الجـــمهوريـــــة الجزائريـــــة الديمــــقراطيـــــــة الشـعـبيـــة
وزارة التعليـــم العـــــــالي والبــــحث العلمـــي
People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research

Ecole Nationale Supérieure d'Informatique

Intelligent and Communicating Systems, ICS

2^{nd} Year Specialty SIL G1

Lab Report n°10

Title:

Iot System based Platform Broker MQTT - Node Red

Studied by:

First Name: Yasmine

Last Name: DINARI

E_mail: ly_dinari@esi.dz

Contents

Theory

1. Node-RED

Node-RED is a powerful visual programming tool designed for IoT applications that simplifies the process of connecting hardware devices, APIs, and online services. Developed by IBM, it provides a browser-based flow editor that makes it easy to wire together flows using a wide range of nodes.

Node-RED's key features include:

- Visual programming interface that allows for drag-and-drop development
- Built-in library of nodes for various protocols and services
- Real-time data processing and visualization capabilities
- Ability to create custom nodes and extend functionality
- Dashboard creation for monitoring IoT data

Compared to other IoT platforms like OpenHAB, Node-RED offers more flexibility in creating custom workflows and integrations. While OpenHAB is primarily designed for home automation with a focus on rules and events, Node-RED provides a more general-purpose approach to data flow programming that can be applied to various IoT scenarios.

2. MQTT Protocol

MQTT (Message Queuing Telemetry Transport) is a lightweight, publish-subscribe network protocol designed for constrained devices and low-bandwidth, high-latency networks. It was created in 1999 by engineers at IBM and EuroTech, primarily for machine-to-machine (M2M) communication.

2.1. Key Characteristics

MQTT has become the de facto standard for IoT messaging due to several advantages:

- Lightweight and efficient, requiring minimal resources
- About 90 times faster than HTTP for sending information
- Approximately 10 times less energy-consuming for sending messages

- Around 170 times less energy-consuming for receiving messages
- Scalable to millions of connected devices
- Supports bidirectional communication between devices and cloud

2.2. MQTT Architecture

The MQTT protocol operates on a publish/subscribe model that decouples the sender (publisher) from the receiver (subscriber). This architecture consists of three main components:

- **Publishers:** Devices or applications that send messages to specific topics
- **Subscribers:** Devices or applications that receive messages from topics they have subscribed to
- **Broker:** Central server that receives all messages from publishers and routes them to the appropriate subscribers

2.3. Quality of Service

MQTT provides three levels of Quality of Service (QoS) that define the delivery guarantee of messages:

- **QoS 0 (At most once):** Messages are delivered according to best efforts of the underlying TCP/IP network. Message loss can occur.
- **QoS 1 (At least once):** Messages are guaranteed to be delivered at least once, but duplicates may occur.
- **QoS 2 (Exactly once):** Messages are guaranteed to be delivered exactly once, with no duplicates.

2.4. MQTT Brokers

Several MQTT brokers are available for implementing MQTT communication:

- **Mosquitto:** Most widely used open-source broker, lightweight and suitable for deployment on devices like Raspberry Pi
- **ActiveMQ:** Enterprise-grade message broker that supports multiple protocols
- **RabbitMQ:** Open-source with commercial version available
- **EMQTT:** Designed for large-scale deployments with up to 1 million connections

Activity

1. A. Node-Red Broker

In this activity, we'll implement an IoT system using Arduino, MQTT, and Node-RED to capture and visualize sensor data from force and light sensors.

1.1. Hardware Setup

First, gather these components:

- Arduino Uno board
- Force-sensitive resistor (FSR)
- Light-dependent resistor (LDR)
- Resistors (10kΩ for voltage dividers)
- Breadboard and jumper wires

Connect the sensors to the Arduino as shown in Figure 2.1:

- Force sensor to analog pin A1 through a voltage divider
- Light sensor (LDR) to analog pin A2 through a voltage divider

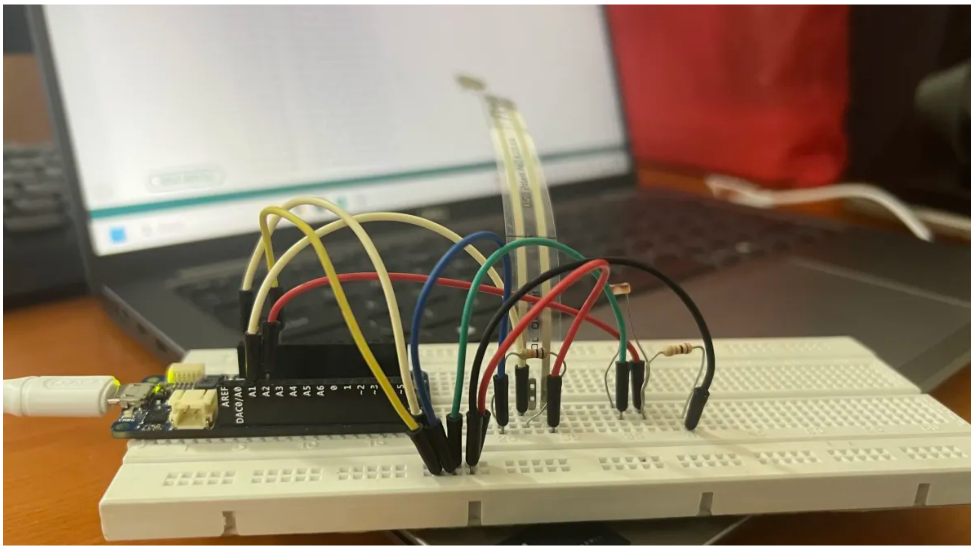

Figure 2.1: Arduino circuit with force and light sensors on breadboard

Figure 2.2 shows the schematic of our connections:

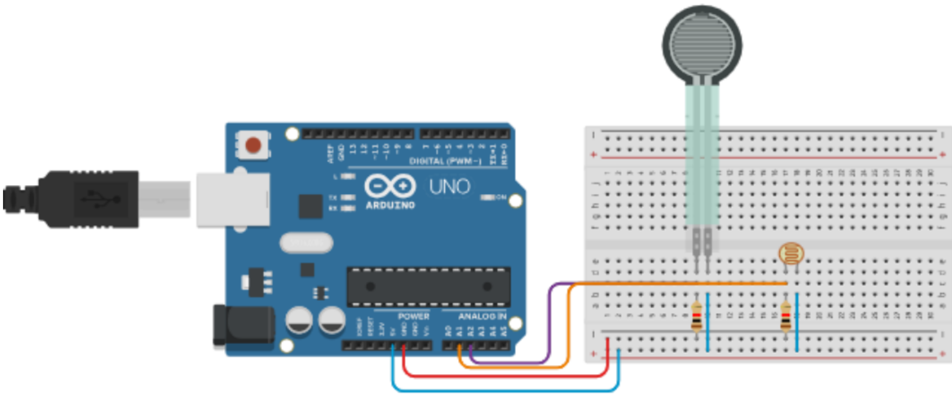

Figure 2.2: Schematic diagram of Arduino with force and light sensors

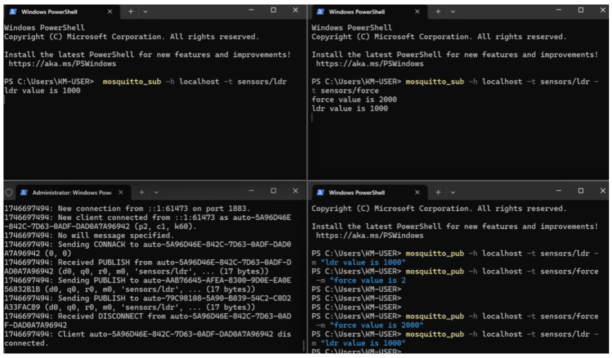

Figure 2.3

1.2. Arduino MQTT Client Implementation

Now, let's program the Arduino to read sensor values and publish them to MQTT:

```
#include            // For Wi-Fi connectivity
#include         // For MQTT functionality

// Wi-Fi credentials
char ssid[] = "Loading...";       // Wi-Fi name (SSID)
char pass[] = "helloworld";         // Wi-Fi password

// MQTT information
const char* mqttServer = "test.mosquitto.org"; // MQTT broker address
const int mqttPort = 1883;                     // MQTT port
const char* mqttTopicForce = "arduino/force";  // Topic for force sensor
const char* mqttTopicLDR = "arduino/ldr";      // Topic for light sensor

WiFiClient wifiClient;                         // Wi-Fi client
PubSubClient client(wifiClient);               // MQTT client

void setup() {
  Serial.begin(115200);                        // Initialize serial port
  WiFi.begin(ssid, pass);                      // Connect to Wi-Fi

  // Wait for Wi-Fi connection
  while (WiFi.status() != WL_CONNECTED) {
```

```
    delay(1000);
    Serial.println("Connecting to Wi-Fi...");
  }
  Serial.println("Wi-Fi connected!");

  // Set up MQTT client
  client.setServer(mqttServer, mqttPort);

  // Connect to MQTT broker
  while (!client.connected()) {
    Serial.println("Connecting to MQTT broker...");
    if (client.connect("ArduinoClient")) {
      Serial.println("Connected to broker!");
    } else {
      Serial.print("Failed to connect, rc=");
      Serial.print(client.state());
      delay(2000);
    }
  }
}

void loop() {
  // Read sensor data
  int forceSensorValue = analogRead(A1);  // Force sensor value
  int lightSensorValue = analogRead(A2);  // Light sensor value

  // Publish force sensor data
  char forceMessage[20];  // Buffer for message
  snprintf(forceMessage, sizeof(forceMessage), "%d", forceSensorValue);
  client.publish(mqttTopicForce, forceMessage);
  Serial.print("Force: ");
  Serial.println(forceMessage);

  // Publish light sensor data
  char lightMessage[20];  // Buffer for message
  snprintf(lightMessage, sizeof(lightMessage), "%d", lightSensorValue);
  client.publish(mqttTopicLDR, lightMessage);
  Serial.print("Light: ");
  Serial.println(lightMessage);

  delay(2000); // Wait 2 seconds before next reading
}
```

Listing 2.1: Arduino MQTT client code

1.3. Mosquitto Broker Setup

Next, install and configure the Mosquitto broker:

```
# On Windows: download and install from mosquitto.org/download/
# On Linux:
sudo apt update
sudo apt install -y mosquitto mosquitto-clients

# Configure Mosquitto (edit mosquitto.conf):
listener 1883
```



```
allow_anonymous true

# Restart Mosquitto service
sudo systemctl restart mosquitto
```

Listing 2.2: Mosquitto broker setup

Test the broker with these commands:

```
# In one terminal, subscribe to topics:
mosquitto_sub -h localhost -t arduino/force -t arduino/ldr

# In another terminal, publish test messages:
mosquitto_pub -h localhost -t arduino/force -m "100"
mosquitto_pub -h localhost -t arduino/ldr -m "500"
```

Listing 2.3: Testing Mosquitto

1.4. Node-RED Installation and Configuration

Install Node-RED and required packages:

```
# Install Node.js and npm first
# Then install Node-RED:
npm install -g --unsafe-perm node-red

# Start Node-RED:
node-red

# Install dashboard nodes:
cd ~/.node-red
npm install node-red-dashboard
```

Listing 2.4: Node-RED setup

Access Node-RED at http://localhost:1880 in your browser.

1.5. Node-RED Flow Implementation

Create a flow to visualize sensor data:

1. Drag two MQTT input nodes to subscribe to "arduino/force" and "arduino/ldr" 2. Add debug nodes to monitor incoming data 3. Add gauge nodes to visualize sensor values

Configure the dashboard:

The final dashboard should look like this:

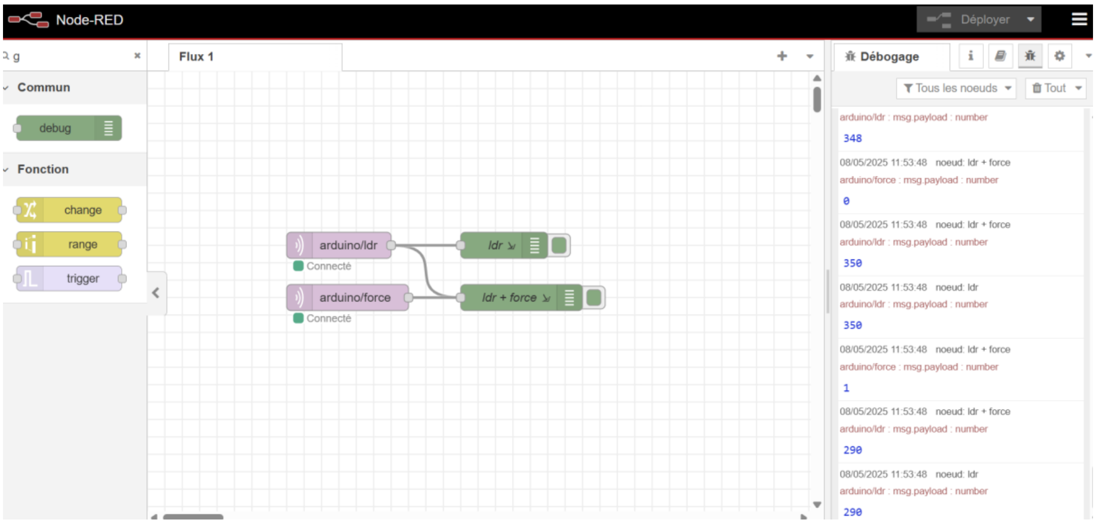

Figure 2.4: Node-RED flow with MQTT inputs and debug nodes

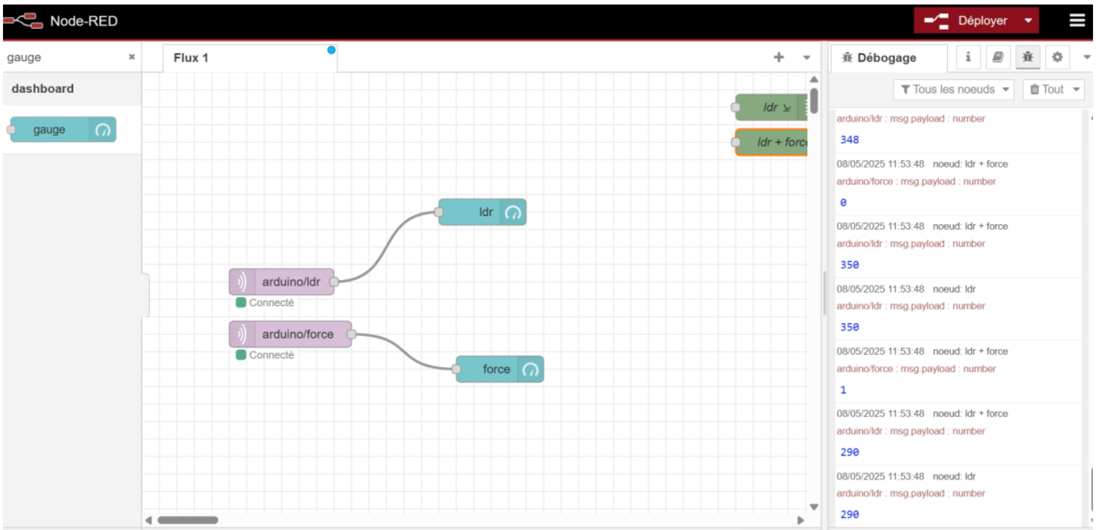

Figure 2.5: Node-RED dashboard configuration with gauge widgets

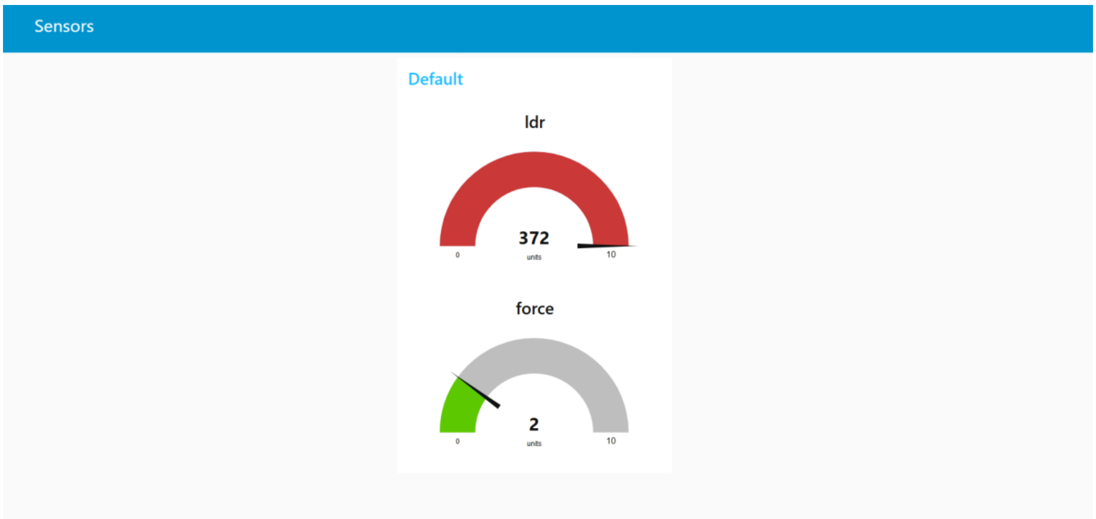

Figure 2.6: Node-RED dashboard showing sensor values in real-time

1.6. Testing the System

Test your system by:

- Applying different pressures to the force sensor
- Changing light conditions for the LDR
- Watching the dashboard update in real-time

2. B. Stand-Alone Platforms

For comparison, let's implement the same system using OpenHAB.

2.1. OpenHAB Installation

Install OpenHAB:

```
# On Windows: Download installer from openhab.org/download/
# On Linux:
wget -qO - 'https://openhab.jfrog.io/artifactory/api/gpg/key/public' | sudo apt-key add -
sudo apt-get install apt-transport-https
echo 'deb https://openhab.jfrog.io/artifactory/openhab-linuxpkg stable main' | sudo tee /etc/apt/sources.list.d/openhab.list
sudo apt-get update
sudo apt-get install openhab
sudo systemctl start openhab.service
```

Listing 2.5: OpenHAB installation

Access OpenHAB at http://localhost:8080.

2.2. OpenHAB Configuration

Follow these steps to configure OpenHAB:

1. Install the MQTT binding through Settings → Bindings

2. Create an MQTT broker thing: - Go to Settings → Things → Add → MQTT Binding → MQTT Broker - Configure: ID: localBroker, URL: tcp://localhost:1883

3. Create MQTT things for sensors: - Add Generic MQTT Things for force and light sensors - Configure channels with topics "arduino/force" and "arduino/ldr"

4. Create items for the sensors:

```
Number ForceSensor "Force Sensor [%d units]" {channel="mqtt:topic:
    localBroker:forceSensor:forceValue"}
Number LightSensor "Light Sensor [%d units]" {channel="mqtt:topic:
    localBroker:lightSensor:lightValue"}
```

Listing 2.6: OpenHAB items

5. Create a sitemap for visualization:

```
sitemap sensors label="Sensor Dashboard" {
    Frame label="Sensors" {
        Text item=ForceSensor
        Text item=LightSensor
        Chart item=ForceSensor period=h refresh=5000
        Chart item=LightSensor period=h refresh=5000
    }
}
```

Listing 2.7: OpenHAB sitemap

2.3. Comparison with Node-RED

Here's how OpenHAB compares to Node-RED:

Feature	**Node-RED**	**OpenHAB**
Ease of setup	Visual programming, intuitive	More configuration steps required
Flexibility	JavaScript functions	Rule-based approach
Dashboard	Drag-and-drop interface	Requires sitemap configuration
Resource usage	Lighter weight	Requires more memory
Learning curve	Quick to learn	Steeper learning curve

Table 2.1: Comparison between Node-RED and OpenHAB

Conclusion

In this lab work, an IoT system was successfully implemented. It collects data from force and light sensors using an Arduino board, transmits the data via the MQTT protocol, and visualizes it using Node-RED and OpenHAB.

MQTT proved to be well-suited for IoT due to its lightweight design, low energy consumption, and publish/subscribe architecture. Node-RED stood out for its ease of use and flexibility, while OpenHAB, though requiring more setup, offered a more structured and maintainable approach, especially for home automation.

Both platforms integrated smoothly with the Mosquitto MQTT broker, highlighting the importance of interoperability in IoT systems. Finally, using a Raspberry Pi demonstrated the benefits of edge computing by reducing cloud dependency and improving system responsiveness.