University of Pisa

Internet of Things

Project

Smart GreenHouse

Grillea Francesco

Pardini Marco

A.Y. 2022-2023

Contents

Chapter 1

Overview

In this chapter, we present a comprehensive overview of our idea of creating an intelligent and efficient greenhouse environment. Our project revolves around the concept of harnessing the power of Internet of Things (IoT) technology to improve greenhouse cultivation. The objective is to optimize the growth conditions for various plant species and enhance overall productivity. This chapter provides an introduction to the project, outlining its objectives, key components, and the significance of implementing a smart greenhouse system.

1.1 Why a smart greenhouse?

Certain vegetables, such as tomatoes, require a consistently warm temperature throughout their growth cycle to thrive. These heat-loving plants rely on warmth to stimulate optimal growth, fruit development, and flavor. However, deviations from the ideal temperature range can pose significant challenges: if the temperature rises above the preferred level, it can lead to heat stress, causing wilting, diminished fruit production, and reduced overall plant vigor; conversely, if the temperature drops below the desired range, the plants may experience stunted growth, delayed ripening, and increased susceptibility to diseases. Maintaining a stable and warm temperature environment is crucial for ensuring the successful cultivation of heat-sensitive vegetables like tomatoes.

The ideal temperature for growing tomatoes typically ranges between 27°C 34°C during the day. These warm-season plants thrive in temperatures that are consistently within this range. At night, temperatures between 20°C and 24°C are generally considered optimal for tomatoes.

1.1.1 The importance of tents

Controlling temperature in a greenhouse can be achieved thanks to the strategic opening or closing of tents. These tents, often made of materials such as shade cloth or plastic, can be adjusted to regulate the amount of sunlight and heat that enters the greenhouse. On hot days, opening up the tents allows for increased air circulation and helps dissipate excess heat, preventing the temperature from rising to detrimental levels. This ventilation helps cool down the greenhouse and creates a more favorable environment for the plants. Conversely, on cooler days or during the night, closing the tents helps retain heat and insulate the greenhouse, preventing temperature drops that could be harmful to the plants. This method of using tents as temperature regulators provides a flexible and efficient way to maintain optimal growing conditions for plants in a greenhouse setting.

1.1.2 Manual solutions

While manually opening and closing tents to control temperature in a greenhouse can be effective, it can lead to certain challenges. One significant issue is the time and effort required. Greenhouses often have numerous tents or shades that need to be adjusted, and doing so manually can be time-consuming, especially in larger setups. It requires regular monitoring of weather conditions and constant vigilance to ensure timely adjustments. Moreover, one of the problems could be the risk of the crank slipping out of hand and potentially hitting the worker. In addition, human error can occur, leading to improper adjustments that may result in temperature fluctuations that harm the delicate plants. To mitigate these challenges, automated systems or technology-assisted solutions are available that can streamline the process, reduce labor, and provide more precise control over temperature regulation in a greenhouse.

1.1.3 The idea

The objective of our project is to develop an IoT environment that, thanks to the power of sensors, can effectively control the temperature inside a greenhouse. By integrating sensors throughout the greenhouse, we aim to monitor the temperature levels with precision and accuracy. These sensors will continuously collect data and transmit it to a cloud system, then, based on the collected information, the remote control application system will then activate or deactivate actuators that control the opening or closing of the tents. If the sensed temperature goes above 28°, the tents will be rolled up, while

if the temperature goes below 26° the tents will be rolled down. This automated approach eliminates the need for manual adjustments, reduces the risk of human error, and provides a more efficient and reliable method of maintaining optimal temperature conditions for the greenhouse.

1.2 Hardware limitations and solutions

In a typical environment, an IoT project involving a greenhouse would utilize various sensors and actuators to monitor and control temperature levels, as well as adjust the positioning of the greenhouse's tents. However, due to resource constraints and specific project requirements, the available hardware for our implementation is limited to dongle nRF52840.

To emulate a realistic environment, we decided to simulate temperature variations using the nRF52840 dongle. The dongle will generate random temperatures within a range 24.0°C ±3.5°C for a low-temperature scenario and 34.0°C ±3.5°C for an high-temperature scenario. To simulate increasing and decreasing temperatures, we will utilize the button on the dongle. When the button is pressed, it will trigger a change in the mean temperature value that will increase (or decrease) the temperature by 10°C .

In our simulation, we will represent the actuators using additional dongles, with LEDs serving as visual indicators for representing the position of the greenhouse tents:

- GREEN LED ON means that tents are in an upright position.
- NO LED means that tents are lowered down.

By employing this simulation approach, we can effectively emulate temperature fluctuations and simulate the control of the greenhouse's tent positioning system using the nRF52840 dongle and LEDs as actuator proxies. This enables us to evaluate and test the functionality of our IoT project in a practical and accessible manner.

Chapter 2

Architecture and Implementation

Before going into details of the implementation, let's describe the chosen placement of sensor inside a real scenario of a $3200m^2$ greenhouse. Dongle nRF52840 got a $37m$ communication range, so we decided to locate the Border Router in the middle of the greenhouse, two Sensor Nodes $20m$ far from Border Router in such a way to monitor the temperature at both ends of the greenhouse and three Actuator Nodes respective $20m$, $20m$ and $30m$ far from the Border Router.

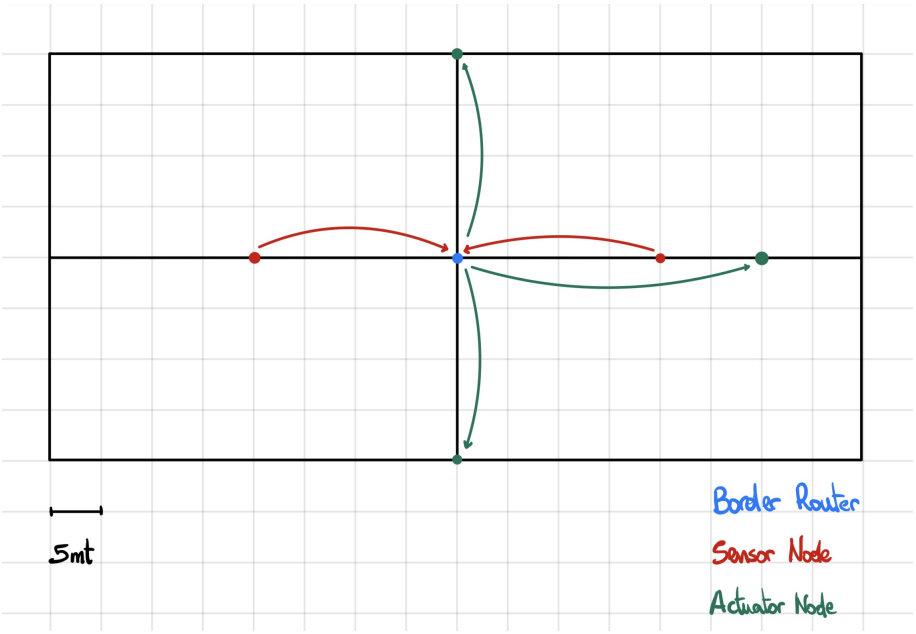

Figure 2.1: Nodes Placement

The figure 2.2 shows the overall architecture of the environment: the left side represent the LLN made up by the sensor nodes and actuators, and the right side represent the Application Side that collects data from the LLN and send back a response according to some application logic. Obviously, the LLN comunicate with the Application Side through a specific node that acts as Border Router.

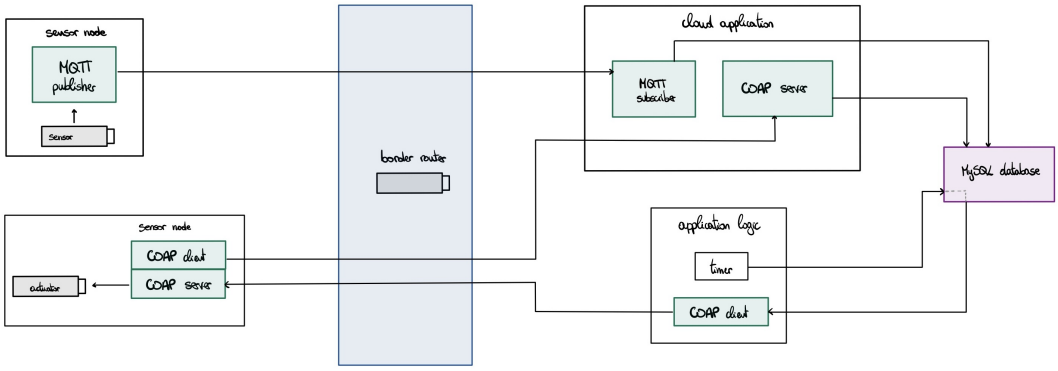

Figure 2.2: Architecture

2.1 LLN

In this section we're going into details of Sensor and Actuator nodes that compose the Low-Power and Lossy Network.

2.1.1 JSON vs XML

We decided to use JSON as message format for messages published by MQTT node. Thanks to JSON, the MQTT node ensures compatibility with a broad range of programming languages and platforms, enabling seamless integration with various IoT devices and applications. Its human-readable syntax facilitates debugging and troubleshooting, while its efficient data serialization and deserialization processes contribute to optimal message transmission and processing efficiency. Furthermore, the extensive availability of JSON parsers and libraries simplifies development efforts and accelerates the implementation of MQTT-based IoT solutions.

XML was deemed unnecessary in this particular case for several reasons. First and foremost, XML (eXtensible Markup Language) tends to be heavier compared to other data formats such as JSON. Since the application revolves

around a smart greenhouse, we decided to chose a lighter message structure like JSON which ensures efficient communication and minimizes the burden on the limited processing power and memory of the devices involved. Additionally, XML's primary advantage lies in its support for validation schemas, which define strict rules for data structure and content. However, in the context of a smart greenhouse, the requirements for data validation might not be as stringent, allowing for a more flexible and lightweight approach.

2.1.2 MQTT Node

The MQTT Nodes represent Sensor Nodes which main functionality is to sense the external environment and periodically publish the sensed temperature to a MQTT Broker. As mentioned previously, since Dongle nRF52840 has neither a real temperature sensor nor the floating point support, we decided to generate random values between 240 ±35. When the button on the Dongle is pressed, it triggers an handler that simulate temperature variation just increasing (or decreasing) a `variation` variable by 100 that will affect the generation of temperatures: it will produce values between 340 ±35. Values are generated every 5 seconds and published to a MQTT Broker at `[fd00::1]` specifying also the Node MAC and the Greenhouse ID the node belongs to.

Also we decided to use LEDs as visual indicator of the status of the sensor node:

- RED: the node is not connected to the Broker yet.
- YELLOW: the connection to the broker has been established successfully.
- GREEN: a message has been published to the Broker.

Message Structure

This is the structure of the JSON message:

```
{
    "app": "smart_greenhouse",
    "greenhouse_id": 1,
    "MAC": "EF:3E:A5:D9:5B:F7",
    "Topic": "Temperature",
    "Value": 248
}
```

The "app" field was introduced to facilitate the deployment of the project in an ambient environment where multiple applications might coexist. By including this field, the system can easily identify and process messages from the smart greenhouse app, ensuring seamless integration with other apps within the same ecosystem. Similarly, the "greenhouse_id" field allows for the monitoring of multiple greenhouses simultaneously. This design choice provides scalability, enabling the project to accommodate different greenhouse setups and efficiently manage data generated from different locations. Furthermore, the inclusion of the "MAC" field in the message schema serves debugging and fault tolerance needs. It provides information about the sending node's IP address, which aids in troubleshooting and identifying potential issues within the network by directly dealing with the failing node, instead of having to turn down the entire network.

2.1.3 CoAP Node

The CoAP Nodes represent Actuator Nodes and acts as both client and server.

Client Side

First of all, each CoAP Node has to register itself to the Cloud Application sending its IP and role: in our implementation all CoAP nodes acts as a tent actuator, but in further implementations can be different role for different Actuator Nodes. During this registration phase the RED LED of the node is on.

Message structure

As mentioned before, JSON was determined to be the most suitable choice. This is the message structure:

```
{
    "app":"smart_greenhouse",
    "role":"tent",
    "greenhouse_id:1
}
```

We decided to include a "role" field for scalability purposes, in order to permit an easy future integration with other actuators. The IP of the actuator is obtained from the server side residing in the Cloud Application.

Server Side

Once the CoAP nodes are registered to the Cloud Application i.e. their IPs are stored in the MySQL database, a CoAP node must behave as server to execute the requests received by the Application Side to open and close tents. As mentioned previously, since we don't have a real actuator that opens/closes the tends through a motor, we decided to simulate such actions using leds. When a PUT request is received from the Application Side, the CoAP Node just turns on the GREEN LED when a "tent up" command is received, and turns off the GREEN LED when a "tent down" command is received: those rules are defined in the `resources/res-tent.c` file.

2.2 Application Side

In this section we're going into details of Cloud Application and Application Logic which is implemented into the Remote Control Application.

2.2.1 Cloud Application

The cloud application is made up of three main components:

- MySQL Database in which telemetry data and actuator IPs are stored.
- MQTT Subscriber which contacts the broker to `127.0.0.1:1883` and subscribe to the topic `temperature` where the temperature values are published. Whenever a new message is received, it stores all information related to that telemetry value into the SensorData table of the MySQL Database.
- CoAP Server which receives a registration message from CoAP Nodes (Client Side) that contains the IPs of the Actuator Nodes, their roles and the greenhouses they belong to. Then the CoAP Server stores all those information inside the Actuators table of the MySQL Database.

2.2.2 Remote Control Application

The Remote Control Application is made up of four main components:

- CoAP Client which sends a message to the CoAP Node (server side).
- Database handler which just retrieves the IPs of the Actuator Nodes and the last `numRows` temperatures stored from the Cloud Application.

- a Thread that periodically checks every 10 seconds the last 10 temperatures stored on the SensorData table of the MySQL database. If the average of those 10 values is greater than a specified threshold (`temperatureThreshold = 28`) and the tends are closed, the thread will ask the CoAP Client to send a request to the CoAP Node to open the tends; otherwise if the average temperature is lower than the same threshold and the tends are opened, the thread will ask the CoAP Client to send a request to the CoAP Node to close them. Note that using this approach we avoid to send "useless" messages since that a request is sent only if the tends are in the opposite state of the one specified in the request.

- UserInputHandler class, that runs the Main method and is in charge of managing inputs from the user. In particular, the user has the possibility to:

  - Run the application by himself by entering button 1: in this case the user may decide to turn tents up or down whenever he wants, or to restart again the automatic remote application control;
  - Change the temperature threshold by entering button 2;
  - Change the timing with which the database is being checked by entering 3;
  - Leave by entering 4.

2.2.3 MySQL Database

In this section we're going to describe the table used to store data received from the Nodes.

The SensorData table stores all data related to the temperature received by the MQTT Nodes: as the temperature value, the MAC of the Node that senses the value and the timestamp of the moment value has been received.

SensorData					
ID	ID_Greenhouse	MAC_Sensor	Topic	Value	Timestamp
735	1	D0:3D:77:E7:5B:EB	Temperature	24.1	2023-06-18 14:07:14
736	1	EF:3E:A5:D9:5B:F7	Temperature	26.3	2023-06-18 14:07:18

Table 2.1: SensorData Table

The Actuators table stores all the data related to the Actuator Nodes: as its IP, the role and the greenhouse it belongs to.

Actuators		
IP_Actuator	ID_Greenhouse	Role
fd00:0:0:0:204:4:4:4	1	tent
fd00:0:0:0:205:5:5:5	1	tent
fd00:0:0:0:206:6:6:6	1	tent

Table 2.2: Actuators Table

Chapter 3

What do we have today?

The objective of this final chapter is to provide an overview of the existing market offerings regarding temperature monitoring sensors that can be connected to an nRF52840 device, as well as explore real-world actuators capable of controlling tent movements.

3.1 Temperature sensors

Some physical temperature sensors, such as the DS18B20 and DHT22, can be seamlessly attached to the nRF52840 dongle for temperature monitoring purposes. The nRF52840 dongle serves as a versatile platform, providing the necessary hardware and connectivity features to interface with these sensors.

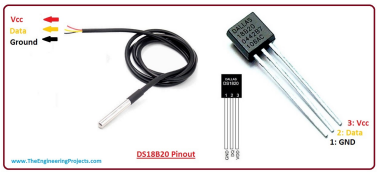

Figure 3.1: DS18B20 temperature sensor

The DS18B20, a digital temperature sensor, can be easily connected to the nRF52840 dongle through its OneWire interface, enabling accurate temperature measurements.

Figure 3.2: DHT22 temperature sensor

Similarly, the DHT22, a combined temperature and humidity sensor, can be interfaced with the nRF52840 dongle to monitor both parameters. By leveraging the capabilities of the nRF52840 dongle, developers can integrate these sensors into their IoT applications, collect temperature data, and employ various wireless communication protocols to transmit the data to a central system for further analysis and control. This flexibility and compatibility of the nRF52840 dongle enable seamless integration of these real sensors, opening up possibilities for temperature monitoring applications in a wide range of scenarios.

3.2 Actuators

During our research, we have come across rotating motors suitable for rolling up and down tents, such as the SGMJV-A5AAA61 model from YASKAWA.

Figure 3.3: YASKAWA rotator

These motor rotators are electrically powered and provide reliable mechanical movement for tent control. However, they do not possess built-in IoT communication capabilities. To integrate them into an IoT-enabled system, we can leverage the nRF52840 dongle. By associating the nRF52840 dongle with these motors, we can utilize its electrical signaling capabilities to provide control signals for turning the rotators on or off. This integration allows for the coordination of tent movements with an IoT network, enabling remote control, automation, and synchronization with other devices or sensors within the system. The nRF52840 dongle acts as a bridge, facilitating the communication between the motor rotators and the IoT ecosystem, enhancing the overall functionality and control options of the tent system.