UNIVERSITÀ DI PISA

Department of Information Engineering
MSc Computer Engineering

Internet of Things

Dam Monitoring

Federico Cristofani

Academic Year 2022/2023

Contents

1. Introduction

1.1 Overview

A dam is a man-made structural barrier built to store and control the flow of water in rivers. They play a key role in storing and capitalizing water, preventing fresh water from leaking into the seas. The reservoirs created by dams not only control floods but also provide water for several useful activities such as power generation, irrigation and industrial usage. Dams management can be very complex and involves many risks, related to the amount of water stored in the reservoir. Storing too much water can result in violent flooding, causing irreparable damage to people and buildings. Preventing the possibility of flooding by keeping an arbitrarily low amount of water in the reservoir can cause damage to human activities due to lack of water, especially during periods of drought. The task is usually carried out manually by human operators and is rarely assisted by information technology, leading to inaccurate and reactive decisions.

The introduction of an IoT based solution to automatize the management may guarantee several advantages:

- Real-time monitoring of water level and other related parameters.
- Pro-active actions based on accurate predictions enhanced by the data collected by the IoT system.
- Fine-grained control of the reservoir, releasing the minimum amount of water to keep the level below a safe threshold while ensuring water supply for human activities.
- Possibility to spread information about dam level among nearby people, giving them access to the collected data.

1.2 Specification

The goal of the project is to develop an IoT telemetry and control system, consisting of a network of IoT devices and server-side components to collect data and implement control logic.

The following requirements have been provided:

- The sensors must publish data adopting MQTT protocol.
- Actuators must use CoAP and must act as servers to expose their resources.
- A cloud application must collect data from MQTT sensors and store them in a MySQL database.
- The actuators must register to the cloud application, acting as a CoAP client in order to create a directory in the database.
- A border router must be deployed in the network in order to provide external access.
- A remote control application reads information about actuators and sensors from the database and implements a simple control logic in order to apply some modifications to one or more actuators based on the data collected from the sensors.
- It is required to provide some user input to implement the user logic for the IoT application.
- A web-based interface deployed using Grafana must be developed in order to show the data collected and stored on the database.

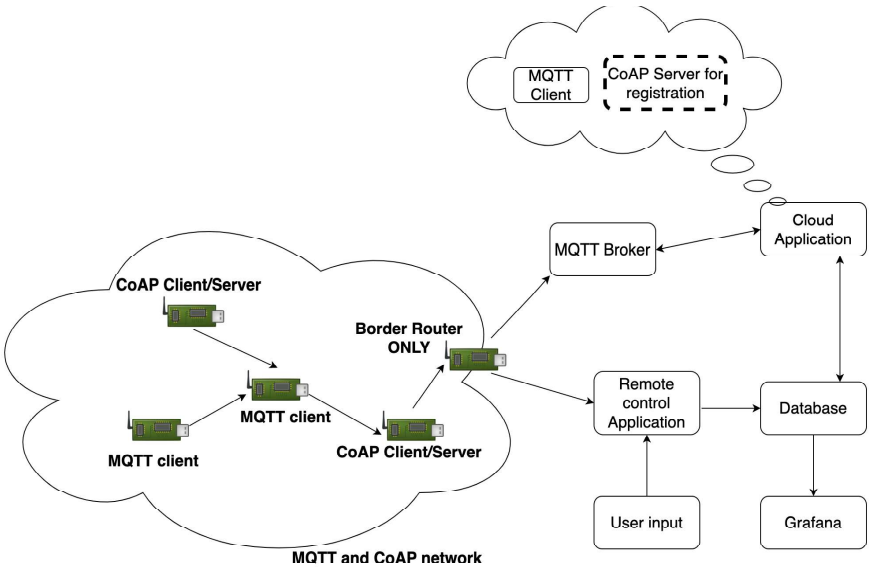

Figure 1.1: Reference architecture

1.3 Use cases

The features that must be taken into account to develop the described system are countless, however a real implementation is beyond the scope of the project. In the following are described the considered use cases:

- **Real-Time Monitoring**: The sensors periodically collect data regarding water level, inflow rate, and outflow rate. Operators can monitor such data both from remote control application and web dashboard.
- **Remote Control**: The actuators allow to remotely control and retrieve the current status of gates and alarm.
  - **Gate**: The gate controllers accept commands from operators to control the opening level of the gates, which directly affects the amount of water flowing through the outflow channels.
  - **Alarm**: The alarm controller accept remote commands to turn on or off alarms in case of water level above SAFE threshold. The alarm is always controlled by automatic control logic.
- **Operative Mode Switching**: Operators can switch between AUTO and MANUAL operative modes for gates control. In AUTO mode, the control logic automatically adjusts the gate opening, while in MANUAL mode is up to operators to set desired opening levels.
- **Automatic Control Logic**: The remote control application includes automatic control logic that set gate opening levels in order to keep the water level between MAX and MIN thresholds. If the water level approaches the SAFE threshold alarm is immediately triggered.
- **Configurable Thresholds**: The system allows operators to configure the MAX, MIN, and SAFE thresholds for water level control.
- **Automatic Reconnection of Sensors**: If a sensor is disconnected from the broker an automatic reconnection mechanism is implemented.
- **Status Indication with LEDs**: Sensors and actuators are equipped with LEDs that provide an indication of their status.

2. Motes

2.1 Sensors

The system requires two types of sensors to collect the data needed to monitor the dam, publishing the information via MQTT connection on a predefined topic.

- **Water-level sensor**, is responsible to publish periodically the level of water stored in the reservoir on topic "water-level".
- **Flow sensor**, is responsible to publish periodically the flow-rate through the associated channel on topic "flow-rate".

In principle, multiple instances of the water level sensor can be installed, for example, to improve robustness through redundancy, however the system works perfectly with a single instance. The matter is different with regard to the flow sensor because at least one sensor per channel must be installed in order to retrieve the metric. This leads to the necessity to configure each sensor with an identifier of the sensed channel, in such a way that the cloud application will be able to associate each measurement to the right channel. There are several approaches to configure the flow sensors, but to keep the design as simple as possible a static approach is chosen, i.e. a different firmware has been compiled for each channel. Obviously the code is the same, but some constants values assume different values based in the sensed channel. Below a snippet of code is reported in order to clarify how the code is organized to handle configurable parameters.

```
1 char* build_message(){
2   snprintf(record, APP_BUFFER_SIZE,
3               "{\"bu\":\"m3/s\", \"e\": [{\"n\":\"%s\", \"v\":%de-%d}]}",
                    STR(FLOW_NAME), read_flow_sensor(), SCALE_EXP);
4
5   return record;
6 }
```

Listing 2.1: Flow-sensor channel independent code

Automatic reconnection
Sensors establish a TCP connection with the MQTT broker to publish measurements, but sometimes that connection can break unexpectedly. In such situation, the sensor would no longer be able to publish new data, interrupting real-time monitoring and consequently the possibility of accurate remote monitoring. To overcome the issue, a simple automatic reconnection mechanism is implemented, with the goal of re-establishing the connection without requiring manual intervention by an operator. The mechanism is triggered whenever the sensor realizes that the broker is no longer reachable, so periodically tries to execute the steps that allows to establish a new connection. Often the disconnection is due to temporary adverse conditions, so a exponential increasing time, up to a maximum, is chosen to perform the periodically attempts. In this way, if the condition resolves quickly the reconnection is fast, but if the condition persists over time the sensor can save energy by increasing the time between attempts.

2.2 Actuators

The system requires two types of actuators to control the dam, namely to control the gate opening level and alarm. The actuators expose a CoAP interface that can be accessed to issue the commands to be

applied.

- **Gate controller**, is responsible to set and show the current opening level of the associated gate. Expose a single resources at endpoint "/actuator/gate/<channel-name>" that can be accessed via GET and PUT methods:
  - GET, retrieve the current gate opening level.
  - PUT, set the gate opening level.
- **Alarm controller**, is responsible to turn on/off and show the current status the alarm that warns about a risky water level of the reservoir. Expose a single resource that can be accessed via GET and PUT methods:
  - GET, retrieve the current alarm status.
  - PUT, set the alarm status.

The gate opening level is expressed in percentage (0 - 100), while the alarm status via a boolean variable (true/false). For the gate controller, the same considerations as for the flow sensor apply, i.e., an actuator is required for each gate. Again, the configuration is static and built at compile time. In the implementation gate controller and alarm controller are merged into a single firmware, assuming that for each gate an alarm actuator is installed. Finally is worth to mention that the gates are grouped in *primary* gates, the ones that control the outflow in normal conditions and *emergency* gates, the ones that allow to quickly drain the dam, jointly to the primaries, if safe threshold limit is exceeded.

Automatic registration
The actuators behaves as CoAP server exposing resources, but the remote control application need to know the URIs in which such resource can be accessed. A static configuration is not a viable solution because the URI is composed both from endpoint, defined by developer, but also from IP address, usually defined at run-time. Although the address can be inferred in advance, the static solution totally lacks flexibility and results in a tricky operation if many nodes are to be installed. The adopted solution allows the nodes to automatically connects to a remote well-known CoAP server, thus behaving as CoAP client, and sending a POST request containing all the required information to access the exposed resources. Only once the registration succeeded the nodes switch from behaving as CoAP client to behaving as CoAP server.

```
1  while(!registered){
2    coap_init_message(&request, COAP_TYPE_CON, COAP_POST, 0);
3    coap_set_header_uri_path(&request, REGISTRY_URI);
4
5    uip_ds6_addr_t* addr = uip_ds6_get_global(ADDR_PREFERRED);
6    uiplib_ipaddr_snprint(addr_buf, ADDR_SIZE, &addr->ipaddr);
7
8    snprintf(message, MSG_LEN, "[
9        {\"name\":\"outflow-%d\", \"type\":\"gate\",\"description\":\"Gate
             controller\",\"tag\":\"%s\",\"uri\":\"coap://[%s]/actuator/gate
             /outflow-%d\", \"value\":0}]",
10       FLOW, (FLOW == 1 ? "Primary":"Emergency"), addr_buf, FLOW);
11
12   coap_set_payload(&request, (uint8_t *)message, strlen(message));
13
14   COAP_BLOCKING_REQUEST(&server_ep, &request, handle_coap_response);
15 }
```

Listing 2.2: Automatic registration process

The required information to register a resource are:

- **Resource name**, name used by remote application to retrieve the URI, i.e. must be unique among all the registered resources.
- **Type**, type of resource, e.g. gate or alarm.

- **Description**, optional description of the exposed resource.
- **Tag**, additional information to classify a resoruce (e.g. primary/secondary gate).
- **Value**, value at registration time of the resource.
- **URI**, used to access the resource.

2.3 Data encoding and representation

Data encoding
The data produced by the nodes are encoded via JSON format in order to allow an easy interoperability across different applications. The reason JSON is chosen as encoding language stems from its high popularity, wide support on many platform (included Contiki-NG), and the lightweight nature of the format that limits the overhead in terms of processing and data exchange. Although it is a text-based encoding language, the additional overhead reduction that could be achieved by adopting a binary format is not worth it because of the lack of support due to the still insufficient popularity.

Data representation
The representation of the data produced by the nodes follows the SenML standard [3]. The choice is driven by desire to adopt a standard format instead of defining a customized approach that may compromise the interoperability of the system and by the fact that SenML is well integrated with JSON encoding language. The "name" attribute of each measure is a URN consisting of the MAC address of the device concatenated with a simple term. Some examples are reported:

```
{"bn":"urn:dev:mac:f4ce36a6d989ca07/", "bu":"m3/s", "e": [
    {"n":"inflow", "v":32139e-3}
]}

{"bn":"urn:dev:mac:f4ce3667db315386/","bu": "m", "e":[
    {"n":"water-level", "v":54703e-3}
]}
```

Listing 2.3: Sensors data format

```
{"n":"urn:dev:mac:f4ce364744271fe4/opening_level", "v": 90, "u":"percent"}

{"n":"urn:dev:mac:f4ce364744271fe4/state", "vb": false}
```

Listing 2.4: Actuator status format

3. Application

3.1 Collector

The cloud application is composed by two sub-component, each dedicated to a specific functionality:

- MQTT subscriber
- CoAP registration server

3.1.1 MQTT subscriber

The data published by the sensors of the network are collected by the MQTT subscriber that must subscribe to the topics associated with the sensors measurements, i.e. "flow-sensor" and "water-level". Once subscribed the workflow is pretty easy:

1. **Receive data**, each time the broker forwards data to the subscriber a callback function is triggered.
2. **Parse data**, the callback function is responsible to parse the data, i.e. extract the information structured in a JSON + SenML format.
3. **Store information**, the information are finally stored in the database.

Some aspects deserve to be highlighted:

- If the inter-arrival time between two messages of the same sensor[1]is less than a minimum time then the message is discarded.
- If the parsing of data fails because the format is not the one expected the message is discarded.

Database: sensor measurements
The part of relational database dedicated to store the sensor measurements is composed by two tables, one per sensor type, i.e. "flowRate" and "waterLevel".

FlowRate	
recordId	Auto-increment identifier
flowChannel	Measurement refereed channel
timestamp	Timestamp of measurement
value	Value of measurement

WaterLevel	
recordId	Auto-increment identifier
timestamp	Timestamp of measurement
value	Value of measurement

Table 3.1: Database sensor data tables

3.1.2 CoAP registration server

The automatic registration mechanism requires a remote CoAP server that accepts POST request and serve them storing the received information on database. The server is part of the cloud application and its workflow is straightforward:

- **Accept request**, the server expose "/registry" endpoint on which POST request must be directed.

[1]The identification is possible because of the URN embedded in the message.

- **Parse request payload**, the data associated to the request are parsed, extracting a list of resources each with its required information.
- **Store resource information**, the list of resources is finally stored into database.

Database: resource registry
The information needed to access the resources exposed by nodes are stored in "resource" table.

Resource	
Resource name	Resource identifier
Type	Type of resource
Description	Description of resource.
Tag	Additional information
Value	Current resource value
URI	Resource URI

Table 3.2: Database resource table

The value is initially set once the resource is created, and subsequent updating is up to the remote application that control the system's actuators.

3.2 Controller

Te remote control application is composed by two sub-systems, each dedicated to a specific functionality:

3.2.1 User interface

The operators can submits commands via a simple command line user interface. The available commands are listed below.

- **Mode** Mode configuration.
- **Threshold** Threshold levels configuration.
- **Flow** Show the current flow rates.
- **Level** Show the current water level.
- **Alarm** Show the current alarm status.
- **Gate** Control the gates.
- **Shutdown** Shutdown the application.
- **Help** Show available commands.

3.2.2 Control logic

The automatic control logic is enabled by the operators via "Mode" command. When disabled the gate control is left to operator, who can retrieve the current water level and flow rates via the appropriate commands set the gate opening accordingly. When the automatic control logic is enabled the gate control is inhibit for the operators and the following algorithm is applied in order to keep the water level at the desired level. The thresholds can be dynamically configured by the operator via "threshold" command

Algorithm 1 Automatic control logic

```
if waterLevel > safeThreshold then
    openAllGates()                                          ▷ Open all gates
else if waterLevel > maxThreshold then
    int step = inflowRate > outFlowRate ? LONG_STEP : STEP
    for primaryGate in primaryGates do
        increaseGateOpening(primaryGate, step)              ▷ Increase gate opening
    end for
else if waterLevel < minThreshold then
    int step = inflowRate < outFlowRate ? LONG_STEP : STEP
    for primaryGate in primaryGates do
        decreaseGateOpening(primaryGate, step)              ▷ Decrease gate opening
    end for
else
    int step
    if ABS(outFlowRate - inflowRate) > 20 then
        step = LONG_STEP
    else if ABS(outFlowRate - inflowRate) > 10 then
        step = STEP
    else
        step = SHORT_STEP
    end if
    for primaryGate in primaryGates do
        if outFlowRate < inflowRate - 1 then
            increaseGateOpening(primaryGate, step)          ▷ Increase gate opening
        else if outFlowRate > inflowRate + 1 then
            decreaseGateOpening(primaryGate, step)          ▷ Decrease gate opening
        end if
    end for
end if
```

3.3 Dashboard

The user interface provided by the remote control application gives operators a limited ability to visually monitor current data. Such functionality is offered via a web dashboard built via Grafana. The dashboard is composed by four elements:

- **Water level**, line chart showing the water level trend over the time.
- **Flow rate**, line chart showing the flow rate trend over the time for each channel.
- **Gates status**, bar gauge showing the opening level for each gate.
- **Alarm status**, simple text indicating the current status of the alarm ("Alarm on"/"Alarm off").

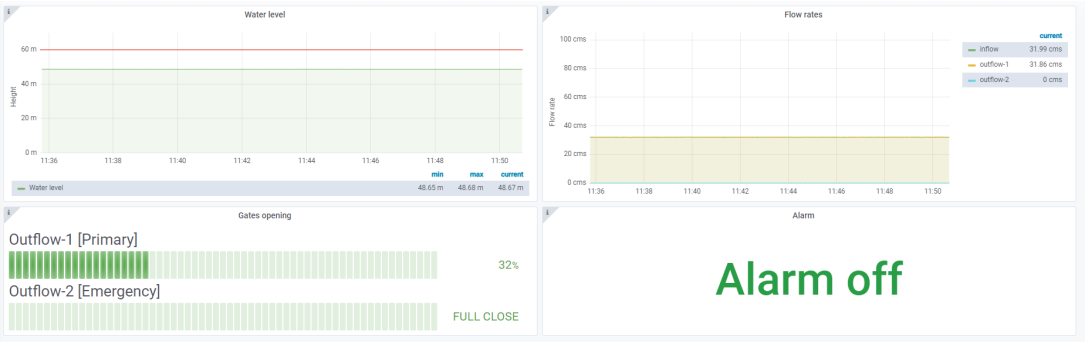

Figure 3.1: Dashboard snapshot

4. Demo

4.1 Firmwares

The implemented system was tested by building a network consisting of three IoT devices, more specifically NRF52840 dongle devices. Since the available devices are less than the number of sensor/actuator types, some firmware modifications were necessary. The final firmware uploaded on device are:

- **Mote-1**, implements border-router functionalities and gate/alarm-controller for primary and emergency gate (i.e. channel outflow-1 and outflow-2).
- **Mote-2**, implements flow-rate sensors for three channels (inflow, outflow-1 and outflow-2)
- **Mote-3**, implements only water-level sensor.

4.2 LED status

The LEDs on devices are used as output for a fast check on the device status:

- **Border router**, LED-2 is always on once prefix is set [1].
- **Gate controller**, LED GREEN/BLUE blinks three time each time the gate opening level is modified
- **Alarm controller**, LED RED blinks periodically when the alarm is on.
- **Sensors**, LED GREEN blinks once when a measurement is published and LED RED blinks periodically each time connection to broker is lost.

4.3 Simulation

The data to be retrieved from the sensors are randomly generated according to a simple logic. The flow rate depends on the opening level of the outflow channels, while the reservoir level varies according to the difference between the total outflow and the inflow into the dam. To ensure consistency among simulated data on different devices, a fake "simulation" topic was introduced, to which devices can subscribe to receive notifications about external events. The initial water level and inflow rate are determined statically, but to ensure thorough testing of the implemented system, these quantities can be changed at run-time by pressing buttons on the sensors.

- **Flow-sensor**, pushing the button the inflow rate increase up to a maximum value, while keeping the button pressed set the inflow rate to zero.
- **Water-level-sensor**, pushing the button the water level is reset to the initial value, while keeping the button pressed the water level exceeds the safe threshold.

[1] Because the border router is deployed jointly with CoAP server the led turn on once the registration process succeeded.

5. Conclusions

The system described is not intended to be used as is because many aspects are neglected, however aims to promote an approach that may improve the water management in the dam and improve the safety for people in nearby. Some choices are dictated by ease of design, such as the JSON language as encoding that could be replaced by binary encoding in the future, especially to reduce the amount of network traffic that in a real implementation could be considerably high. Finally, the implemented logic is very simple and unsuitable for the purpose of automatic control, moreover more parameter and advanced techniques should be taken into consideration . However, the structure of the application and firmwares may be a good starting point to cover the critical aspects and build a real functioning system.

Bibliography

[1] V.M.V.S Aditya et al. "IoT Based Water Level Monitoring System for Dams". In: *2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT)*. 2021, pp. 1–6. DOI: 10.1109/ICCCNT51525.2021.9579945.

[2] D. Dhinakaran et al. "Dam Management and Disaster Monitoring System using IoT". In: *2023 International Conference on Sustainable Computing and Data Communication Systems (ICSCDS)*. 2023, pp. 1197–1201. DOI: 10.1109/ICSCDS56580.2023.10105132.

[3] Cullen Fluffy Jennings et al. *Sensor Measurement Lists (SenML)*. RFC 8428. Aug. 2018. DOI: 10.17487/RFC8428. URL: https://www.rfc-editor.org/info/rfc8428.