UNIVERSITÀ DI PISA

Computer Engineering

Internet of Things

Beekeper System

Group Project Report

TEAM MEMBERS:
Biagio Cornacchia
Matteo Abaterusso

Academic Year: 2022/2023

Contents

1 Introduction **2**

2 Design **3**
2.1 Cloud Application 3
2.2 Remote Control Application 3
2.3 MQTT Sensor 5
2.4 CoAP Actuator 6

3 Implementation **8**
3.1 Cloud Application 8
3.2 Remote Control Application 8
3.3 MQTT Sensor Node 8
3.3.1 Message Types 9
3.3.2 MQTT Topics 9
3.3.3 Data Encoding 9
3.3.4 Other details 10
3.4 CoAP Actuator 10
3.4.1 Message Types 10
3.4.2 CoAP Resource 10
3.4.3 Data Encoding 10
3.4.4 Other details 11
3.5 RPL Border Router 11

4 Deployment **12**

1 Introduction

The aim of the project is to integrate the modern Internet of Things technologies with beekeeping. In general, bee colonies are organized into hives. For the colony to be healthy, attention must be paid to the conditions inside the hive, such as **humidity**, **temperature** and the **ratio** of **carbon dioxide** to **oxygen**. So being aware of these data can be very useful for the beekeeper, who can possibly intervene to help the colony.

One of the events that the beekeeper must be careful about is swarming, which is the possibility that the colony may decide to leave the hive. For this purpose, **frequency** and **noise** within the hive are factors that can be exploited by the beekeeper to prevent such an event.

Another interesting metric to monitor is the **number** of **bees leaving** and **entering** the hive. This value indicates the mortality rate of a colony, which can be influenced by the use of insecticides in the area or the presence of varrosis in the colony.

Finally, knowing the **weight** of the hive provides insight into how much honey is imported daily and whether new frames need to be inserted. In addition, weight is an index to understand the space available in the hive. If this is too low, the queen may decide not to lay eggs and the bees may stop collecting honey.

Figure 1: Example of a smart hive.

Bees are autonomous, but it is possible to help them in their growth and honey production by acting on the levels of temperature, humidity and ratio of carbon dioxide to oxygen. To achieve this, a **heating system** is used to warm the hive while a **ventilation system** is used to lower the temperature, humidity and amount of carbon dioxide. The beekeeper must be able to decide how the values measured by the sensors are to be used to control the implementation systems independently, according to his or her preferences.

2 Design

The key features of the system are:

- **Sensor integration**: once the beekeeper has connected the various sensors and actuators to the hive, they can communicate with the cloud application through the border router. At this point, the application will consider them to be unlinked nodes.
- **Area creation**: the beekeeper can create virtual areas within the application to represent their apiaries.
- **Adding hives**: once an area has been defined, the beekeeper can create hives. To each hive, the beekeeper can connect sensors and actuators in the '*unlinked*' state.
- **Rule-based implementation**: the beekeeper can associate specific rules with each hive. These allow the actuators to be controlled based on the values measured by the sensors.

The components of the system will be analyzed in the following sections.

2.1 Cloud Application

The main purpose of the cloud application is to collect data from the low power and lossy network. It acts as **MQTT client** and as **CoAP server**. The former role allows to receive all measurements collected by the MQTT sensors, while the latter allows a CoAP actuator to register to the system. In addition, it receives node configuration messages from both sensors and actuators, in order to update their state information.

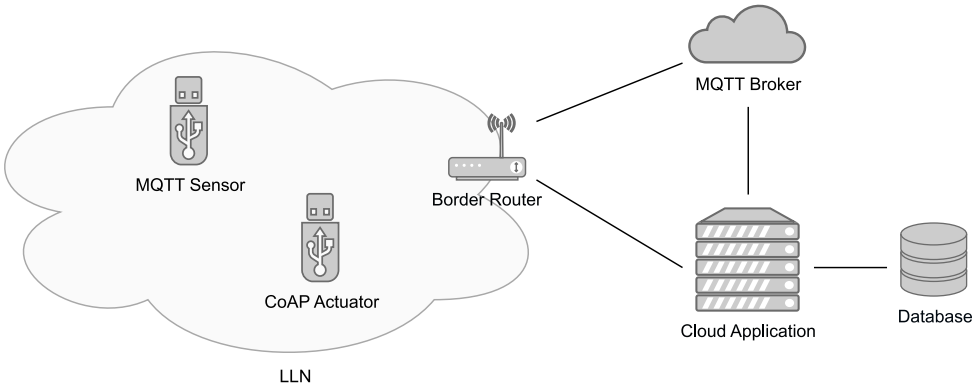

Figure 2: Cloud application interaction scheme.

As shown in the figure, all data collected by the cloud application are organized and store in a database.

2.2 Remote Control Application

The remote control application is used by the beekeepers to manage their apiaries. In particular, it exploits the data created by the cloud application and the **rules** created by the beekeeper to implement the **control logic**. The logic consists of **periodic polling** to check the status of the hive and apply rules if necessary. In this regard, the beekeeper must

set the frequency with which this check is to take place. The remote control application acts as **CoAP client** in order to communicate with the actuator nodes.

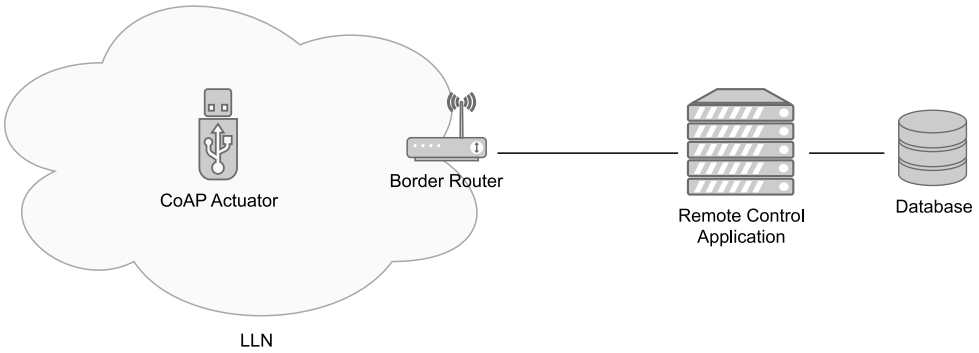

Figure 3: Remote control application interaction scheme.

The beekeeper interacts with the remote control application using a command line interface.

2.3 MQTT Sensor

The MQTT sensor is a node that periodically collects several values from the surrounding environment. These data are encapsulated in an **MQTT packet** and transmitted to the cloud application through the **broker**. The period with which the sensor samples data from the environment can be changed through a press (of a duration of less than 3 seconds) of the button on the sensor. The behavior of the sensor node is described by the following scheme:

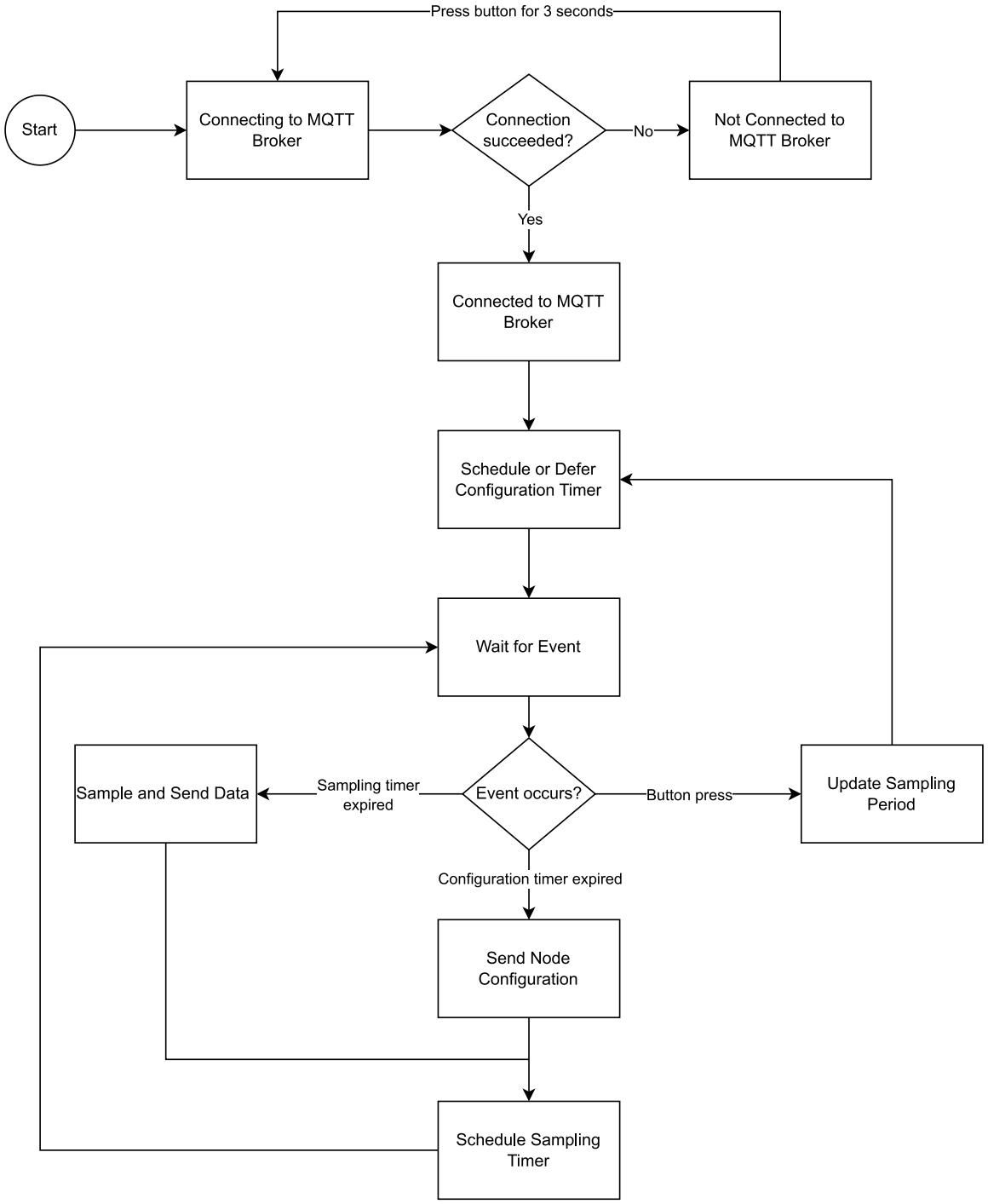

Figure 4: MQTT sensor state machine.

The nodes goes through 3 states which are represented by a LED. If the LED shows a steady light, the node is **disconnected** from the broker. Instead, if the LED is blinking, the node is **trying to connect** to the broker. Finally, if the LED is off, the node is **connected** to the broker. If the node is disconnected, it is possible to retry a new connection holding down the button for more than 3 seconds.

2.4 CoAP Actuator

The CoAP actuator is a node that executes commands received from the remote control application. These commands are sent through a **CoAP request**. The behavior of the actuator node is described by the following scheme:

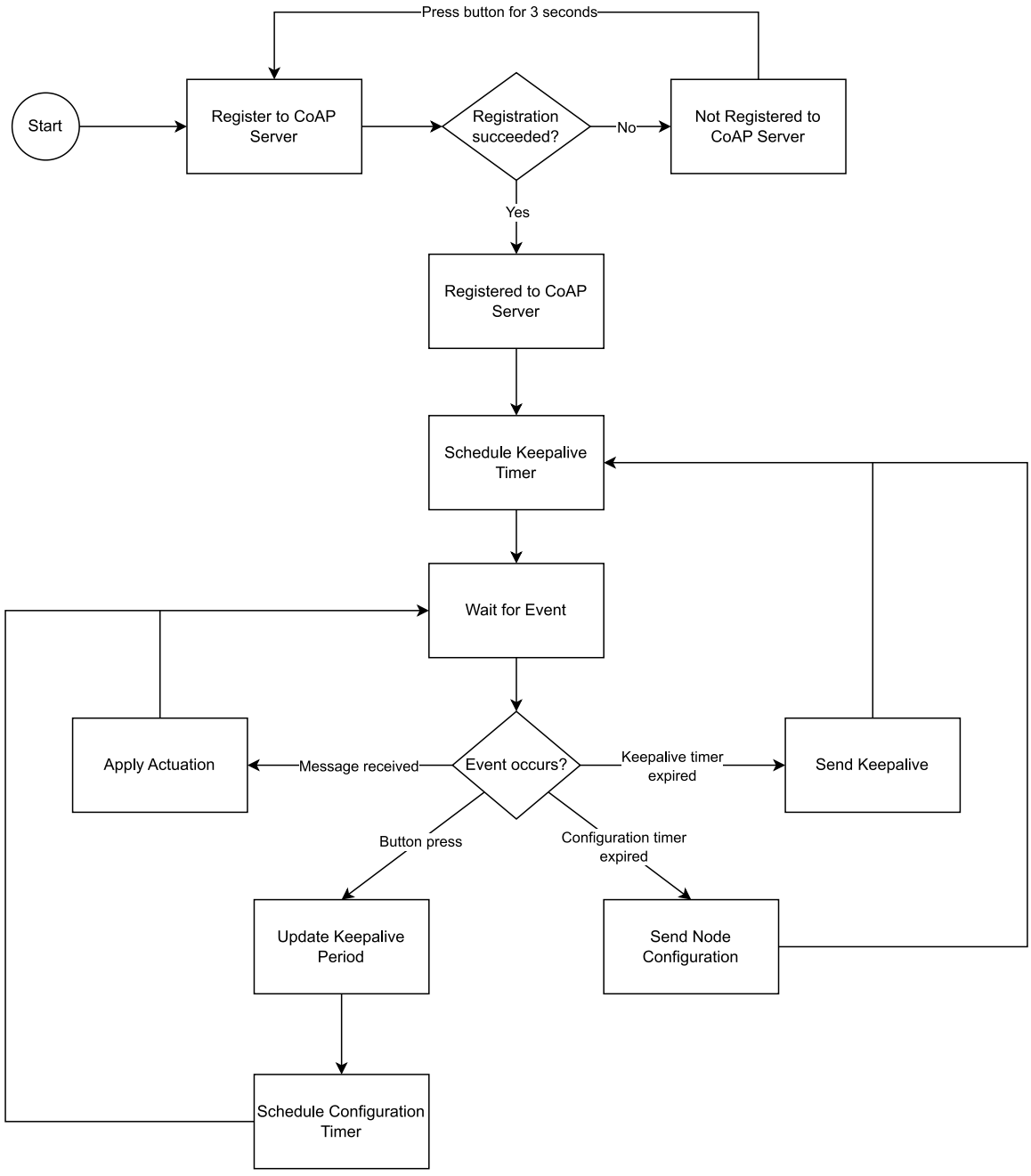

Figure 5: CoAP actuator state machine.

Also this node goes through 3 states which are represented by a LED. If the LED shows a steady light, the node is **not register** to the cloud application. Instead, if the LED is blinking, the node is **trying to register** to the application. Finally, if the LED is off, the node is **registered** to application. If the node is not registered, it is possible to retry a new registration holding down the button for more than 3 seconds. In order to track the actuator node state, a **keepalive mechanism** is used. In particular, the actuator node sends a keepalive to the cloud application with a period that can be configured using the button.

3 Implementation

In this section will be analyzed some implementation details about the software components of the system.

3.1 Cloud Application

The cloud application is written in *Python* and implements the MQTT and CoAP communication using respectively the *Paho* and *Coapthon* libraries. Specifically, the application consists of two threads:

- **CoAP manager**, which exposes the *registration* and *keepalive* resources to the CoAP actuator
- **MQTT manager**, which subscribes itself to the topics related to the configuration and measurements of the different sensor nodes

The states of the actuators and sensor nodes along with the received measurements are stored by the cloud application in a *MySQL* database.

3.2 Remote Control Application

The remote control application is written in *Python* and communicates with the CoAP actuators using the *Coapthon* library. Specifically, the application consists of 3 threads:

- **CLI manager**, which implements the command line interface and manages the commands issued by the beekeeper
- **Event manager**, which implements the control logic and generates the control messages for the actuator nodes
- **CoAP client**, which sends the control messages produced by the event manager

The event manager handles an **events queue** in order to implement the periodic polling described in Section 2.2. Specifically, an event consists of the periodic evaluation of hive rules. The manager creates an **event timeline**, and between consecutive events **stops** its execution.

Regarding rules, an example of **heating system** rule is the following:

```
if hive temperature < 25:
    set target temperature = 30
```

Instead, an example of **ventilation system** rule is:

```
if hive temperature > 30 and/or hive humidity > 70 and/or hive co2 > 600:
    set ventilation level = high
```

3.3 MQTT Sensor Node

The MQTT node is programmed using *Contiki-NG*, an operative system developed for constrained devices. This OS allows to use the MQTT protocol on top the *802.15.4* protocol stack.

3.3.1 Message Types

Each sensor node sends **configuration** and **measurement** messages. A configuration message is used to notify the application of a node's existence and to update its status. The first field contained is the *node id* (i), which uniquely identifies the node in the application and derives from its MAC address. The second field is the *sensor node type* (t), which specifies the type of measurement provided by the sensor. The possible types are *thc* (temperature, humidity and co2), *fn* (frequency and noise), c (counter) and w (weight). Finally, the last field is the *sampling time* (s), which indicates the period with which the sensor node sends the sampled data. This information is exploited to infer if the node is still alive.

Instead, the measurement message contains the data collected by the sensor. Each measurement message contains specific information based on the type of sensor that generated the measurement. The basic structure of all messages includes the *node id* (i) used to identify the source of the measurement. For the *thc* sensor node, the message includes the t field, which contains the temperature, the h field, which contains the humidity and the c field, which contains the ratio of carbon dioxide to oxygen. For the *fn* sensor node, the message includes the f field, which contains the frequency and the n fields, which contains the noise. Then, the c sensor node includes the in field, which contains the number of bees that entered the hive and the o fields, which contains the number of bees that came out of the hive. Finally, the w sensor node includes the w field, which contains the weight.

3.3.2 MQTT Topics

Since the application uses the MQTT protocol, every message types is associated with a **topic**. For the configuration message, all sensor nodes use the same topic that is *configuration*. Regarding to measurement messages, each sensor node uses a different topic that represents the type of measurements. The possible topics are:

- **temperature_humidity_co2**
- **frequency_noise**
- **counter**
- **weight**

3.3.3 Data Encoding

The data to be sent are simple and don't require a complex structure and don't have to be validated. So, the *JSON encoding language* has been used, in order to have a compact and lightweight representation of the data.

An example of configuration message is shown below:

```
{
    "i": "200020002000",
    "t": "thc",
    "s": 60
}
```

3.3.4 Other details

As said in Section 2.3, by pressing the button a different sampling time can be selected. To notify the changes to the application, a configuration message is sent. Specifically, to **avoid spam** of configuration messages, the transmission is scheduled 10 seconds after the selection of the sampling time through a **timer**. In this way, each press of the button in this interval will **defer** the timer instead of sending a new message.

3.4 CoAP Actuator

Even the CoAP actuator is programmed using *Contiki-NG*. Specifically, the CoAP implementation is exploited for the communication between actuators and the application.

3.4.1 Message Types

Each actuator sends both **registration** and **keepalive** messages. A registration message is used to register the actuator along its information to the application. The first field of the message is the *node id* (i), which derives from the node MAC address as for MQTT nodes. The second field is the *IPv6 address* of the node (ip), which is used by the remote control application to contact the CoAP node. Then, the third field is the *actuator type* (t), that can be *ta* (temperature actuator) and *va* (ventilation actuator). Finally, the last field is the *keepalive period* (k), which indicates the period with which the actuator node will send the keepalive message.

Instead, the keepalive message is used by both the actuator and the application to track the connection state. It is a simple message that contains only the *node id* (i) field.

3.4.2 CoAP Resource

In addition to operating as a CoAP client, the actuator is also a CoAP server and as such exposes a **resource** based on the actuator type (*/temperature* for the heating system, */ventilation* for the ventilation system). This, through a PUT method, allows the state of the actuator to be controlled. In the case of the heating system, the only parameter required to activate the actuator is the temperature to be achieved, represented with the t field of the payload. Instead, in the case of the ventilation system, the parameter needed is the ventilation mode, which can be off, low, medium or high, represented with the v field of the payload.

3.4.3 Data Encoding

As for MQTT, the messages have a simple structure and don't have to be validated. So even in this case, the *JSON encoding language* has been used.

An example of registration message is shown below:

```
{
    "i": "200020002000",
    "ip": "fd00::202:2:2:2",
    "t": "ta",
```

```
    "k": 60
}
```

3.4.4 Other details

The actuator doesn't implement any logic, but the actuation **totally depends** on the remote control application. So, if the connection between the actuator and application is lost the node **stops** the actuation. In this way, the remote control application can always predict the state of the actuation without asking the node for its status.

3.5 RPL Border Router

In order to allow sensors and actuators from the low power and lossy network to communicate with the applications, an RPL border router is needed. For this purpose, the *Contiki-NG* implementation has been used.

4 Deployment

The system has been tested on the **Cooja** simulation environment and then deployed on **nRF52840 dongles**. Specifically, 4 dongles have been used as sensor nodes, 1 dongle as RPL border router and 2 dongles as actuator nodes. The data recorded by the sensors have been randomly produced.

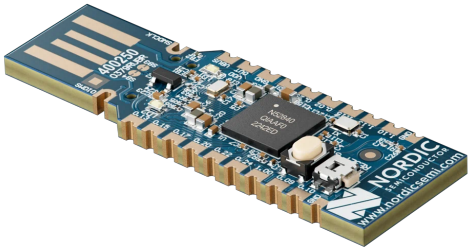

Figure 6: nRF52840 dongle.

In order to visualize the system status and the measurements sampled by the sensors, a web-based interface has been implemented using **Grafana**. In order to see the information associated with an hive, it is necessary to select an area name and an hive id from the dashboard. The dashboard shows the last measured values from all the sensors and the connectivity state of all hive nodes. An example of dashboard is shown below:

Figure 7: A preview of the Grafana dashboard.