humidity_{\texttt{Min RAW Value}}} \times 100$$

Listing 3.4 – Humidity calculation

Code To read the analog sensors, we use the Contiki function `phidgets.value(PHIDGETXV_X)` from the library in `z1-phidgets.h` .

PHIDGET5V_1	P6.0
PHIDGET5V_2	P6.3
PHIDGET3V_1	P6.1
PHIDGET3V_2	P6.7

To read the sensed data (humidity) from the $3V$ Phidget connector, we need to make the following steps :

1. Include the driver header
2. Enable the ADC sensors
3. Request a reading

```
/*  Include header file*/
#include "dev/z1-phidgets.h"
/*  Activate sensors*/
SENSORS_ACTIVATE(phidgets);
/* Request reading*/
data_to_be_sent_Ptr->Humidity=phidgets.value(PHIDGET3V_2);
```

Listing 3.5 – Code fragment for humidity reading

Luminosity

Sensor We used a P/N 1143_0 Phidget Light Sensor.

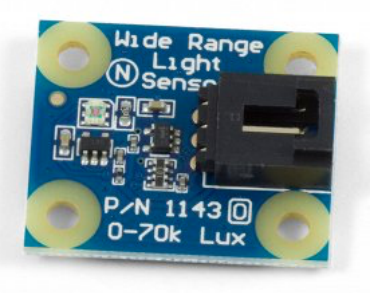


Figure 3.8 – Light sensor

The sensor specifications are listed below.

Light Level Min	1 lx
Light Level Max (3.3V)	46200lx
Light Level Max (5V)	70000 lx

Since we already used the Phidget connector port $3V$ with the humidity sensor and since the $5V$ connector needs $5V$ power from $\mu - USB$, we connected the light sensor power pins (VCC and GND) to the connector port $3V$ with the humidity power pins (VCC and GND) and the data pin (sensed value input) to the $5V$ Phidget connector. Hence the light sensor is powered by $3V$ and deliver measured value to $5V$ Phidget input.

Luminosity reading Knowing the maximum value of luminosity, we have :

$$\text{Luminosity} = \frac{\text{Luminosity(RAW Value)} \times 46200}{4095} \times \frac{5}{3} \tag{3.2}$$

where the 5/3 factor is due to the relationship of the Phidget$5V$ 3.1. The sensed value conversion is done in the receiver side.

Code To read the sensed data (luminosity) from the $5V$ Phidget connector, we need to make the following steps :

1. Include the driver header
2. Enable the ADC sensors
3. Request a reading

```
/*  Include header file*/
#include "dev/z1-phidgets.h"
/*  Activate sensors*/
SENSORS_ACTIVATE(phidgets);
/* Request reading*/
data_to_be_sent_Ptr->Luminosity=phidgets.value(PHIDGET5V_1);
```

Listing 3.6 – Code fragment for luminosity reading

3.3.2 Data Transfer and routing : networking

Mutihop with Rime

To implement Rime in our work, we used the platform program `example-multihop.c` in the directory `Contiki\examples\rime\`.

- Under the reception of a packet, the node forwards it to a random neighbor : call of function `forward()`.
- The forward function picks randomly a node from the neighbor list and sends the packet to.
- The list of neighbors nodes is created and updated by the mean of an announcement mechanism.
- The packet is forwarded from one node to another till it reaches the sink.

- To activate Rime, the Makefile of the application must contain the following line :

```
CONTIKI_WITH_RIME = 1
```

- We adapt the program `example-multihop.c` to our scenario and we added the C functions for sensing the three parameters (temperature, humidity and luminosity).
- The addressing mode is static and by default the sink node has the address 1.0. For the sensing motes, we used addresses : 2 and 3 $\longrightarrow$ addresses are burned in the nodes before load the program.
- In the receiver side, the sink retrieves the three parameters from the data packet in the same way as in the RPL case.

Mutihop with RPL

The RPL protocol comes with the use of the uIP stack. In contiki system and same as we did when using Rime stack, the Makefile must contain the following parameters:

```
WITH_UIP6=1
UIP_CONF_IPV6=1
CFLAGS+= -DUIP_CONF_IPV6_RPL
```

or we can simply replace those lines by :

```
CONTIKI_ITH_IPV6 = 1
```

To implement the network between motes and the sink node, an UDP connection was used. Two Contiki API are provided for an UDP communication : `Raw UDP API` and `Simpl-UDP API`. Table 3.3 lists the basic functions for the `Simpl-UDP API`.

The communication between the sender mote and the sink is based on a `Servreg-hack` algorithm in which the motes publish a service (temperature, humidity, luminosity) and the receiver (sink) must subscribe on the service to make its IP address known for sender motes. To establish a connection between a sender node and the sink, the following steps were made.

`simple_udp_send(struct simple_udp_connection*c, const void*data, unit16_t datalen)`	Send UDP packet
`simple_udp_sendto(struct simple_udp_connection*c, const void*data, unit16_t datalen, const uip_ ipaddr_t*to)`	Send UDP packet for IP address
`simple_udp_sendto_port(struct simple_udp_connection*c, const void*data, unit16_t datalen const uip_ipaddr_t*to, unit16_t port)`	Send UDP packet for an IP address and a UDP port
`simple_udp_register(struct simple_udp_connection*c, unit16_t local_port, uip_ipaddr_t*remote_addr, unit16_t remote_port, simple_udp callback receive callback)`	Register for UDP connection

Table 3.3 – Simple-UDP API functions

Sender side

1. Init the Servreg-hack application.

```
servreg_hack_init();
```

2. Register for an UDP connection.

```
simple_udp_register(&unicast_connection, UDP_PORT, NULL,
 UDP_PORT, receiver);
```

3. Search for subscribers to get IP' address.

```
addr = servreg_hack_lookup(SERVICE_ID);
```

4. Sense / get the : temperature, humidity and luminosity.

```
    data_to_be_sent_Ptr->NodeID=node_id ;
  data_to_be_sent_Ptr->Humidity=phidgets.value(PHIDGET3V_2);
  data_to_be_sent_Ptr->Luminosity=phidgets.value(PHIDGET5V_1);
  data_to_be_sent_Ptr->Temperature=tmp102_read_reg(TMP102_TEMP);
```

5. Send a data packet containing the three parameters to the subscriber (sink).

```
simple_udp_sendto(&unicast_connection,data_to_be_sent_Ptr, sizeof(
    data_to_be_sent), addr);
```

Receiver side

1. Init the Servreg-hack application, configure IP addressing and

```
  servreg_hack_init();
ipaddr = set_global_address();
create_rpl_dag(ipaddr);
```

2. Subscribe (IP address) to receive a service (3 parameters)

```
servreg_hack_register(SERVICE_ID, ipaddr);
```

3. Register for an UDP connection.

```
simple_udp_register(&unicast_connection, UDP_PORT, NULL, UDP_PORT
  , receiver);
```

4. Wait to receive.

3.3.3 Data reception : sink side

At the reception of a data packet, the sink which is connected to the gateway via USB port, retrieves the different fields separately and writes them to the serial port (USB).

Sensor ID

To sensor ID or the node identifier is retrieved first when a data packet arrives.

```
msgPtr->NodeID;
```

Temperature

To read the temperature value from the data packet, the sink makes the following steps:

1. read the temperature field (2 bytes) from the data packet.
2. retrieve the integer part from the temperature field.
3. retrieve the fraction part from the temperature field.
4. print the two values to the serial output (USB) with the data format : `integer.fraction`

```
/*---- --Temperature retieving---------------------- */
uint16_t temp_raw_value ;
int16_t  tempint;
uint16_t tempfrac;
uint16_t absraw;
int16_t  sign;
sign = 1;
temp_raw_value=msgPtr->Temperature ;
absraw = temp_raw_value;
if (temp_raw_value < 0) { // Perform 2C's if sensor returned negative
    data
absraw = (temp_raw_value ^ 0xFFFF) + 1;
sign = -1;
}
tempint  = (absraw >> 8) * sign;
tempfrac = ((absraw>>4) % 16) * 625; // Info in 1/10000 of degree
}
```

Listing 3.7 – Temperature reading at reception

Humidity

To retrieve the humidity from the data packet, the sink makes the following steps:

1. read the humidity field (2 bytes) from the data packet.
2. retrieve the RAW humidity from the humidity field.
3. use the formula 3.4 to determine the humidity value in percentage.
4. print the humidity value to the serial output (USB).

```
/*------ Humidity retrieving ------------------------ */
uint16_t humidity_raw_minvalue3V =1700; // depends on the power input
   (5V, 3,3V)
//uint16_t humidity_raw_maxvalue3V = 4095;
uint16_t humidity_relative_value_PHIDGET3V;
uint16_t humidity_raw_value_PHIDGET3V ;
uint16_t humidity_value_PHIDGET3V ;

humidity_raw_value_PHIDGET3V = msgPtr->Humidity ;
if (humidity_raw_value_PHIDGET3V < humidity_raw_minvalue3V) {
humidity_relative_value_PHIDGET3V= 0;
}
humidity_relative_value_PHIDGET3V = humidity_raw_value_PHIDGET3V -
   humidity_raw_minvalue3V;
humidity_relative_value_PHIDGET3V *= ((double)100 / (double)2395);
humidity_value_PHIDGET3V =  100 - humidity_relative_value_PHIDGET3V;
```

Listing 3.8 – Humidity reading at reception

Luminosity

To retrieve the luminosity from the data packet, the sink makes the following steps:

1. read the luminosity field (2 bytes) from the data packet.
2. retrieve the RAW luminosity from the luminosity field.
3. use the formula **??** to determine the correct luminosity value.
4. print the luminosity value to the serial output (USB).

```
/*---------------Luminosity retrieving------- -------*/
uint16_t luminosity_raw_value_PHIDGET35V;
uint16_t luminosity_value_PHIDGET35V;

luminosity_raw_value_PHIDGET35V = msgPtr->Luminosity ;
luminosity_raw_value_PHIDGET35V *=  ((double)5 / (double)3);
luminosity_value_PHIDGET35V =  (luminosity_raw_value_PHIDGET35V * ((
   double)46200 / (double)4095) );
```

Listing 3.9 – Luminosity reading at reception

The Gateway

4

Contents

The data received by the sink is forwarded to the Internet via the gateway. We used a Raspberry Pi 3 Model B board, connected to the sink via USB, to be our gateway (Figure 4.1).

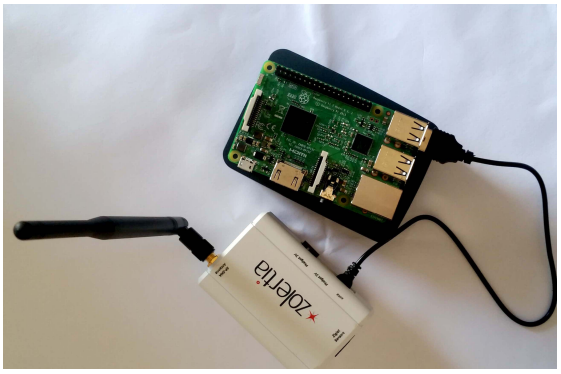


Figure 4.1 – The gateway + Sink

4.1 The Raspberry Pi

The raspberry Pi is a low cost small, cheap ARM-based PC that runs Debian GNU/Linux from an SD card [8]. The main features of the Raspberry Pi 3 Model B are listed in the Table 4.1

Feature	Specification
CPU	Quad-core 64-bit ARM Cortex A53 clocked at 1.2 GHz
GPU	400MHz VideoCore IV multimedia
Memory	1GB LPDDR2-900 SDRAM
Video outputs	HDMI, composite video (PAL and NTSC) via 3.5 mm jack
Audio	HDMI, stereo analog
USB	4
Storage	SD card
Networking	10/100Mbps Ethernet and 802.11n Wireless LAN
Power	5 V via MicroUSB or GPIO header

Table 4.1 – Raspberry Pi 3 Model B specifications

Three tasks are performed by the gateway :

1. Data serialization
2. Data reading and split
3. Database update

4.2 Data serialization

The sink forwards the received data to the serial port USB, connected to Raspberry Pi, by means of the C library function `printf()`. The sink has knowledge of the sent message structure. Each time a data packet is received, the sink points to each data field separately and forwards it to the serial port.
Each parameter is separated from the subsequent one by two points **(:)** and each data packet is separated from the subsequent by a carriage return **(/n)** , so the reception of a new packet is marked by a new line.

```
 printf("%d:",msgPtr->NodeID) ;
 printf("%d:", humidity_value_PHIDGET3V);
 printf("%d:",luminosity_value_PHIDGET35V);
 printf("%d.%04d\n", tempint, tempfrac);
printf("\n");
```

Listing 4.1 – Serial port data forward

4.3 Data reading and split

To retrieve data from the serial port, we read the Linux system file ttyUSB0 located at `/dev/ttyUSB0`. The received serial data is loaded and interpreted line by line (packet by packet). Based on the knowledge of each parameter order in the received data line (packet), we split the message into corresponding fields : NodeID, Humidity, Luminosity and Temperature.

```
ser = serial.Serial('/dev/ttyUSB0' , 115200)
while True :
data = ser.readline()
if data:
x = data.decode("utf-8").split(":")
NodeID= x[0]
Humidity =  x[1]
Luminosity =  x[2]
Temperature = x[3]
```

Listing 4.2 – Python code for data split

4.4 Database update

After reading and identifying each parameter in the received data, the gateway sends the data to a database hosted in a virtual machine in the cloud. The stored data is processed by fuzzy logic algorithm to determine two irrigation parameters : the flow rate and the delay. So to record and process data effectively in real time, two solutions were adapted:

4.4.1 Solution A

All parameters (temperature, humidity and luminosity, delay, flow rate) are stored in the same record (table) of a database. Four scripts are programmed to do the work. Processing actions are organized as follow :

1. The gateway sends the sensed parameters (temperature, humidity and luminosity) to the database record. The script in action is a Python script running in the gataway.

2. A Python script reads the record and retrieve the input parameters (temperature, humidity and luminosity) for exploration to determine the irrigation parameters.

3. A fuzzy logic algorithm (Matlab / R) explores the reading inputs and generates output parameters (delay, flow rate).

4. A Python script updates and writes the output parameters to the same record.

Database structure

We created a database `PISTE01A_2018` with one table `Agro_Environmental_Parameters` of 9 columns. The `Row_ID` field is incremented every data update and is necessary to track the last values when reading the current values and to plot the parameter evolution per time.

#	Name	Type	Collation	Attributes	Null	Default	Extra
1	**Row_ID**	int(11)			No	*None*	AUTO_INCREMENT
2	**Sensor_ID**	int(11)			Yes	*NULL*	
3	**Luminosity**	int(11)			Yes	*NULL*	
4	**Humidity**	int(11)			Yes	*NULL*	
5	**Temperature**	varchar(10)	latin1_swedish_ci		Yes	*NULL*	
6	**Delay**	double			Yes	*NULL*	
7	**Flow_Rate**	double			Yes	*NULL*	
8	**Time**	time			Yes	*NULL*	
9	**Date**	date			Yes	*NULL*	

Figure 4.2 – Solution A : `Agro_Environmental_Parameters` table structure

Database access

When updating the table, the Python script uses the following parameters for access.

User	`PISTE01`
Database	`PISTE01A_2018`
Table	`Agro_Environmental_Parameters`

When creating the user `PISTE01A_2018` , we should affect him the necessary privileges to access from everywhere (hostname) the database and insert data.

Code

Below the code for the database access.

```
import mysql.connector
USERNAME ="PISTE01"
PASSWORD = "xxx"
DBNAME = "PISTE01A_2018"
HOSTNAME = "xxx-xx-xx-xxx-x.xx-xxxx-x.compute.amazonaws.com"
con = mysql.connector.connect(host = HOSTNAME ,user= USERNAME, passwd =
    PASSWORD ,db = DBNAME, port = 3306)
con.autocommit = True
cursor = con.cursor()
print ("Connected to PISTE01A-2018 Database!")
cursor.execute("INSERT INTO Agro_Environmental_Parameters(Sensor_ID,
    Luminosity,Humidity,Temperature,Time,Date) VALUES(%s,%s,%s,%s,%s,%s
    )" ,(ID,LUM,HUM,TEMP,time.strftime(r"%H:%M:%S", time.localtime()),
    time.strftime(r"%Y-%m-%d", time.localtime())))
```

4.4.2 Solution B

Two database records are used but only two scripts to run.

1. The gateway sends the sensed parameters (temperature, humidity and luminosity) to the first database record `Sensor_Data`. The script in action is a Python script running in the gataway.

2. A R script reads the table and retrieve the input parameters (temperature, humidity and luminosity), explore them, returns the irrigation parameters (delay, flow rate) and finally copies inputs and outputs to a new database record named `Agro_Environmental_Parameters` .

Database structure

We created a database PISTE01B_2018 with two tables Agro_Environmental_Parameters and Sensor_Data. The Agro_Environmental_Parameters has the same structure as in solution A.

#	Name	Type	Collation	Attributes	Null	Default	Extra
1	**row_id**	int(11)			No	*None*	AUTO_INCREMENT
2	**sensor_id**	int(11)			Yes	*NULL*	
3	**lum**	int(11)			Yes	*NULL*	
4	**hum**	int(11)			Yes	*NULL*	
5	**temp**	varchar(11)	latin1_swedish_ci		Yes	*NULL*	
6	**time**	time			No	*None*	
7	**date**	date			No	*None*	

Figure 4.3 – Solution B : `Agro_Environmental_Parameters` Structure

#	Name	Type	Collation	Attributes	Null	Default	Extra
1	**Row_ID**	int(11)			No	*None*	
2	**Sensor_ID**	int(11)			Yes	*NULL*	
3	**Luminosity**	int(11)			Yes	*NULL*	
4	**Humidity**	int(11)			Yes	*NULL*	
5	**Temperature**	varchar(10)	latin1_swedish_ci		Yes	*NULL*	
6	**Delay**	double			Yes	*NULL*	
7	**Flow_Rate**	double			Yes	*NULL*	
8	**Time**	time			Yes	*NULL*	
9	**Date**	date			Yes	*NULL*	

Figure 4.4 – Solution B : `Sensor_Data` Structure

Database access

When updating the table, the Python script uses the following parameters for access.

User	`PISTE01`
Database	`PISTE01B_2018`
Table	`Sensor_Data`

Same as when creating use `PISTE01A_2018`, the user `PISTE01B_2018` , should have the necessary privileges to access and modify data from everywhere.

Code : Database connection and update

```
import mysql.connector
USERNAME ="PISTE01"
PASSWORD = "xxx"
DBNAME = "PISTE01B_2018"
HOSTNAME = "xxx-xx-xx-xxx-x.xx-xxxx-x.compute.amazonaws.com"
con = mysql.connector.connect(host = HOSTNAME ,user= USERNAME,passwd =
    PASSWORD ,db = DBNAME, port = 3306)
con.autocommit = True
cursor = con.cursor()
print ("Connected to PISTE01B-2018 Database!")
cursor.execute("INSERT INTO Sensor_Data(sensor_id,lum,hum,temp,time,
  date) VALUES(%s,%s,%s,%s,%s,%s)" ,(ID,LUM,HUM,TEMP,time.strftime(r"
  %H:%M:%S", time.localtime()),time.strftime(r"%Y-%m-%d", time.
  localtime())))
```

Data analysis

5

Contents

The fourth step of the application corresponds to exploit the WSN measurements. Therefore, we applied an analysis based on statistical processing on the collected WSN data in order to establish the watering quantities in a precise way thus allowing a good irrigation management.
More precisely, we implemented and validated the fuzzy algorithm. It is based on the fusion of temperature, light (solar radiation) and moisture of soil data to establish the volume of water needed for irrigation as well as the time and the required flow.

5.1 Fuzzy Logic Principle

The fuzzy logic controller (FLC) acquires data from these sensors and then applies well-devised fuzzy rules to produce appropriate time and duration for irrigation. The following block diagram presents the structure of FLC.

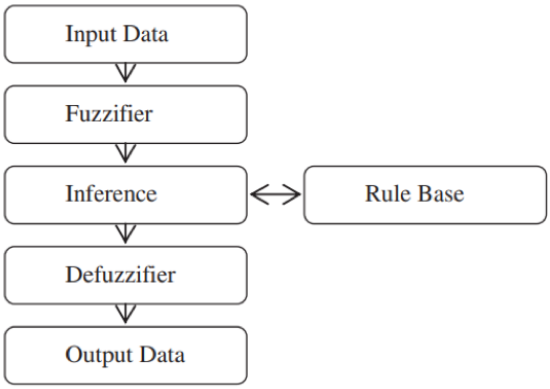


Figure 5.1 – Fuzzy Logic Algorithm Structure (Touati et al., 2013).

All variables are fuzzified using trapezoidal and triangular membership functions. The membership functions are distributed according to the possible values of each variable after fuzzification. The fuzzification process, Max-Min inference engine and Mamdani-type rule base produce the required decision for each situation. After application of a centroid defuzzification, the controller produces the desired output. Three inputs and one output are used in our work :

- Soil moisture: gives the water stress in soil. It is the most important information to know whether or not irrigation is needed and estimate the amount of water that should be dispensed to the crop.
- Outside temperature: measures ambient temperature. This input enables the FLC reducing water evaporation by avoiding irrigation at high ambient temperatures.
- Radiation: measures solar radiation in the field. Likewise the ambient temperature input, since solar radiation accelerates evapotranspiration, it helps the FLC reducing water loss through evapotranspiration.
- Duration: the FLC provides the time and duration of irrigation through the output port.

More details on the algorithm are available in the following references [9, 7].
In the first phase, we simulated data (three data variables) of the considered three parameters under Matlab. In the second phase, the data generated from part three has been scanned directly in real time and the algorithm has been adapted for intelligent irrigation.

5.2 Implementation

To provide the irrigation decision with the required parameters (duration and flow rate), we first develop a fuzzy logic algorithm in Matlab and apply it on the three input parameters. Since our work is planned to be run on the cloud, and since Matlab is'nt an easy resource to use on cloud based platform, we decide to redevelop our code with R language and run it under Rstudio which is easy to install and use on the cloud.
To explore the sensed data stored in the database and generate the desired outputs, we split our work on three phases:

1. Phase 1 : read the database to generate a test file that contains the measurements of the 3 physical variables (temperature, moisture of soil and luminosity).

(2) Phase 2 (R script) : application of the Fuzzy Gaussian membership function on the 3 variables + application of the Mamdani function

(3) Phase 3 : update the database with the obtained results from the R script.

In the following, we provide fragments from the developed code to implement the three phases.

5.2.1 Solution A

Phase 1 : Database reading (Python script)

```
import MySQLdb
USERNAME ="PISTE01"
PASSWORD = "xxx"
DBNAME = "PISTE01A_2018"
HOSTNAME = "xxx-xx-xx-xxx-x.xx-xxxx-x.compute.amazonaws.com"
file = open("input_data.txt","r+")
con = MySQLdb.connect(host = HOSTNAME ,user= USERNAME, passwd = PASSWORD
    ,db = DBNAME, port = 3306)
con.autocommit = True
cursor = con.cursor()
count =1
while(True):
cursor = con.cursor()
cursor.execute("SELECT Luminosity,Humidity,Temperature from Agro_
    Environmental_Parameters where Row_ID = %s", count)
row = cursor.fetchone()
```

Phase 2 : Fuzzy algorithm application (R script)

- **System definitions**

```
library(sets)
sets_options('universe', seq(from = 0,to = 4196, by = 1))
variables1 <-set(temperature =fuzzy_partition(varnames =c(cold = 0,
    medium = 25,hot = 50),sd = 12.5),
humidity =fuzzy_partition(varnames =c(wet = 0, medium = 37.5,dry = 75),
    sd = 18.75),
radiation =fuzzy_partition(varnames =c(low = 0, medium = 2098,intense =
    4196),sd = 1049),
volume =fuzzy_partition(varnames =c(small = 0, medium = 50,big = 100),
    sd = 12.5))
```

```
rules1 <-set(
fuzzy_rule(temperature %is% hot || humidity %is% wet || radiation %is%
    intense,
volume %is% big),
fuzzy_rule(temperature %is% cold || humidity %is% dry|| radiation %is%
    low,
volume %is% small))
system1 <- fuzzy_system(variables1, rules1)
variables2 <-set(temperature =fuzzy_partition(varnames =c(cold = 0,
    medium = 25,hot = 50),sd = 12.5),
humidity =fuzzy_partition(varnames =c(wet = 0, medium = 37.5,dry = 75),
    sd = 18.75),
radiation =fuzzy_partition(varnames =c(low = 0, medium = 2098,intense =
     4196),sd = 1049),
duration =fuzzy_partition(varnames =c(short = 0, medium = 30,long = 60)
    ,sd = 12.5))
rules2 <-set(
fuzzy_rule(temperature %is% hot || humidity %is% wet || radiation %is%
    intense,
duration %is% long),
fuzzy_rule(temperature %is% cold || humidity %is% dry|| radiation %is%
    low,
duration %is% short))
system2 <- fuzzy_system(variables2, rules2)
```

- **Database access and results generation : fuzzy algorithm application**

```
con = file("~/Desktop/Codes-PISTE01-2018/SolutionA/input_data.txt","r")
line_in = length(readLines(con))
v<-scan(file="~/Desktop/Codes-PISTE01-2018/SolutionA/input_data.txt",
    what=double(),nmax=3,skip= count )
count <- count +1;
f1 <- fuzzy_inference(system1, list( temperature=v[1], humidity=v[2],
    radiation=v[3]))
a <- gset_defuzzify(f1, "centroid")
f2 <- fuzzy_inference(system2, list( temperature=v[1], humidity=v[2],
    radiation=v[3]))
b <- gset_defuzzify(f2, "centroid")
line_out =paste(a,b,sep=" ")
write(line_out,file ="~/Desktop/Codes-PISTE01-2018/SolutionA/output_
    data.txt",append=TRUE)
cono = file("~/Desktop/Codes-PISTE01-2018/SolutionA/output_data.txt","r
    ")
```

Phase 3 : Database update (Python script)

```
import mysql.connector
import time
USERNAME ="PISTE01"
PASSWORD = "iot"
DBNAME = "PISTE01A_2018"
HOSTNAME = "xxx-xx-xx-xxx-x.xx-xxxx-x.compute.amazonaws.com"
 con = mysql.connector.connect(host = HOSTNAME ,user= USERNAME,passwd =
    PASSWORD ,db = DBNAME, port = 3306)
con.autocommit = True
cursor = con.cursor()
print ("Connected to PISTE01A-2018 Database!")
file_in = open("output_data.txt","r")
print "Name of the file: ", file_in.name
count =1
for line in file_in:
fields = line.split(" ")
a=float(fields[0])
b=float(fields[1])
cursor.execute("UPDATE Agro_Environmental_Parameters SET Delay='%f',
   Flow_Rate ='%f' WHERE ROW_ID='%d'"%(a,b, count))
count = count +1
```

5.2.2 Solution B

Phase 1 : Database reading (R script)

```
#----DataBase (Sensor_Data) variables retrieve -------#
mySqlCreds <- list(dbhostname ="localhost",
dbname ='PISTE01_2018',
username='PISTE01',
pass = 'iot',
port = 3306)
drv <- dbDriver("MySQL")
con<- dbConnect(drv,
host = mySqlCreds$dbhostname,
dbname = mySqlCreds$dbname,
user =mySqlCreds$username,
password = mySqlCreds$pass,
port=mySqlCreds$port)
RowQuery <- "SELECT COUNT(*) FROM Sensor_Data "
rowscount <- dbGetQuery(con, RowQuery)
myQuery <- paste("SELECT * from Sensor_Data where Row_ID = '",count," ';
   ",sep="")
```

```
data.frame <- dbGetQuery(con, myQuery)
v1 <-  as.double(data.frame$temp)
v2 <- as.double(data.frame$hum)
v3 <- as.double(data.frame$lum)
```

Phase 2 : Fuzzy algorithm application (R script)

The system definitions are the same as in code from solution A.

- **Database access and results generation : fuzzy algorithm application**

```
 f1 <- fuzzy_inference(system1, list( temperature= v1, humidity= v2 ,
   radiation=  v3  ))
volume <- gset_defuzzify(f1, "centroid")
f2 <- fuzzy_inference(system2, list( temperature=  v1, humidity= v2,
   radiation=  v3  ))
delay <- gset_defuzzify(f2, "centroid")
flowrate  <- volume / delay
```

Phase 3 : Database update (R script)

```
  #----DataBase (Agro_Environmental_Parameters) Writing -------#
values <- data.frame(
Row_ID= data.frame$row_id ,
Sensor_ID= data.frame$sensor_id ,
Temperature = data.frame$temp    ,
Luminosity = data.frame$lum ,
Humidity = data.frame$hum ,
Time = data.frame$time ,
Date =  data.frame$date ,
Flow_Rate = flowrate ,
Delay= delay
)
dbWriteTable(con,"Agro_Environmental_Parameters",values, overwrite=
   FALSE,append=TRUE, row.names = FALSE)
```

Remark : the three phases were developed and written in the same R script.

Application development

6

Contents

The main purpose of our work is to provide a web interface to supervise remotely and in real time the measured agro-environmental parameters. We propose to develop a simple web site interface (user dashboard) allowing us to view and explore measures. To allow on line access to the user dashboard, we choose to work with the cloud platform of Amazon **AWS** (Amazon Web Services). With AWS, two alternatives were discussed :

- **AWS IoT Service** : devices or sensors are directly connected to a cloud server and publishing measures periodically. The cloud IoT service provide many functionalities to do the data visualization and processing.
- **AWS EC2 Service** : deploy a virtual server in the cloud. The integrity of our work will be hosted and running over the virtual server. The remote access to our interface will be via the DNS / IP address of the server.

In our project, we considered the two alternatives; only the second alternative (EC2) is developed in this chapter.

6.1 AWS EC2

One of the most reputable hosting services is Amazon Web Services (AWS). AWS offers many services to store data, to connect IoT devices and to deploy application in the cloud. Among AWS services, we choose to deploy Amazon's Elastic Compute Cloud (EC2) which allow building virtual servers (instances) in the Amazon cloud. Amazon EC2 supplies a large choice of near instant access to on-demand re-sizable resources which allows to scale capacity as computing requirements change.

The billing system in EC2 allows user to pay, per hour, only for the capacity in use. The development and hosting of our application was made of the following steps :

1. Virtual server deployment : launch EC2 instance.
2. Virtual server preparation : software installation.
   - Installation and setting up of LAMP server.
   - Installation and setting up of R and RStudio.
3. User interface development.
   - Web development.
   - Node-RED deployment.

6.1.1 Virtual server deployment : EC2 instance

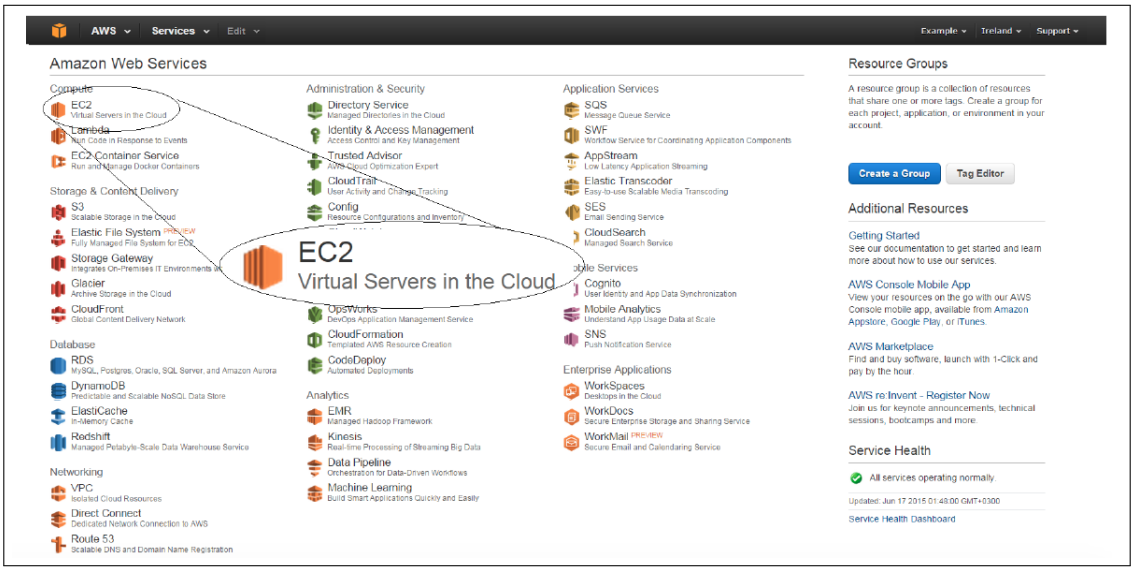


Figure 6.1 – EC2 Service

Launch EC2 instance

The deployment of EC2 instance is accessible via the EC2 dashboard. To launch an instance, the following steps were made.

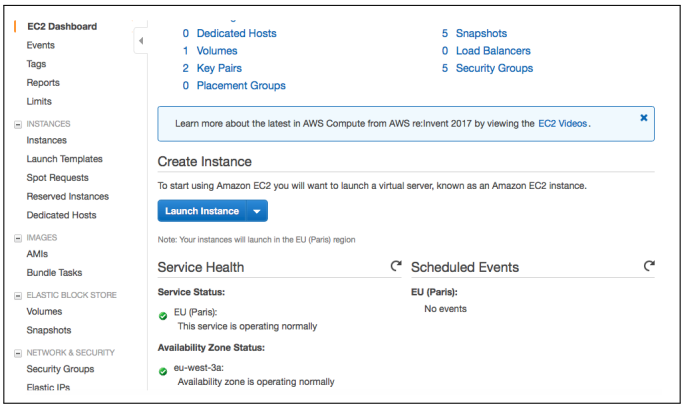


Figure 6.2 – Launch EC2 instance

(1) Choose AMI : to create an EC2 instance, we first select the system image (OS selection).

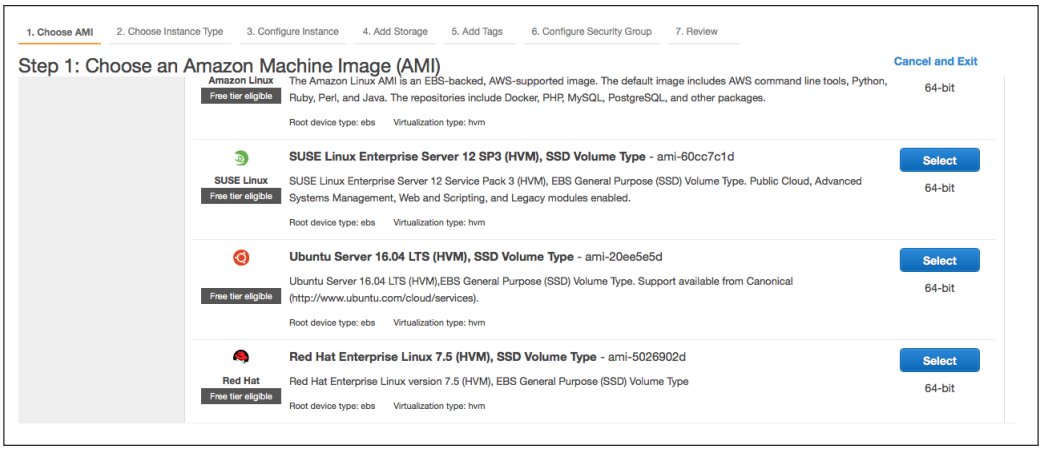


Figure 6.3 – AMI selection

(2) Choose instance type.

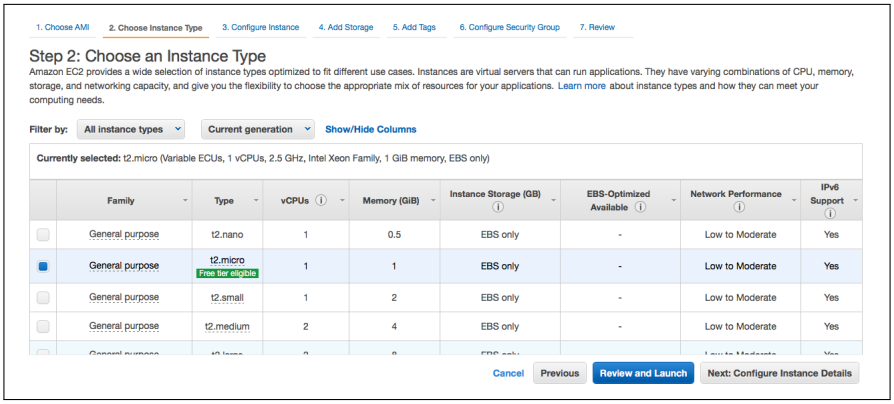


Figure 6.4 – Instance type

(3) Configure security group : a security group acts as a virtual firewall that controls inbound and outbound traffics from the instance. When we launch our VM, we should configure the security group by adding the desired rules.

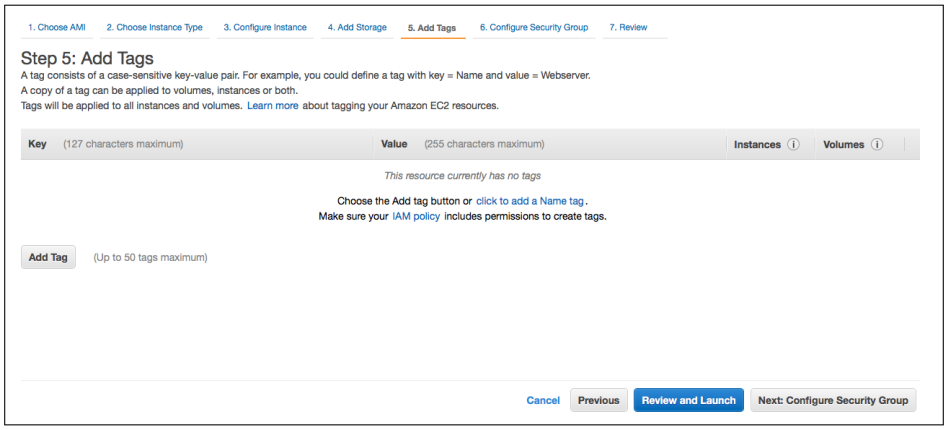


Figure 6.5 – Security group configuration

For our application, we must allow connections via the following ports:

- port 22 (SHH rule) : for SHH remote access (install and configure packages).
- port 80 (HTTP rule) : to allow access from browser(user interface).
- port 3306: to accept remote connections to the SQL server (data bases).
- port 8787 : for RStudio web access (R-based analysis environment to run developed scripts).
- port 1880 : for Node-RED web access (user dashboard development).

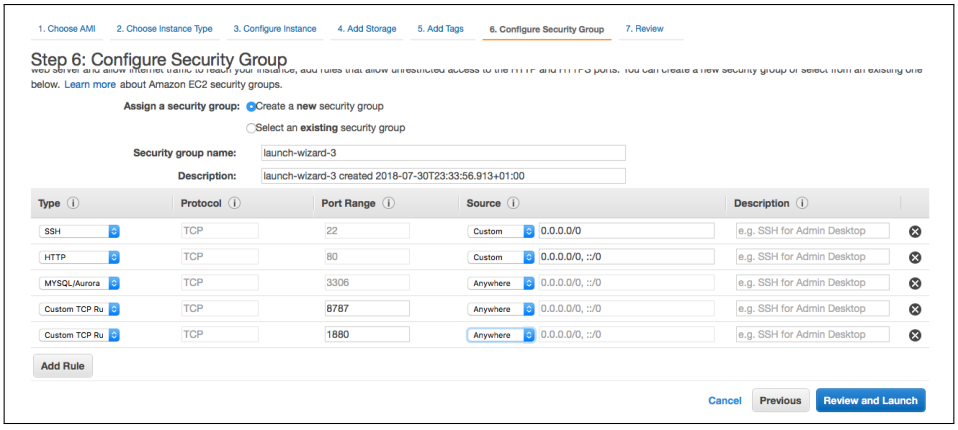


Figure 6.6 – Security rules

4 Download the private key : to allow secure connection, Amazon imposes a pair key access : private key stored at the user machine side and a public key that AWS stores. The private key is created and downloaded to the user machine before launching instance.

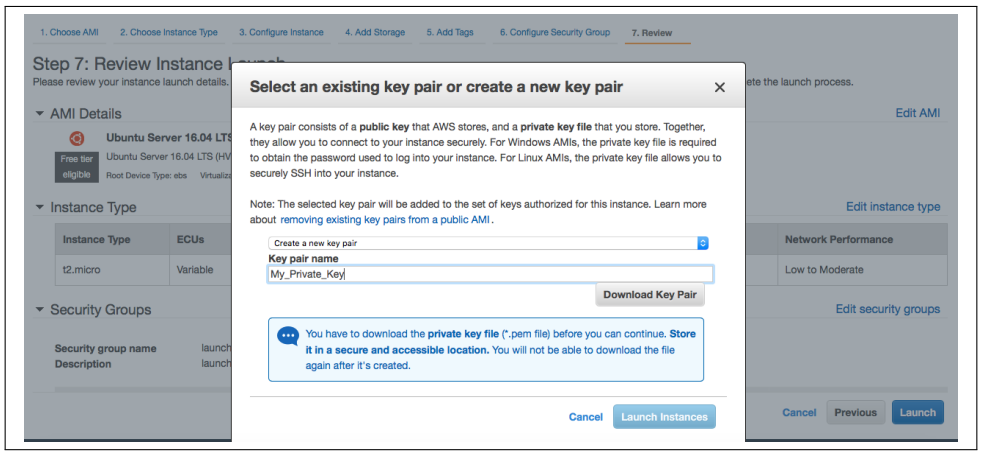


Figure 6.7 – Private key download

Once configured, the EC2 instance will be running. Remote access parameters are given in the instance description tab in the EC2 instance dashboard.

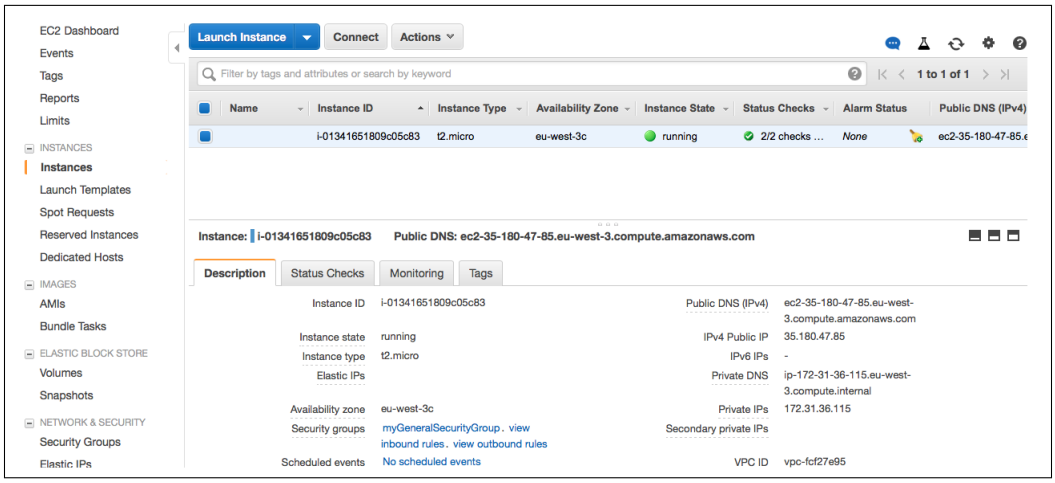


Figure 6.8 – EC2 instance dashboard

To remotely access our EC2 machine, and install required softwares, we should connect with an SHH client.

- To connect using the public ip of the EC2 instance :

```
ssh -i path_to_your_key/key.pem ubuntu@public_ip
```

Exemple:

```
ssh -i  /home/.aws/start-key.pem ubuntu@35.180.47.85
```

- To connect using the DNS of the EC2 instance :

```
ssh -i path_to_your_key/key.pem ubuntu@DNS
```

Exemple:

```
ssh -i  /home/.aws/start-key.pem ubuntu@ec2-35-180-47-85.eu-west
    -3.compute.amazonaws.com
```

6.1.2 Software installation

Setting up LAMP

To work with databases and web development, we choose to use the Linux Apache MySQL PHP (LAMP) server.

1. Install LAMP

```
sudo apt-get update
sudo apt-get install lamp-server^
```

2. Install PHPMyAdmin

To make database management easy over web interface, we use PHPMyAdmin. PHPMyAdmin is a free software tool written in PHP, intended to handle the administration of MySQL over the Web. To install it, we tape the following command :

```
sudo apt-get install phpmyadmin
```

The last command would install PHPMyAdmin in /usr/share/phpmyadmin. To make it available in the web root, we create the following symbolic link :

```
sudo ln -s /usr/share/phpmyadmin /var/www/html/phpmyadmin
restart apache2
```

Once installed, PHPMyAdmin should be accessible via the browser with the address : `public_ip/phpmyadmin`

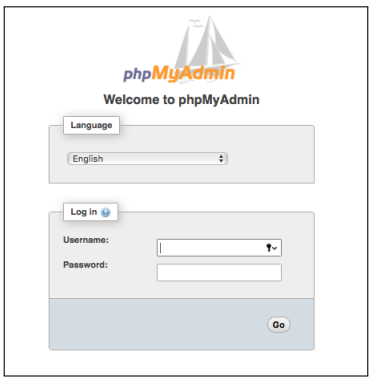


Figure 6.9 – PHPMyAdmin interface

Setting up R and RStudio

To install and configure R and RStudio, the following steps were made :

1. RStudio requires password authenticated user. So we added user `piste` and make it a super user (sudoer).

```
sudo adduser piste
sudo adduser piste sudo
```

2. Install R

```
sudo apt-get install r-base-core
```

3. Download package lists from repositories.

```
sudo apt-get update
```

4. Upgrade packages according to the updated information.

```
sudo apt-get upgrade
```

5. Install last version of R.

```
sudo apt-get install r-base-core
```

6 Install RStudio.

```
sudo apt-get install gdebi-core
wget http://download2.rstudio.org/rstudio-server-0.99.446-amd64.
    deb
sudo gdebi rstudio-server-0.99.446-amd64.deb
```

7 To access RStudio, in the web browser we open the address : `DNS:8787` and use the username and password we created (`piste`).

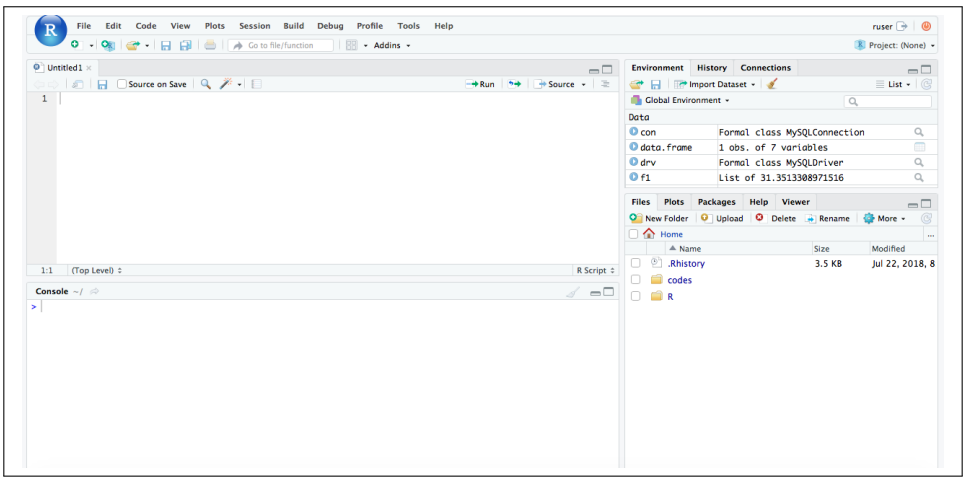


Figure 6.10 – RStudio web interface

6.2 Application development

To develop the user dashboard, two solutions were adapted :

- Web development.
- Node-RED deployment.

Note : to develop the user interface, we use only two sensors (sensor 1, sensor 2) with two predefined `Node_ID : (2,3)`.

6.2.1 Web development

We use web development tools like : html, PHP, CSS. To be accessible via the browser remotely, the project folder should be placed in the directory :`/var/www/html` of the virtual server. In the following screen shots from the developed user dashboard.

Figure 6.11 – Web interface : Home

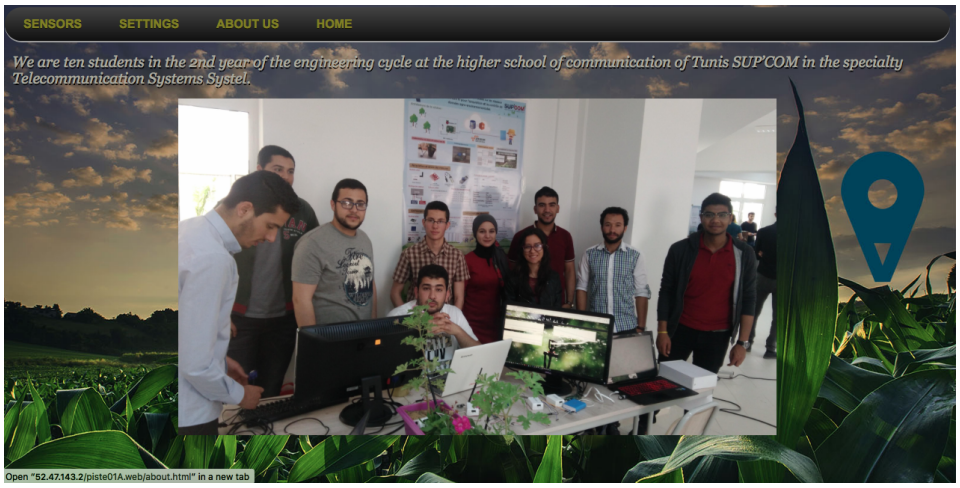


Figure 6.12 – Web interface : About us

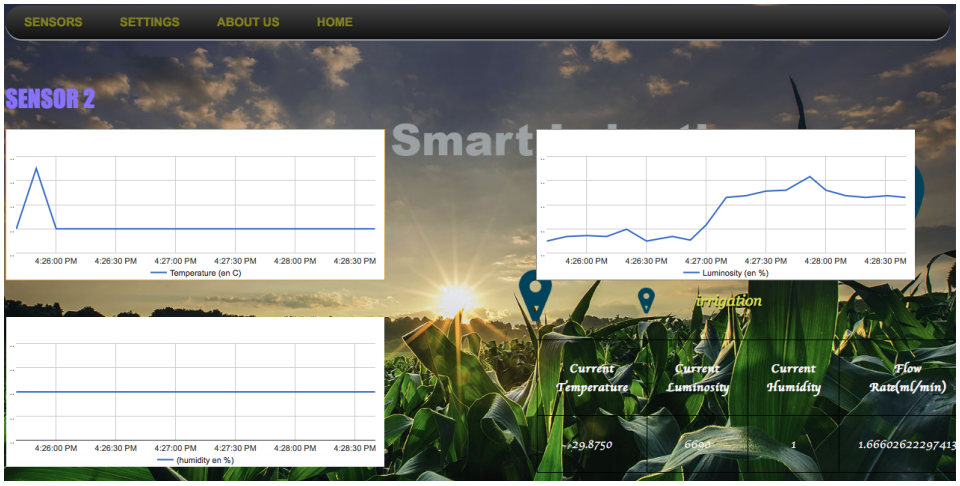


Figure 6.13 – Web interface : Sensor 1

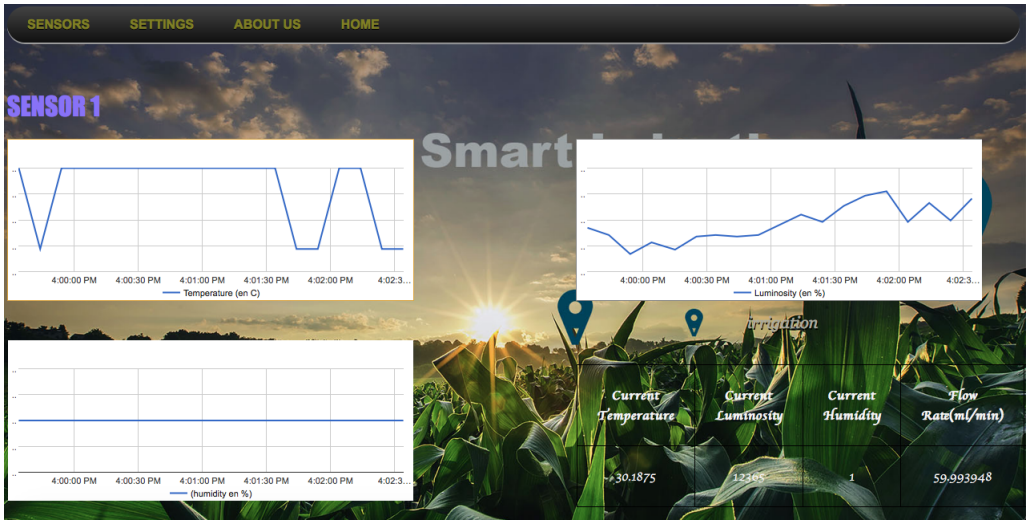
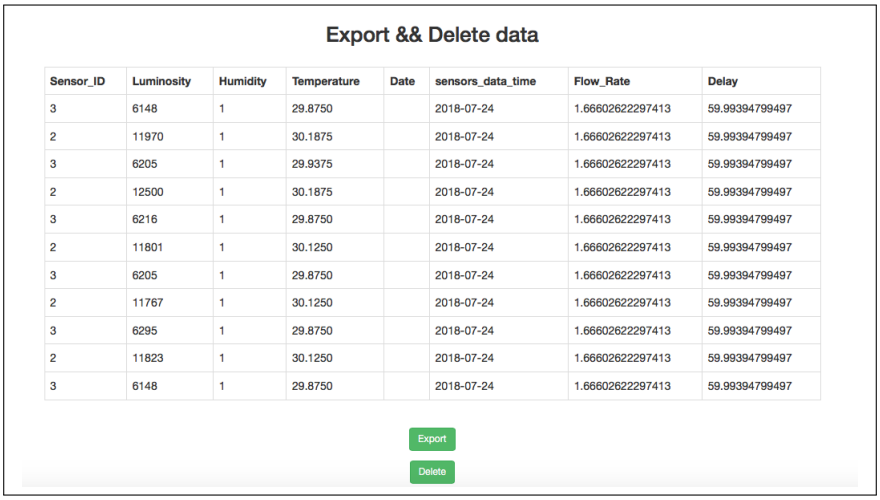


Figure 6.14 – Web interface : Sensor 2



Export && Delete data

Sensor_ID	Luminosity	Humidity	Temperature	Date	sensors_data_time	Flow_Rate	Delay
3	6148	1	29.8750		2018-07-24	1.66602622297413	59.99394799497
2	11970	1	30.1875		2018-07-24	1.66602622297413	59.99394799497
3	6205	1	29.9375		2018-07-24	1.66602622297413	59.99394799497
2	12500	1	30.1875		2018-07-24	1.66602622297413	59.99394799497
3	6216	1	29.8750		2018-07-24	1.66602622297413	59.99394799497
2	11801	1	30.1250		2018-07-24	1.66602622297413	59.99394799497
3	6205	1	29.8750		2018-07-24	1.66602622297413	59.99394799497
2	11767	1	30.1250		2018-07-24	1.66602622297413	59.99394799497
3	6295	1	29.8750		2018-07-24	1.66602622297413	59.99394799497
2	11823	1	30.1250		2018-07-24	1.66602622297413	59.99394799497
3	6148	1	29.8750		2018-07-24	1.66602622297413	59.99394799497

Export

Delete

Figure 6.15 – Web interface : Export measures

6.2.2 Node-RED

Another alternative way to create a user dashboard is to deploy Node-RED which is a flow based programming tool for wiring together hardware devices, APIs and online services. It provides a browser-based editor that makes it easy to wire together flows using the wide range of nodes in the palette that can be deployed to its runtime in a single-click [6].
Node-RED's dashboard conception is based on nodes. Each node has a well-defined purpose and manipulate data in some manner. The node behavior is defined and edited by the user.

installation

1. Configure Security Group tab and add a new 'Custom TCP Rule' for port 1880.
2. Log into the instance and install node.js and Node-RED

```
curl -sL https://deb.nodesource.com/setup_4.x | sudo -E bash -
sudo apt-get install -y nodejs build-essential
sudo npm install -g node-red
```

3. Run Node-RED in the terminal by executing the command : `node-red`. At this level, Node-RED is accessible via web browser at address : `http://<instance ip>:1880/`.

User Interface

To develop the user interface, we need :

- A Node-RED node to read and write to a MySQL database.
- Two Node-RED nodes to make two SQL queries, one for sensor 1 data and one for sensor 2 data.
- Three Node-RED nodes for each sensor to extract the corresponding data for each parameter among Temperature, Luminosity and Humidity.

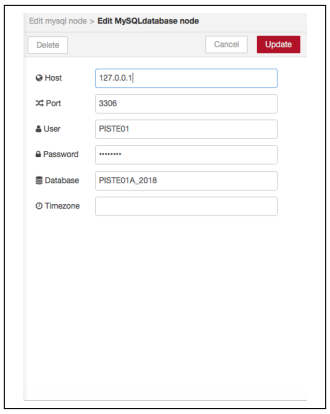


Figure 6.16 – Node-RED MySQL node

```
"SELECT Luminosity, Humidity, Temperature FROM `
  Agro_Environmental_Parameters` WHERE `Sensor_ID` = 3 ORDER BY
  ROW_ID DESC LIMIT 1";
```

Listing 6.1 – SQL Query Sensor 2

```
"SELECT Luminosity, Humidity, Temperature FROM `
  Agro_Environmental_Parameters` WHERE `Sensor_ID` = 2 ORDER BY
  ROW_ID DESC";
```

Listing 6.2 – SQL Query Sensor 1

The developed Node-RED Dashboard has the following flow.

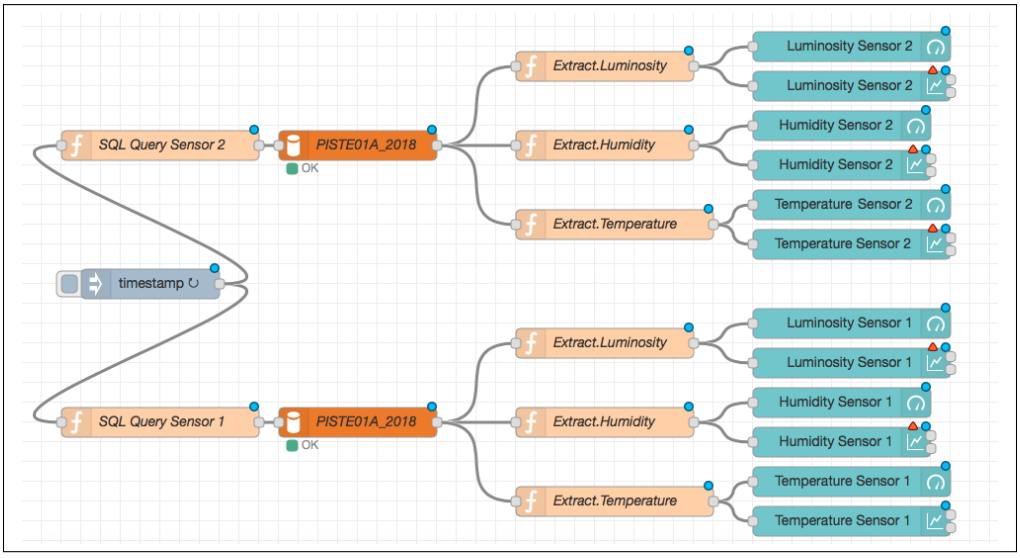


Figure 6.17 – Node-RED Flow

Figures 6.18-6.19 show the sensor's dashboards for real time monitoring.

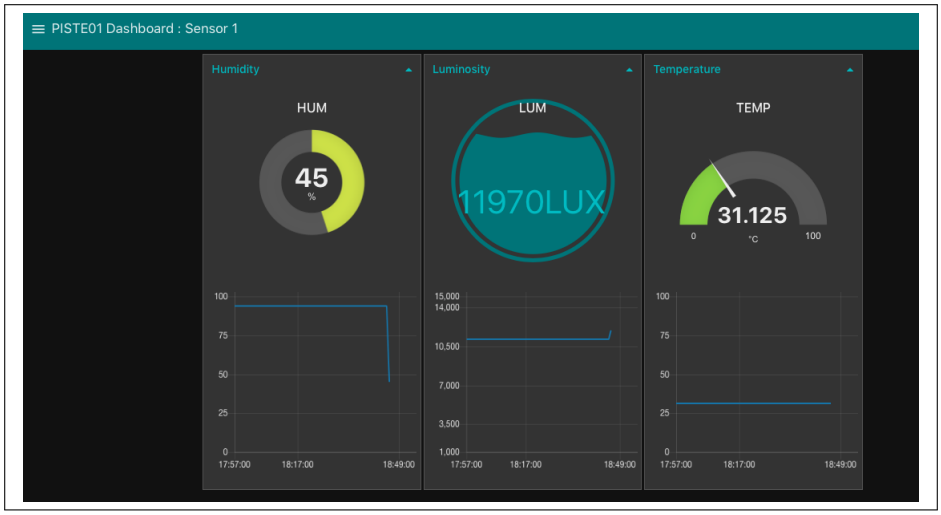


Figure 6.18 – User interface : sensor 1

Figure 6.19 – User interface : sensor 2

Conclusions and Future work

7

This project was a great opportunity to initiate and get skills in the field of IoT. We developed a real time application from the bottom line of sensing and forwarding data to the data processing in the cloud. The work presented in this report was evaluated and tested in a real Contiki 6LoWPAN network using Zolertia Z1 modules. We were able to process the measurement of three parameters (1 internal + 2 externals): temperature, soil humidity and luminosity using two different protocol stacks in Contiki. We proposed and validated two technical solutions for the data processing in the cloud based on the use of Python and R languages.
This project allowed us to become familiar with cloud virtualization and more precisely with AWS platform. We get basic skills to handle and manage an EC2 instance.

Due to the limitation in the number of external connectors in the Z1 module, only two possible connections, we could not use actuators to control the irrigation process. Deploying Z1 module in our application was made regarding the interoperability with the Contiki 6LoWPAN protocol stack and precisely the RPL routing protocol. Using a powerful wireless platform with 6LoWPAN compatibility and with more physical connectors to supervise, plus the irrigation, the measurement and provisioning of the soil nutrient will be a great value to the project.

Bibliography

[1] A. Bahga and V. Madisetti. *Internet of Things: A Hands-On Approach*. Vijay Madisetti, 2014.

[2] *Basics: Project 036a Soil Moisture Sensor YL-69, FC-28 or HL-69*. http://acoptex.com/project/179/basics-project-036a-soil-moisture-sensor-yl-69-fc-28-or-hl-69-at-acoptexcom.

[3] Antonio Linan Colina et al. *Contiki applications for Z1 motes for 6LowPAN*. http://hdl.handle.net/2117/82767. Master.

[4] Antonio Linan Colina et al. *IoT in five Days*. http://wireless.ictp.it/school_2016/book/IoT_in_five_days-v1.0.pdf , Rev 1.1. E-Book, 2016.

[5] *Contiki OS Wiki*. https://github.com/contiki-os/contiki/wiki.

[6] Node-Red official page. *Node-RED Flow-based programming for the Internet of Things*. https://nodered.org.

[7] Farid Touati et al. "A fuzzy logic based irrigation system enhanced with wireless data logging applied to the state of Qatar". In: *Computers and Electronics in Agriculture* 98.Supplement C (2013), pp. 233 –241.

[8] Eben Upton. *Programming the Raspberry Pi*. Raspberry Pi Foundation.

[9] Gabriel Villarrubia et al. "Combining Multi-Agent Systems and Wireless Sensor Networks for Monitoring Crop Irrigation". In: *Sensors* 17.8 (2017).

[10] Zolertia. *Z1 Datasheet*. http://zolertia.sourceforge.net/wiki/images/e/e8/Z1_RevC_Datasheet.pdf. 2010.