



UNIVERSITÀ DI PISA

Computer Engineering

Internet Of Things

Smart Agriculture

Project Report

Gianluca Gemini

Academic Year: 2021/2022

GitHub Repo: https://github.com/yolly98/smart_agriculture_IoT_project

Contents

1	Introduction	2
1.1	Technologies used	2
1.1.1	Software	2
1.1.2	Hardware	2
2	The project idea	3
3	Project Design	5
3.1	Use Case	5
3.2	UML diagram	6
3.3	The Master State Machine	7
3.4	The Node State Machine	8
3.5	The Database Design	9
3.5.1	The Land Table	9
3.5.2	The Configuration Table	10
3.5.3	The Measurement Table	11
3.5.4	The Violation Table	11
3.5.5	The Irrigation Table	12
4	Implementation	13
4.1	The Border Router	13
4.2	The Node	13
4.3	The COAP Node	14
4.4	The MQTT Node	15
4.5	The Master Node	16
5	Tests	20
5.1	Cooja Test	20
5.2	Test With Real nodes	21
6	Grafana	24

1 Introduction

The project is about **smart agriculture** and is a system that allows people to monitor the status of their cultivated areas by automatically managing irrigation.

The network consists of a border router, an undefined number of scattered **nodes** throughout the lands and a **central server** (the Master) that coordinates and manages all nodes. From the project specifications half of the nodes must use the **COAP** protocol and half the **MQTT** protocol, so the Master is able to communicate using both protocols.

1.1 Technologies used

The technologies used are as follows and their usage will be explained in the next chapters

1.1.1 Software

- Ubuntu 18.04 LTS
- Contiki-NG v4.7 on Docker (<https://github.com/contiki-ng/contiki-ng>)
- C-language
- Python-language
- Docker
- MySQL database on Docker
- Coapthon (COAP python library)
- Paho (MQTT python library)
- PySerial (serial python library)
- mysql python library
- Grafana on Docker

1.1.2 Hardware

- General purpose machine
- CC2650 Launchpad
- nRF52840 dongle board

2 The project idea

The project is designed to fit an undefined number of **lands** and an **undefined extension** for each one, for this reason any area can be monitored by one or more nodes of the network. Each node must be identified with **two identifiers**, one relative to the land (land id) and one related to nodes on the same land (node id).

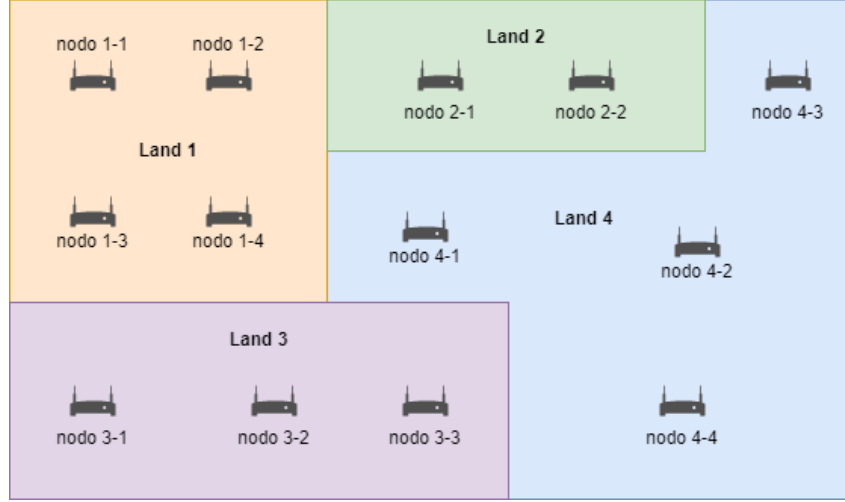


Figure 1: An example of lands and nodes

Each land differs from the other for the following parameters:

- *id*
- *area* (in acres)
- *locality* (in legal name)
- *name* (common name)
- *crop*
- *soil type* (Sand, Loam, Silt, Clay, Loamy Sand ...)

Within every land are nodes that measures the following values:

- *soil moisture* (in percentage)
- *ph level* (values from 0 to 14)
- *light ray level* (in DLI - Daily Light Integral)
- *soil temperature* (in Celsius)

In addition, a land, depending on its characteristics, has **limits** for each type of value. When a measure exceeds its limit, a **violation** is generated so that the problem can be resolved before the crop is damaged.

For the **light ray** are no bounds because every day, depending on the hour, the measurements **vary drastically**.

In case of **soil moisture** violation, the system acts autonomously by starting an **irrigation** operation in the area.

Practical example of a land:

- *id*: 1
- *area*: 0.1
- *locality*: pag 5, part 4, Carbognano (VT)
- *name*: Filaro
- *crop*: tomatoes
- *soil*: loam
- *moisture threshold*: 14
- *minimum ph level*: 6
- *maximum ph level*: 7
- *minimum soil temperature*: 4
- *maximum soil temperature*: 22

A node corresponds to a **configuration** that determines its **behavior**. If the node is new in the network, it is assigned a **default configuration** relative to the territory in which the node is located. Each configuration can be changed upon request by the master.

A configuration must have the following parameters:

- *land id*
- *node id*
- *protocol used* (COAP or MQTT)
- *address* (if is a COAP node)
- *status* (online/offline)
- *irrigation enabler* (it's possible disable irrigation for a node)
- *irrigation limit* (over this value the node starts an irrigation)
- *irrigation duration* (time after that the irrigation is stopped)
- a *timer* for each measure

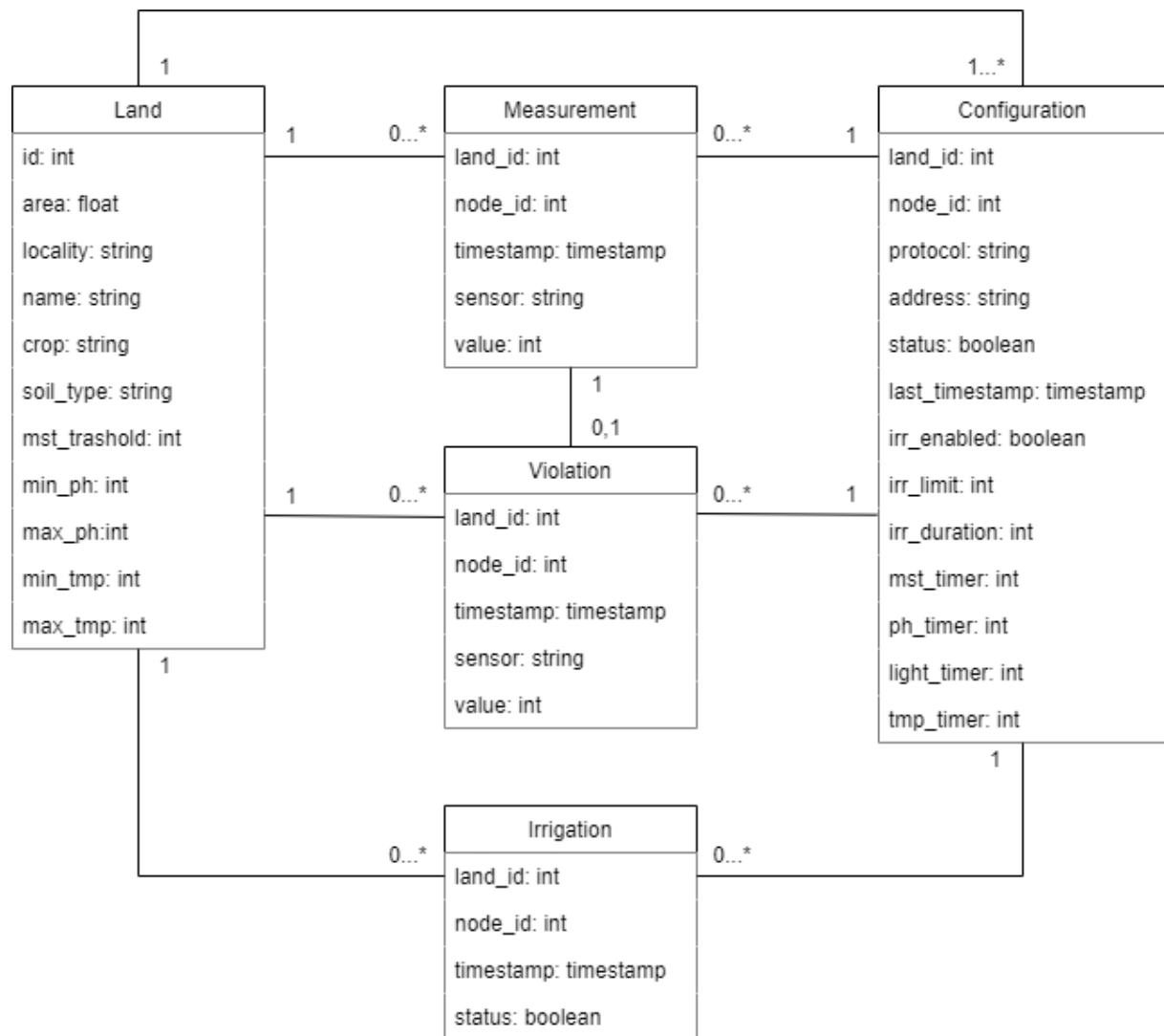
3 Project Design

3.1 Use Case

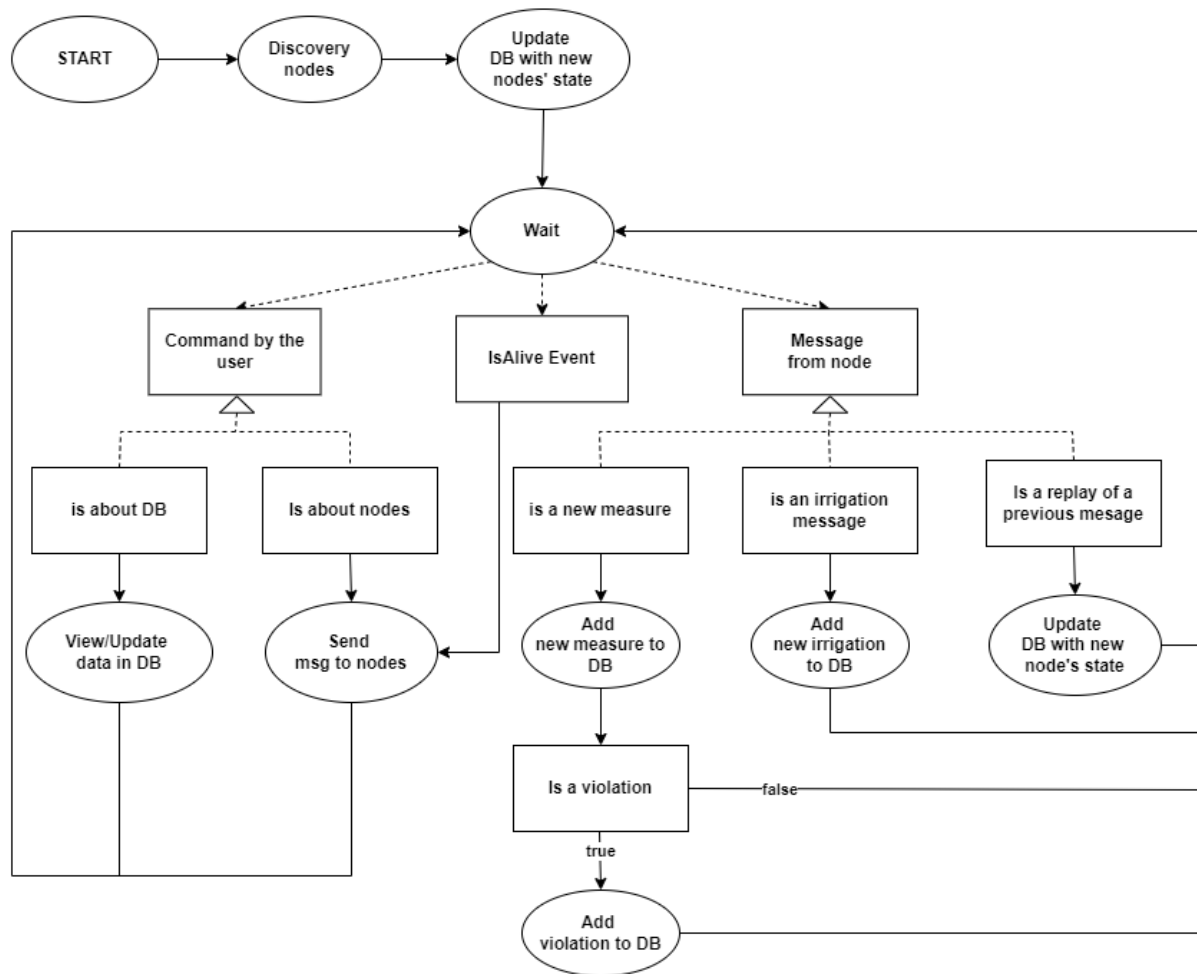
The **user** can interact with the **terminal interface** of the central server (master) to check the network status, modify data on the database and send messages to nodes. These are all operations that it's possible to do.

- Type 'help' to see all commands
- Type 'exit' to close the master
- See all lands data and configurations data
- See all measurements, violations and irrigations made in the past
- Change configuration of a node
- Ask a node its configuration
- Ask a node to do a measurement
- Check if a node is online
- Add new land, configuration, measurement, irrigation
- Update an existing land and configuration
- Set online a node
- Set offline all nodes
- Delete a land, configuration, measurement, violation and irrigation
- Do a test about the message reception
- See all registered nodes at that moment

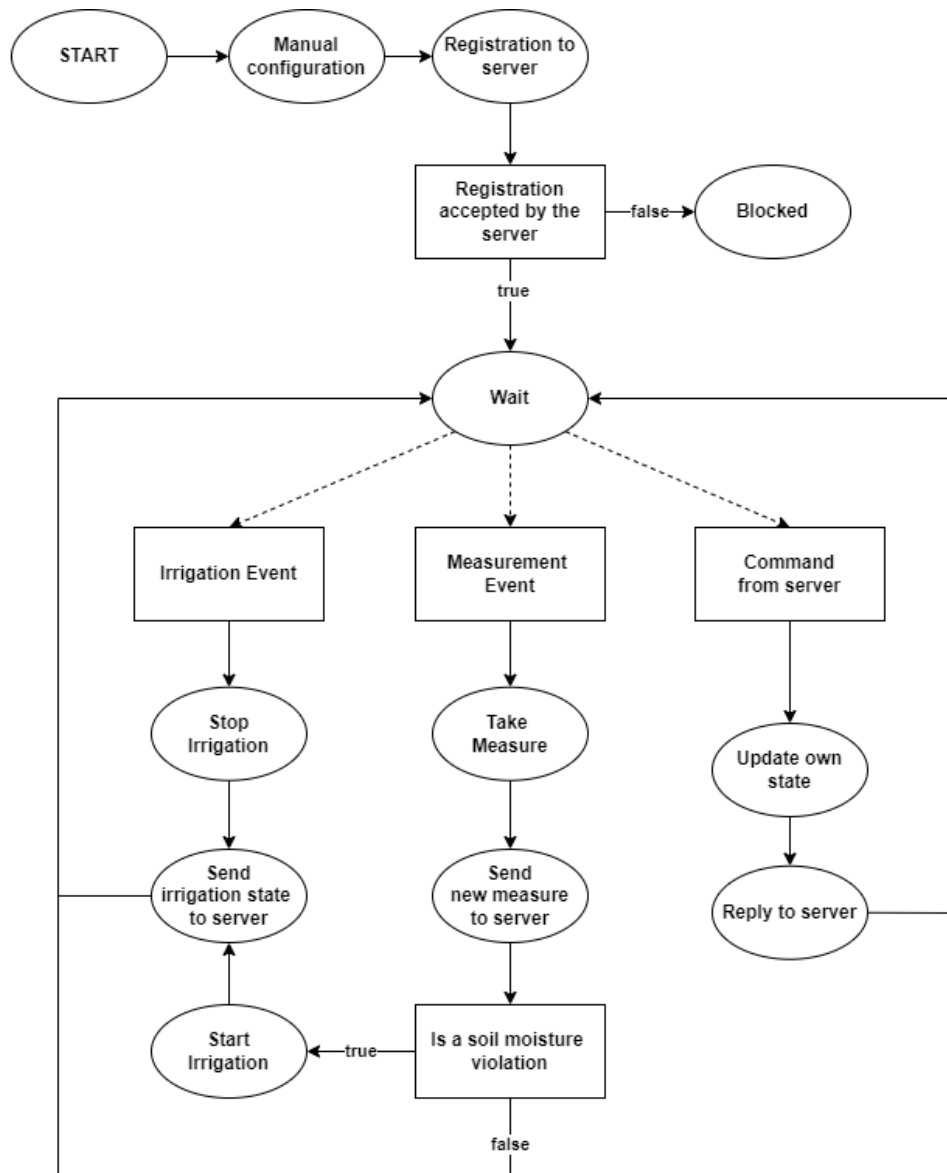
3.2 UML diagram



3.3 The Master State Machine



3.4 The Node State Machine



3.5 The Database Design

The following are the **sql** scripts used to build the **database**. The parameters of the tables are in accordance with those explained in the "Project Idea" section.

3.5.1 The Land Table

```
CREATE TABLE IF NOT EXISTS land (  
    id int(11) NOT NULL,  
    area float NOT NULL,  
    locality varchar(100) NOT NULL,  
    name varchar(100) NOT NULL,  
    crop varchar(100) NOT NULL,  
    soil_type varchar(100) NOT NULL,  
    mst_trashold int(11) NOT NULL,  
    min_ph int(11) NOT NULL,  
    max_ph int(11) NOT NULL,  
    min_tmp int(11) NOT NULL,  
    max_tmp int(11) NOT NULL,  
    PRIMARY KEY(id)  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

- *id*: land identifier
- *area*: land extension in acres
- *locality*: legal name by which the land is identified in the maps
- *name*: common name of the land
- *crop*: cultivation of the land
- *soil_type*: soil type of the land according to the standard classification
- *mst_trashold*: soil moisture below which irrigation should be initiated
- *min_ph* e *max_ph*: interval in which the ph level should be
- *min_tmp* e *max_tmp*: interval in which the soil temperature should be

3.5.2 The Configuration Table

```
CREATE TABLE IF NOT EXISTS configuration (  
  land_id int(11) NOT NULL,  
  node_id int(11) NOT NULL,  
  protocol varchar(100) DEFAULT NULL,  
  address varchar(100) NOT NULL,  
  status varchar(100) NOT NULL DEFAULT "online",  
  last_timestamp timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,  
  irr_enabled varchar(100) NOT NULL,  
  irr_limit int(11) NOT NULL,  
  irr_duration int(11) NOT NULL,  
  mst_timer int(11) NOT NULL,  
  ph_timer int(11) NOT NULL,  
  light_timer int(11) NOT NULL,  
  tmp_timer int(11) NOT NULL,  
  PRIMARY KEY (land_id, node_id)  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

- *land_id*: land identifier in which the node is
- *node_id*: node identifier inside a land
- *protocol*: indicates if the node use the COAP or MQTT protocol
- *address*: indicates the IPv6 address of the node if it uses COAP
- *status*: indicates if the node status is online or offline
- *last_timestamp*: indicates the last time that the node replied to the master
- *irr_enabled*: indicates if the automatic irrigation is enabled
- *irr_limit*: indicates the soil moisture volume below which an irrigation should be initiated
- *irr_duration*: duration of an irrigation
- *mst_timer*: soil moisture measuring period
- *ph_timer*: ph level measuring period
- *light_timer*: light ray measuring period
- *tmp_time*: soil temperature measuring period

3.5.3 The Measurement Table

```
CREATE TABLE IF NOT EXISTS measurement (  
  land_id int(11) NOT NULL,  
  node_id int(11) NOT NULL,  
  m_timestamp timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,  
  sensor varchar(100) NOT NULL,  
  m_value int(11) NOT NULL,  
  PRIMARY KEY(land_id, node_id, m_timestamp, sensor)  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

- *land_id*: identifies the land from which the measurement originates
- *node_id*: identifies the node from which the measurement originates
- *m_timestamp*: timestamp at which the measurement is made
- *sensor*: type of measuring (light ray, soil moisture, soil temperature, ph level)
- *m_value*: measured value

3.5.4 The Violation Table

```
CREATE TABLE IF NOT EXISTS violation (  
  land_id int(11) NOT NULL,  
  node_id int(11) NOT NULL,  
  v_timestamp timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,  
  sensor varchar(100) NOT NULL,  
  v_value int(11) NOT NULL,  
  PRIMARY KEY(land_id, node_id, v_timestamp, sensor)  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

- *land_id*: identifies the land from which the violation originates
- *node_id*: identifies the node from which the violation originates
- *v_timestamp*: timestamp at which the measurement is made
- *sensor*: type of measuring (light-ray, soil moisture, soil temperature, ph level)
- *v_value*: measured value

3.5.5 The Irrigation Table

```
CREATE TABLE IF NOT EXISTS irrigation (  
  land_id int(11) NOT NULL,  
  node_id int(11) NOT NULL,  
  i_timestamp timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,  
  irr_status varchar(100) NOT NULL,  
  PRIMARY KEY(land_id, node_id, i_timestamp)  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

- *land_id*: identifies the land from which the irrigation event originates
- *node_id*: identifies the node from which the irrigation event originates
- *i_timestamp*: timestamp at which the irrigation event happens
- *irr_status*: type of the irrigation event (starting or ending)

4 Implementation

This chapter will explain the node and master implementation. The nodes source codes are written in **C language** by using the **Contiki-NG** project, the master source code is written in **python**.

4.1 The Border Router

The **Border Router** node connects the nodes of the network to internet by allowing the communication with master, in order to do this it uses the **tunslip6** program and the **RPL** routing protocol. The Border Router source code is the same that is in the example folder of the official Contiki-NG repository, for further discussion see the link <https://github.com/contiki-ng/contiki-ng/tree/develop/examples/rpl-border-router>.

4.2 The Node

The COAP and MQTT nodes have the **same high-level functioning**, however at the low-level they are adapted to the differences in their communication protocols. Both nodes perform the same **manual configuration** procedure:

- Starting of the node
- Red LED flashes while waiting for user interaction
- The user presses the button as many times as the value of the land_id
- As soon as the user stops pressing the button, the LED remains steady for two seconds and starts flashing again while waiting for the node_id configuration
- The user presses the button as many times as the value of the node_id
- The node sends to master a configuration request
- If the configuration request is accepted the led turns off and the node is ready, otherwise the led keeps blinking and the node needs to be restarted and reconfigured

After configuration, a node acts autonomously according to the configuration parameters, it **collects data** from the land (soil moisture, light ray, ph level, soil temperature) and it **notifies** them to the **master**.

As mentioned above, a node **independently** performs an **irrigation** operation if the soil moisture is below the configured limit.

Both types of communication uses messages in **JSON** format for the parsing efficiency and for the smaller message size than the other encodings such as XML.

Values recorded by nodes with the **measurements** are **simulated** with random values.

Each node has in its configuration file 'project-conf.h' an environment variable to enable the **serial logging**, in fact for devices with limited resources, logging can be significantly decrease performance.

4.3 The COAP Node

In order to communicate with the COAP nodes it's needed to know their **IPv6 address**, all of them know the master address already at startup. The master knows the address only of the nodes already registered (a node is registered if its configuration is accepted).

A COAP node has **seven resources** and each resource can implement the operations **GET** to return its status, **PUT** to update it (PUT was chosen instead of POST because updating a resource generally corresponds with a complete replacement of all parameters) , **trigger()** to notify the master of the status.

The COAP node has the following resources:

- *config_rsc*: manages the identifiers *land_id* and *node_id*, implements GET e PUT operations.
- *irr_rsc*: deals with events regarding irrigation and its status (on or off), implements GET, PUT and trigger.
- *is_alive_rsc*: stores no parameters and implements only GET, its only purpose is to respond to the master by proving that the node is online
- *mst_rsc*: stores the last soil moisture measure and manages the measurement period, implements GET, PUT and trigger
- *ph_rsc*: stores the last ph level measure and manages the measurement period, implements GET, PUT and trigger
- *light_rsc*: stores the last light ray moisture measure and manages the measurement period, implements GET, PUT and trigger
- *tmp_rsc*: stores the last soil temperature measure and manages the measurement period, implements GET, PUT and trigger

The **master** registered itself to the resources of each COAP nodes that implement the **trigger** operation (it's the COAP **observing** procedure), in this way the node can **notify** the master of its resources status without it making a GET request, in fact when a node makes a measurement, it must be the one to notify it because the master does not know when it occurs.

All the resources have a *state* variable to store the node general status (STATE_CONFIGURED or STATE_ERROR) that depends on the **master acceptance** of its manual configuration. In fact if the node is not correctly configured all its resources must not reply to any request (GET or POST) and not send something to the master.

4.4 The MQTT Node

The MQTT communication doesn't need addresses, but it's needed a **broker** that acts as a central server to which all MQTT nodes and the python server (master) are connected. **Mosquitto** was used as broker on the same machine where the master runs.

Each node registers with the broker (all nodes know the address of the broker) with a **topic** in the format *NODE/land_id/node_id*, the master registers with the topic *SERVER*. The broker manages the address of registered nodes **internally**, so each MQTT message must go through the broker and contain only the topic and body.

This type of communication is **simpler** than that with COAP, in fact if the master wants to send a message to a node, it only needs to specify the topic related to that node, while to send something to the master it is enough to publish the message with the topic *SERVER*.

Unlike COAP communication, which handles master requests separately depending on the resource, in MQTT communication all **messages** are **handled by the same function** that parses the JSON message, identifies the type of command received, and the operation to be performed on the **data structures**.

Since MQTT is a protocol that exploits **TCP**, using this technology is more **resource-consuming** than COAP (which uses UDP). To solve this problem in both the master and the nodes, it is necessary to implement a **message queue** to limit the sending rate. In fact, without the queue, the broker cannot handle the messages and **loses connection** with the nodes.

The MQTT node **data structures** are the following:

- *timers*: it handles all timers like those for periodic measurements
- *irr_config_str*: it handles all configurations relative the irrigation activity
- *measurements_str*: it contains the last values recorded for each type of measure (soil moisture, ph level, light ray, soil temperature)
- *configuration_str*: it handles all configuration parameters like the land_id, node_id, timer duration of the measurements and contains the 'irr_config_str' data structure
- *node_str*: it contains the node status (STATE_CONFIGURED or STATE_ERROR) and all the previous structures except 'timers_str'

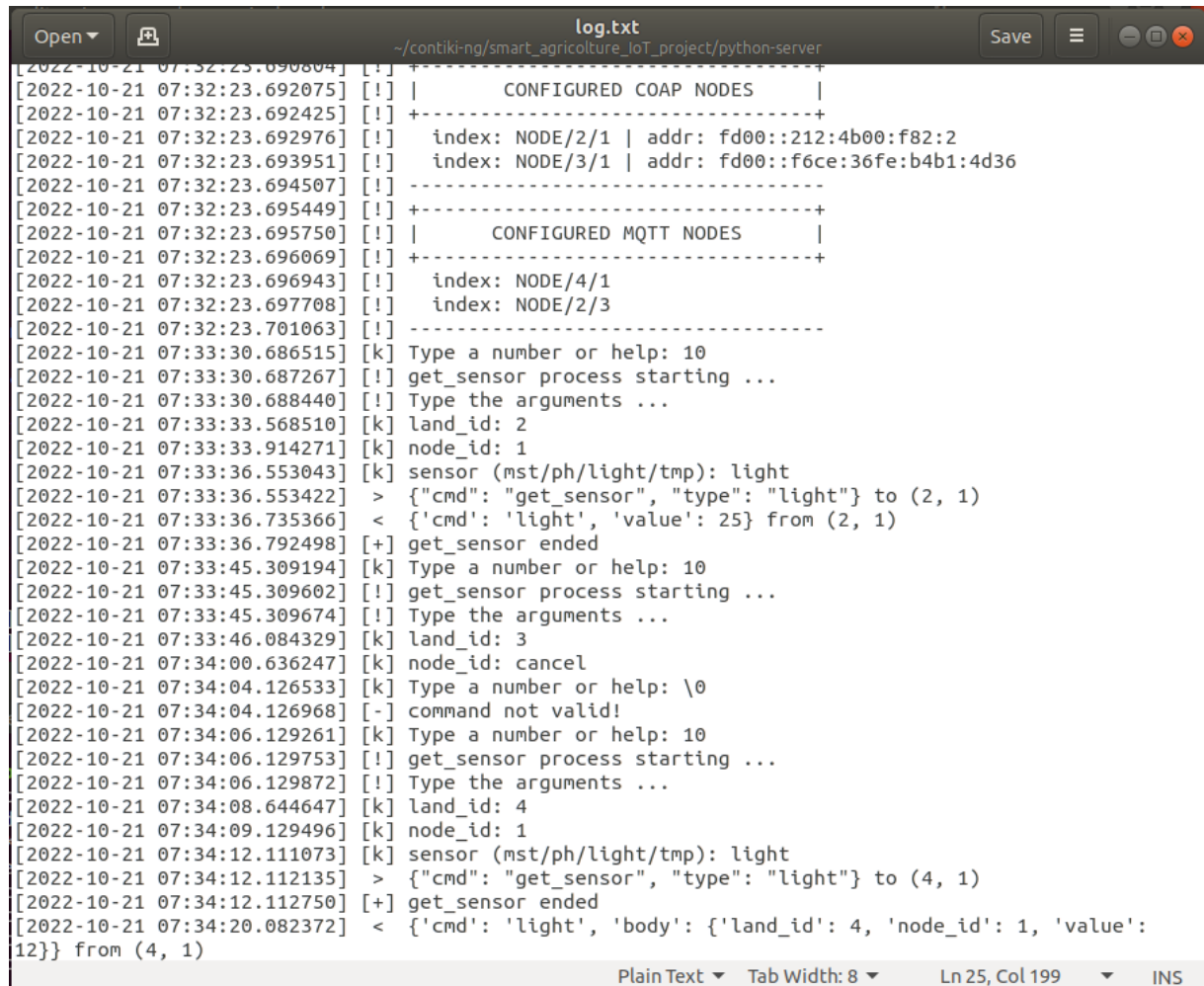
4.5 The Master Node

As already mentioned in the previous chapters, the master is written in **python** and is able to **communicate with both protocols** (COAP and MQTT) using the **Coapathon** and **Paho** libraries.

The master has a **log file** in which all commands received from user, messages sent to nodes, and messages received by nodes are recorded, every **log line** consists of timestamp, message type and message.

The **message types** are as follows:

- [!] : info message
- [+] : success message
- [-] : error message
- [k] : input from keyboard
- > : sending message
- < : receiving message



```
log.txt
~/contiki-ng/smart_agriculture_loT_project/python-server

[2022-10-21 07:32:23.690804] [!] +-----+
[2022-10-21 07:32:23.692075] [!] | CONFIGURED COAP NODES |
[2022-10-21 07:32:23.692425] [!] +-----+
[2022-10-21 07:32:23.692976] [!] index: NODE/2/1 | addr: fd00::212:4b00:f82:2
[2022-10-21 07:32:23.693951] [!] index: NODE/3/1 | addr: fd00::f6ce:36fe:b4b1:4d36
[2022-10-21 07:32:23.694507] [!] +-----+
[2022-10-21 07:32:23.695449] [!] | CONFIGURED MQTT NODES |
[2022-10-21 07:32:23.696069] [!] +-----+
[2022-10-21 07:32:23.696943] [!] index: NODE/4/1
[2022-10-21 07:32:23.697708] [!] index: NODE/2/3
[2022-10-21 07:32:23.701063] [!] +-----+
[2022-10-21 07:33:30.686515] [k] Type a number or help: 10
[2022-10-21 07:33:30.687267] [!] get_sensor process starting ...
[2022-10-21 07:33:30.688440] [!] Type the arguments ...
[2022-10-21 07:33:33.568510] [k] land_id: 2
[2022-10-21 07:33:33.914271] [k] node_id: 1
[2022-10-21 07:33:36.553043] [k] sensor (mst/ph/light/tmp): light
[2022-10-21 07:33:36.553422] > {"cmd": "get_sensor", "type": "light"} to (2, 1)
[2022-10-21 07:33:36.735366] < {'cmd': 'light', 'value': 25} from (2, 1)
[2022-10-21 07:33:36.792498] [+] get_sensor ended
[2022-10-21 07:33:45.309194] [k] Type a number or help: 10
[2022-10-21 07:33:45.309602] [!] get_sensor process starting ...
[2022-10-21 07:33:45.309674] [!] Type the arguments ...
[2022-10-21 07:33:46.084329] [k] land_id: 3
[2022-10-21 07:34:00.636247] [k] node_id: cancel
[2022-10-21 07:34:04.126533] [k] Type a number or help: \0
[2022-10-21 07:34:04.126968] [-] command not valid!
[2022-10-21 07:34:06.129261] [k] Type a number or help: 10
[2022-10-21 07:34:06.129753] [!] get_sensor process starting ...
[2022-10-21 07:34:06.129872] [!] Type the arguments ...
[2022-10-21 07:34:08.644647] [k] land_id: 4
[2022-10-21 07:34:09.129496] [k] node_id: 1
[2022-10-21 07:34:12.111073] [k] sensor (mst/ph/light/tmp): light
[2022-10-21 07:34:12.112135] > {"cmd": "get_sensor", "type": "light"} to (4, 1)
[2022-10-21 07:34:12.112750] [+] get_sensor ended
[2022-10-21 07:34:20.082372] < {'cmd': 'light', 'body': {'land_id': 4, 'node_id': 1, 'value':
12}} from (4, 1)
```

Figure 2: the master log file

The master consists of **5 threads**

- t1: it listens to user commands
- t2: it is responsible for periodically testing whether nodes are online
- t3: it is responsible for send MQTT message
- t4; it is responsible for receive MQTT message
- t5: it is responsible for receive COAP message

The **t1 thread** sends COAP messages and interacts with mysql database at the request of the user, about MQTT messages, t1 is responsible only for push them in the **messages queue**, then the **t3 thread** will send the MQTT messages taking them from the queue with a **fixed period** (5 seconds). The reason why this approach was chosen is due to the limited resources of the nodes and the overhead of the Mosquitto broker.

At **startup**, the master retrieves the list of previously registered nodes from the mysql server and tries to contact them on the network (**discovery operation**). All responding nodes are stored in a **cache** with the purpose of always knowing which nodes are reachable without having to do a new search. **Periodically** a new search is performed to make sure that all nodes are still online.

Whenever the master communicates with a node, it also takes care of modifying on the mysql server its data if it has changed, in addition, at the request of the user, the master can send any change to the nodes configurations.

The main functionalities of the master are as follows:

- Registering nodes to the network
- Collect data from node measurements
- Identify outliers ('violations') of measurements
- Keep the database updated with node status, measurements, violations, and irrigations
- Execute user commands

Node registration occurs at master startup for already configured nodes, or during normal execution for other nodes. New COAP nodes register by contacting the master through a master **resource** available only for this purpose, whereas new MQTT nodes register to the master by sending a **configuration request** simply using the *SERVER* topic and specifying in the JSON message format the type of message (in this case specifying that it is a configuration request). A **configuration/registration request** from a node can be **rejected** if the land_id does not match an existing land, or if a node with the same node_id already exists in the network.

The **commands** that the **user** can have the master execute are as follows and are visible through the *help* command:

Show data from database (no message to nodes)

- *view lands*
- *view configurations*
- *view last measurements*
- *view last violations*
- *view last irrigations*

Commands to nodes (send message to nodes and update database)

- *irr_cmd* (changes the irrigation configuration of an online node)
- *get_config* (asks a node to send its configuration)
- *assign_config* (assigns a new configuration to a node)
- *timer_cmd* (change one of the measurement timers of a node)
- *get_sensor* (asks to a node to perform a measurement)
- *is_alive* (asks to a one or more nodes if they are online)

ADD data to database (no message to nodes)

- *add new land*
- *add new configuration*
- *add irrigation event*
- *add measurement event*

UPDATE data (no message to nodes)

- *update land*
- *update configuration*
- *set node online*
- *set all node offline*

DELETE data from database (no message to nodes)

- *delete land*
- *delete configuration*
- *delete many irrigation events*
- *delete one irrigation event*
- *delete many measurement events*
- *delete one measurement event*
- *delete many violation*

- *delete on violation*

Other commands

- *test received messages* (simulates the arrival of a message from a node)
- *list of nodes in memory* (shows the list of currently registered nodes)
- *exit* (ends the master with all its threads)

Running on the same machine where the master is running are the **mySQL server** on docker, **Grafana** on docker, and **Mosquitto**.

5 Tests

For the **test phase** it was used a network consisting of the **master** on a Ubuntu machine, **5 nodes** of which one is the **Border Router** and the others are **half MQTT nodes** and **half COAP nodes**. Two tests were done, the first is a simulation performed with the **Cooja** tool integrated in Contiki-NG project, the second was performed by using **real boards**.

Prerequisites:

- Ubuntu 18.04
- Docker
- Python3
- Python Paho
- Python Coapthon
- Python PySerial
- Mysql on Docker with database (see file database.sql in github repo)
- Grafana on Docker with mysql extension
- Contiki-NG v4.7 on Docker (view Contiki-NG repo to install it)
- Mosquitto MQTT broker
- Project Source Files putted into Contiki-NG folder

5.1 Cooja Test

Below is the list of steps to start the simulation

- start docker
- start mysql container
- start Contiki container
- run "mosquitto -v" in a terminal
- go to project folder and run "python3 python-server/server.py"
- run "docker exec -it [contiki-id] bash"
- run "cd tools/cooja"
- run "ant run" to start the simulation tool
- open simulation using the file "smart_agriculture_cooja_sim.csc"
- go to project folder and run "cd border-router/web-server"
- run "make TARGET=cooja connect-router-cooja"
- press start in the simulation window

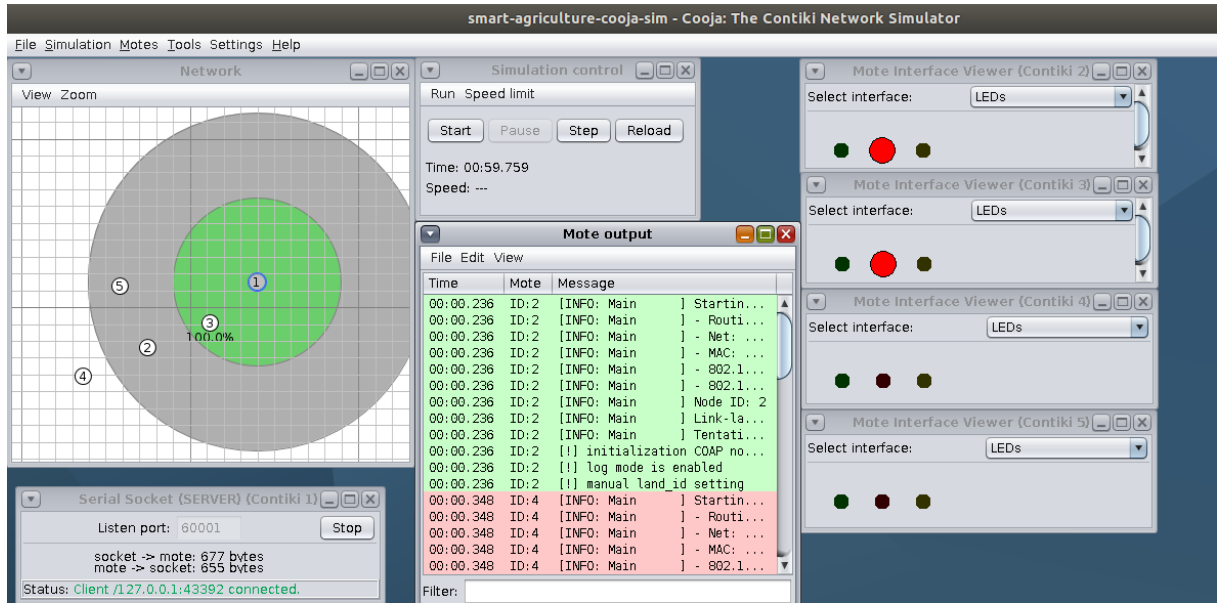


Figure 3: The Cooja Simulation

5.2 Test With Real nodes

For the real test, the same network as in the simulation was replicated using real boards as nodes. Specifically, the nodes were configured as follows:

- Border Router: CC2650 Launchpad
- COAP node 1: CC2650 Launchpad
- COAP node 2: nRF52840 dongle
- MQTT node 1: nRF52840 dongle
- MQTT node 2: nRF52840 dongle

The master, database, and MQTT broker were put on the same Ubuntu machine as in the simulation.

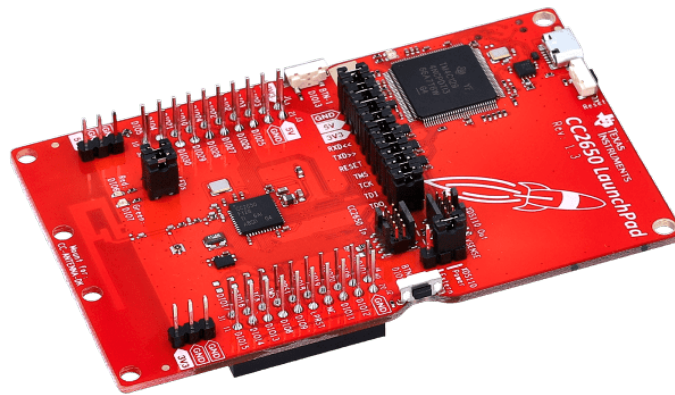


Figure 4: CC2650 Launchpad

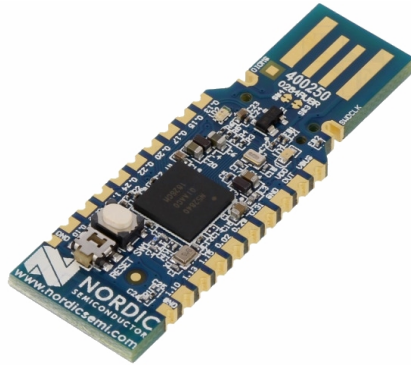


Figure 5: nRF52840 dongle

To test the network it is first necessary to **flash the boards** with the source code, in the github repository there are folders with the boards configurations.

To load the code onto the boards, it is necessary to connect them to a usb port and check the **tttyACM** name assigned to it by the operating system (run on the terminal "ls /dev").

Terminal **commands to flash the boards** (each command must be sent from terminal starting from the folder where the node source files are):

- Border Router:
"make TARGET=cc26x0-cc13x0 BOARD=/launchpad/cc2650 PORT=/dev/ttyACMx NODEID=0x0001 border-router.upload"
- COAP node 1:
"make TARGET=cc26x0-cc13x0 BOARD=/launchpad/cc2650 PORT=/dev/ttyACMx NODEID=0x0002 coap-node.upload"
- COAP node 2:
"make TARGET=nrf52840 BOARD=dongle PORT=/dev/ttyACMx NODEID=0x0003 coap-node.dfu-upload"
- MQTT node 1:
"make TARGET=nrf52840 BOARD=dongle PORT=/dev/ttyACMx NODEID=0x0004 mqtt-node.dfu-upload"
- MQTT node 2:
"make TARGET=nrf52840 BOARD=dongle PORT=/dev/ttyACMx NODEID=0x0005 mqtt-node.dfu-upload"

In the above commands replace "ttyACMx" with the port corresponding to the board (e.g., ttyACM0, ttyACM1 ...), note that the launchpad occupies two ttyACM ports, while the dongle occupies only one.

After board flashing it's possible **read the serial output** from the board with the following commands:

- CC2650 Launchpad:
"make TARGET=cc26x0-cc13x0 BOARD=/launchpad/cc2650 PORT=/dev/ttyACMx login"
- nRF52840 dongle:
"make TARGET=nrf52840 BOARD=dongle PORT=/dev/ttyACMx login"

Finally, in order to connect the Border Router to the master, it's necessary to start the **webserver** with the command "make TARGET=cc26x0-cc13x0 PORT=/dev/ttyACMx connect-router" (run the command inside the webserver folder).

The image displays four terminal windows showing the configuration of a smart agriculture IoT project. The first terminal shows the configuration of a Master Node with COAP and MQTT nodes. The second terminal shows the configuration of an MQTT Node. The third terminal shows the configuration of a COAP Node. The fourth terminal shows the configuration of a COAP Node sensor.

```

osboxes@osboxes: ~/kontiki-ng/smart_agriculture_iot_project/python-server
File Edit View Search Terminal Help

1/29
5[+]
|-----+-----|
| CONFIGURED COAP NODES |
|-----+-----|
1[+]
|-----+-----|
| CONFIGURED MQTT NODES |
|-----+-----|
1[+]
| Index: NODE/4/1 |
| Index: NODE/2/3 |
|-----+-----|
1[+]
5[+] Type a number or help: 29
1[+]
|-----+-----|
| CONFIGURED COAP NODES |
|-----+-----|
1[+]
| Index: NODE/2/1 | addr: fd00::212:4b00:f82:2 |
| Index: NODE/3/1 | addr: fd00::f6ce:3f9f:b4b1:4d36 |
|-----+-----|
1[+]
|-----+-----|
| CONFIGURED MQTT NODES |
|-----+-----|
1[+]
| Index: NODE/4/1 |
| Index: NODE/2/3 |
|-----+-----|
1[+]
5[+] Type a number or help:

osboxes@osboxes: ~/kontiki-ng/smart_agriculture_iot_project/border-router/webserver
File Edit View Search Terminal Help

osboxes@osboxes:~/kontiki-ng/smart_agriculture_iot_project/border-router/webserver$ make TARGET=cc26x
0-13x
PORT=/dev/ttyACM0 connect-router
sudo -u ./ ./tools/serial-to-tun.sh -s /dev/ttyACM0 fd00::1/64
*****SLIP started on `./dev/ttyACM0`
opened tun device `./dev/tun0`
ifconfig tun0 inet 'hostname' mtu 1500 up
ifconfig tun0 add fd00::1/64
ifconfig tun0 add fe80::0:0:0:1/64
ifconfig tun0

tun0: flags=4305<UP,POINTOPOINT,RUNNING,NOARP,MULTICAST> mtu 1500
inet 127.0.1.1 netmask 255.255.255.255 destination 127.0.1.1
inet6 fd00::1 prefixlen 64 scopeid 0x0<global>
inet6 fe80::1 prefixlen 64 scopeid 0x20<link>
inet6 fe80::2db8:31ef:9c81:ee30 prefixlen 64 scopeid 0x20<link>
unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00 txqueuelen 500 (UNSPEC)
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

```

Master Node

```

osboxes@osboxes: ~/kontiki-ng/smart_agriculture_iot_project/mqtt-node-s...
File Edit View Search Terminal Help

> {"cmd":"light","body":{"land_id":2,"node_id":3,"value":93}}
[!] adding message to mqtt list
[+] soil temperature detected: 33
> {"cmd":"tmp","body":{"land_id":2,"node_id":3,"value":33}}
[!] adding message to mqtt list
[!] Publishing [ irr_status, len: 102]
[!] Publishing [ timer_status, len: 116]
[!] Publishing [ moisture, len: 62]
[!] Publishing [ ph, len: 55]
[!] Publishing [ light, len: 58]
[!] Publishing [ tmp, len: 57]

osboxes@osboxes: ~/kontiki-ng/smart_agriculture_iot_project/coap-node-sensor
File Edit View Search Terminal Help

[!] actual configuration:
land_id: 3
node_id: 1
irr_config: { enabled: true, irr_limit: 20, irr_duration: 20}
mst_timer: 60
ph_timer: 24
light_timer: 45
tmp_timer: 46
node in online

```

MQTT Node

```

osboxes@osboxes: ~/kontiki-ng/smart_agriculture_iot_project/coap-node-sensor
File Edit View Search Terminal Help

[!] node in online
< get sensor/tmp
[+] soil temperature detected: 30
> {"cmd":"tmp","value":30}

```

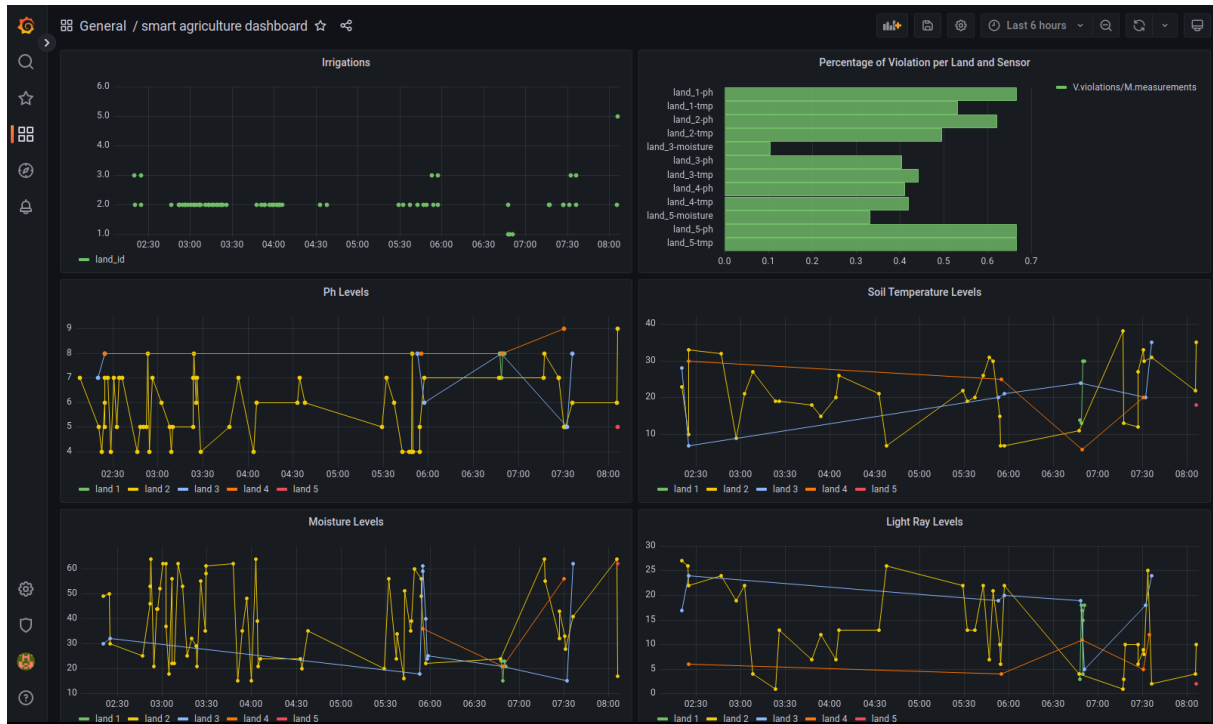
COAP Node

Figure 6: Terminals with nodes output

6 Grafana

Grafana is a web tool that allows to create a **custom dashboard** to **monitor data** from various sources, in this case the source is mysql. The following images show the designed dashboard.

Note that the measurement data is very unstable because it is randomly generated by the nodes.



AVG Soil Temperature per Land				AVG Ph per Land			
land	avg_soil_temperature	min_bound	max_bound	land	avg_ph	min_bound	max_bound
1	22.3	4	22	1	6.93	6	7
2	20.9	10	25	2	6.27	6	7
3	21.6	16	33	3	6.51	5	7
4	16.3	18	36	4	6.88	6	8
5	19	20	38	5	6	6	8

Node Status			AVG Moisture per Land		
node	status	timestamp	land	avg_moisture	trashold
NODE/2/1	online	2022-10-21 08:05:51	1	32.1	14
NODE/2/2	online	2022-10-21 08:05:31	2	39.0	11
NODE/2/3	online	2022-10-21 07:30:15	3	40.1	22
NODE/3/1	online	2022-10-21 08:06:08	4	43	8
NODE/4/1	online	2022-10-21 07:30:05	5	38.3	27
NODE/5/1	online	2022-10-21 08:06:06			
NODE/1/1	offline	2022-10-21 07:19:19			

Configurations										
node	protocol	address	irr_enabled	irr_limit	irr_duration	mst_timer	ph_timer	light_timer	tmp_	
NODE/1/1	COAP	fd00::205.5.5.5	true	22	20	1	1	1		
NODE/1/2	null	null	true	22	20	720	720	30		
NODE/2/1	MQTT		true	0	60	60	60	60		
NODE/2/2	COAP	fd00::202.2.2.2	true	0	60	60	60	1		
NODE/2/3	MQTT		true	0	60	60	60	60		
NODE/2/4	MQTT		true	25	25	720	1440	60		
NODE/2/5	null	null	true	25	25	69	60	60		
NODE/3/1	MQTT	null	true	20	20	60	24	45		
NODE/3/2	MQTT		true	20	20	69	1440	120		
NODE/3/4	MQTT		true	20	20	1440	1440	120		
Lands										
id	area	locality	name	crop	soil_type	mst_trashold	min_ph	max_ph	min_tmp	
1	0.100	pag 5, part 4, Carbo...	Fondi	lettuce	loam	14	6	7	4	
2	0.500	pag 2 part 1 Carbog...	Crafeno	tomato	silt loam	11	6	7	10	
3	1	pag 4 part 5 Carbog...	Filaro	olive	clay loam	22	5	7	16	
4	2	pag 3 part 2 Carbog...	Galli	olive	sandy loam	8	6	8	18	
5	1	pag 8 part 3 Carbog...	Corpiè	apple	slity clay	27	6	8	20	

(it is possible to import the dashboard from the "smart-agriculture-dashboard.json" file in the github repository)