Smart Fish Farm

Internet of Things

Matteo Manni

https://github.com/MatteoManni99/UNI_IOT_SmartFishFarm

Index

1 Introduction

The aquaculture industry is instrumental in addressing the increasing worldwide seafood demand, which greatly impacts human nutrition and the global economy. However, this expansion has posed notable challenges, such as operational efficiency, responsible resource management, and environmental implications. In response to these challenges, the Smart Fish Farm project seeks to create a solution by integrating IoT technologies to improve the efficiency and profitability of aquaculture operations.

Within this context, maintaining precise control over water temperature within a specific range is crucial for successful aquaculture operations. Additionally, ensuring that the quality of water remains above a minimum threshold is essential to prevent adverse effects on the fish being cultivated. For simplicity, we have grouped in a single measurement all parameters which regard water quality, encompassing values as pH, turbidity, and others.

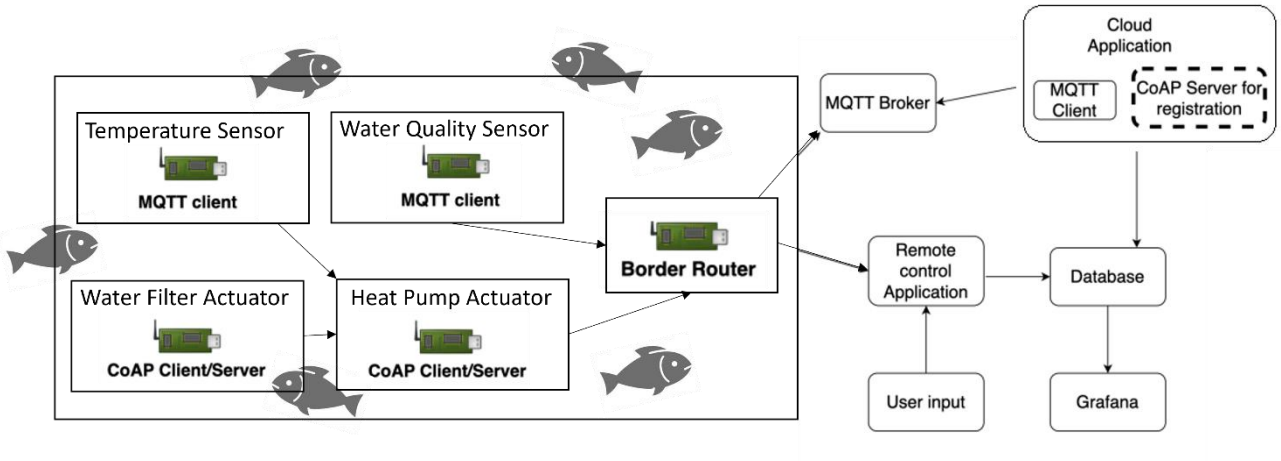
Therefore, our implementation is focused on sensing water parameters through sensor nodes that are interconnected in an IoT network. Then through a java application, that collect measurements, we can take decision and acting on problems that could occur.

1.1 Notes

Many things have been simplified and not all possible errors have been managed, for simplicity we assumed that more or less everything went right, as this is a project for educational purposes to test technologies in the IOT field.

2 Architecture

The complete scheme is shown below:



All the devices we used are nRF52840-Dongles, which we used to simulate sensor functionalities, actuators, and a RPL border router.

In particular, we have developed 5 different sensors:

- A Temperature Sensor
- A Water Quality Sensor
- Heat Pump – An Actuator to cool or heat the water.
- Water Filter – An Actuator to clean and regulate water.
- RPL Border Router

2.1 Sensor Implementation

The Sensors make use of the MQTT protocol for sending data to the controller via the MQTT Broker. The Temperature Sensor transmits messages using the topic '*temperature'*, while the Water Quality Sensor use the '*quality'* topic for communication.

They also subscribe to a topic in order to receive messages from the Application. In particular, the possibility of changing the sampling rate has been implemented. They subscribe respectively to the topics '*temp_sampling*' and '*quality_sampling*'.

Mqtt Sensors were developed as state machines. They pass from a first initialization state to the last operational state which is the one reached after having established the connection with the border router and having signed up to the topic. The temperature sensor LED is initially pink, and the water quality sensor LED starts yellow, then both turn green in the "subscribed" state.

Sensing of both water properties was randomly simulated with a function.

2.2 Actuator Implementation

The Actuators employ the CoAP protocol for registering to the remote application and then receive remote requests from the Application. They first work as CoAP clients to register to the application, sending a post request, this way the application keeps track of the actuators and their addresses with a database.

The actuators also act as CoAP servers, so requests can be made.

The Heat Pump Actuator manages put requests and get requests. The exposed resource can be in the "*OFF*", "*ON_HOT*" or "*ON_COLD*" state causing the green, red and blue LEDs to light up respectively. Obviously the put request is used to change the state and the get is used to acquire the current state.

The Water Filter Actuator manages put requests and get requests. The exposed resource can be in the "OFF" or "ON" state causing the green, red LEDs to light up respectively. Obviously the put request is used to change the state and the get is used to acquire the current state. A button has also been implemented in this actuator which is used to switch from off to on mode and vice versa.

2.3 RPL Border Router

For the border router we used the basic one provided by contiki-ng, with the only modification of a blue LED that lights up at startup.

2.4 Remote Control Application

This is a Java application for controlling the IOT network, a "Controller" class has been developed where the main is present and a command line interface has been developed for the user, the rest of the code has been mainly divided between 2 packages : cloudApplication and database. To implement the CoAP and Mqtt functionality in java two different libraries were used, *Californium* and *Paho* respectively.

2.4.1 Controller

In the Controller class we have the management of the Command Line Interface:

```
Command List:
0 X     -> set the temperature sampling period to X sec
1 X     -> set the water quality period to X sec
2 X Y   -> change the heat pump X to mode Y
3       -> switch temperature autopilot on/off
4 X Y   -> change the water filter X to mode Y
quit    -> shutdown the application
help    -> show the command list
```

You can see the autopilot mode that can be activated for which the application enters an automatic mode for managing the water temperature. If the temperature remains between two pre-established thresholds, then it keeps the heat pump off, otherwise it turns it on either in hot or cold mode.

The CoAP server is also started, and we subscribe to the topics on which the sensors publish the measurements.

2.4.2 cloudApplication

This package consists of 4 classes.

- *COAPclient* where the methods for sending requests to the actuators are defined
- *COAPserver* and *CoapResRegistration* where there is the code that manages the requests of the actuators asking to register, only the POST handler is defined
- *MQTTclient* where the methods for receiving Coap messages are present.

2.4.3 database

The whole application saves data on a SQL relational database, specifically in 3 different tables:

```sql
CREATE TABLE `actuator` (
  `idactuator` int(11) NOT NULL,
  `type` varchar(20) NOT NULL,
  `address` varchar(50) NOT NULL,
  PRIMARY KEY (`idactuator`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```sql
CREATE TABLE `sensor_quality` (
  `idsensor` int(11) NOT NULL,
  `date` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `quality` int(11) NOT NULL,
  PRIMARY KEY (`idsensor`,`date`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```sql
CREATE TABLE `sensor_temp` (
  `idsensor` int(11) NOT NULL,
  `date` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `temperature` float NOT NULL,
  PRIMARY KEY (`idsensor`,`date`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

In this package, methods for insertion and access to the database are defined, in particular they are organized in three different classes, one for each table: *ActuatorDA*, *QualityDA*, *TemperatureDA*.

2.5 Data Encoding

In our sensor network, where resources are limited, we've opted for JSON as our preferred data encoding format. All data generated by our sensors gets transmitted to the controller in the form of JSON objects. This choice was primarily influenced by the constrained nature of our sensor devices.

One significant advantage of JSON lies in its flexibility and simplicity, especially when compared to alternatives like XML. XML often comes with a more intricate and redundant structure, making it less suitable for our specific needs. By embracing JSON, we can simplify the data representation process, reducing unnecessary complexity and overhead.

```
{
"sensor_id": x
“quality”: y
}
```

```
{
"sensor_id": x
“temperature”: y
}
```

3 Grafana

We integrate to the application real-time monitoring and data visualization through Grafana dashboards. By utilizing Grafana, we've created a user-friendly interface that presents the data in a visually appealing and easily understandable manner. The dashboard can continuously update in real-time.

In this way the final user immediately sees at a glance if there have been any critical issues and if so, the user can take the necessary measures.