Internet of Things
Second cycle degree in Computer Science
University of Bologna

IoT-Based Indoor Air Quality Monitoring System

Keran Jegasothy - 1007457

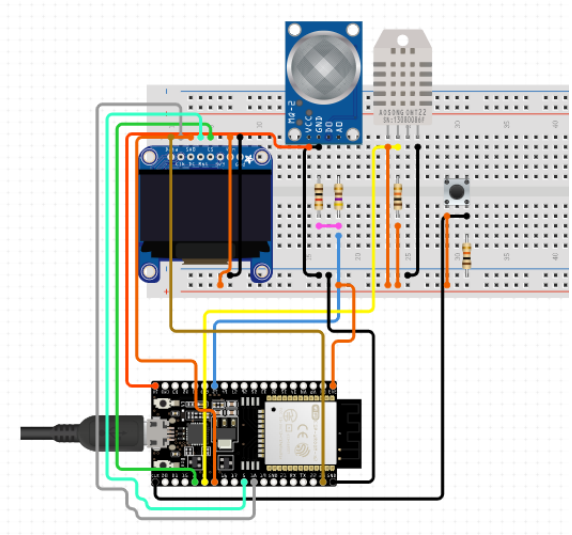
ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Contents

1 Introduction

The project consists in the deployment of an IoT application for smart home. The IoT device is formed by:

- ESP32 (microcontroller)
- DHT22 (temperature and humidity sensor)
- MQ-2 (gas sensor)
- Module with button
- Display OLED



The IoT device acquires data from sensors every SAMPLE_FREQUENCY seconds. By default SAMPLE_FREQUENCY is set to 5 seconds.
This device has two modality. In the first mode it uses the MQTT protocol, while in the second it uses the CoAP and HTTP protocols. To change modes, you have to press and hold the button.
The device transmits the following data:

- Temperature
- Humidity
- Gas concentration
- WiFi signal strength
- Air Quality Index

- GPS
- Device ID

All this data is shown on the OLED display. Furthermore, on this display it will also be possible to read the protocol active on the ESP32.

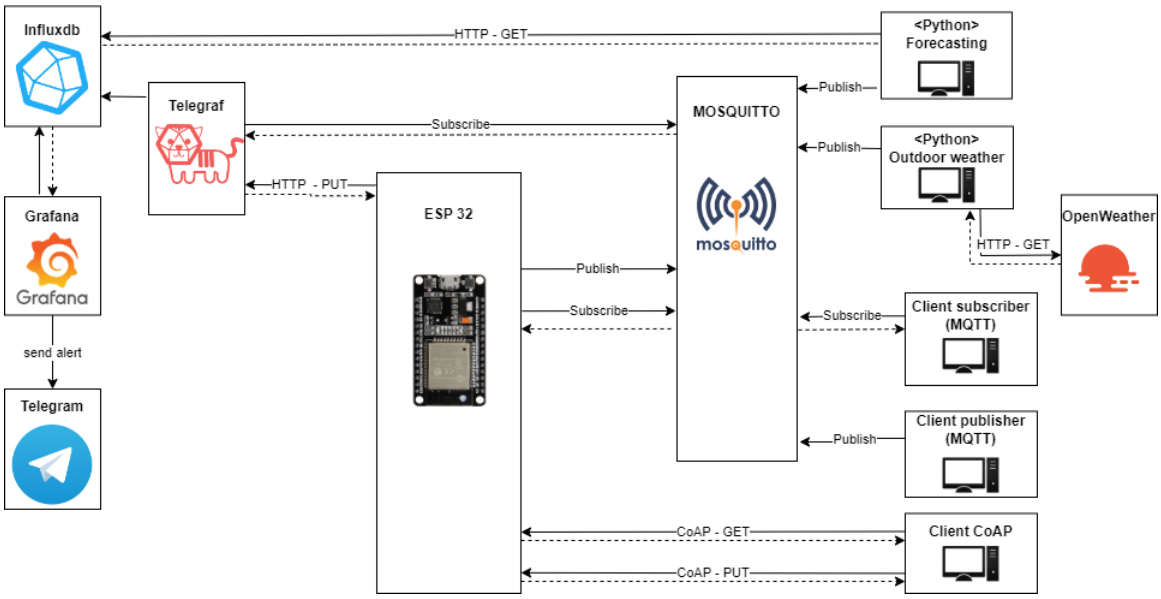
The Air Quality Index (AQI) is calculated as follows:

$$\text{AQI(x)} = \begin{cases} 0, avg \geq MAX\_GAS\_VALUE \\ 1, MIN\_GAS\_VALUE \leq avg < MAX\_GAS\_VALUE \\ 2, avg < MIN\_GAS\_VALUE \end{cases}$$

The client has the possibility to change the MIN_GAS_VALUE, MAX_GAS_VALUE and SAMPLE_FREQUENCY values using both the mqtt protocol and the CoAP protocol, based on the protocol active on the device. So, if the device is in the first mode, then the client will have to use the mqtt protocol to change the above values, otherwise it will have to use the CoAP protocol.
All data produced by the IoT device is saved in influxDB. Furthermore, the data relating to outdoor temperature, humidity and pressure are also saved in influxdb. All the saved data can be visualized, through graphs, in Grafana.Grafana, also sends notifications on telegram if the AQI is equal to 0 or 2 for at least 5 min.
Finally, there is a code that uses the ARIMA model to forecasting the value of temperature, humidity and gas concentration. This data is also saved in influxdb and displayed in grafana.

2 Project's Architecture

Below is an image with the architecture of the project.



The project can be divided into 3 parts:

- MQTT protocol
- CoAP and HTTP protocol
- Forecasting

2.1 MQTT protocol

The MQTT protocol requires a broker and mosquitto has been chosen. ESP32 performs both the role of publisher and subscriber. It plays the role of publisher in order to publish the various data (temperature, humidity, gas concentration, AQI, GPS and ID) obtained from the various sensors.The data has the Json format and the topic is "esp32". Also, it plays the role of subscriber when the client sends a message with topic "SAMPLE_FREQUENCY", "MIN_GAS_VALUE" or "MAX_GAS_VALUE" via the mqtt protocol. In this way the client has the possibility to change the values of these parameters.

The client also has the ability to play both the role of publisher and subscriber. When playing the role of publisher, he has the possibility to publish data with the following topics: "SAMPLE_FREQUENCY", "MIN_GAS_VALUE" or "MAX_GAS_VALUE". So, as already mentioned above, they are used to modify parameters in the ESP32. While, when it plays the role of subscriber, it has the

possibility to subscribe to the topic "esp32" and receive all the data produced by the IoT device.

In order to save all the data produced by the ESP32 in influxdb, Telegraf is used. Telegraf is an open source server agent that plays the role of subscribe and is subscribed to the topic "esp32". In this way, every time the EPS32 sends the data to the broker, they are forwarded to telegraf, and subsequently to influxdb.

Finally, there is also some Python code that uses the MQTT protocol. In particular there are two codes, the first foresees the future values of temperature, humidity and gas and publishes them with the following topics: "Temperature-Forecast", "HumidityForecast" and "GasForecast". While the second, makes a GET (http) request to OpenWeather in order to obtain the temperature, humidity and pressure of a city, and then publishes it with the following topics: "TEMPERATURE", "HUMIDITY" and "PRESSURE". Since Telegraf is subscribed to all these topics, all these data are saved in influxdb.

2.2 CoAP and HTTP protocol

The ESP32 has a mode, in which it uses the CoAP and HTTP protocols. When a client makes a GET request with CoAP, the ESP32 responds by sending it either all data or just the requested value based on the endpoint. Below are all the endPoints: "temperature", "humidity", "gas", "rssi", "aqi", "gps", "id", "allData".
In addition, the CoAP client has the possibility to make a PUT request in order to modify some parameters. These endPoints are reported below: "set/maxGasValue", "set/minGasValue", "set/sampleFrequency"

When the ESP32 is in this mode, every time it produces data, it sends them to telegraf via a PUT request using the HTTP protocol.

I decided to use the HTTP protocol directly to avoid using the CoAP protocol with the addition of a proxy, which would have slowed down the arrival of the packet.

2.3 Forecasting

The ARIMA model was used to predict future values for temperature, humidity and gas concentration. All the necessary data present in the database are given in input, furthermore it is necessary to give in input what is the sample_fraquency set in the IoT device and the field to predict between temperature, humidity and gas concentration.

3 Project's Implementation

3.1 ESP32

The code for the ESP32 was written using the Arduino IDE and with the following libraries: DHT, Wire, Adafruit_SSD1306, WiFi, PubSubClient, WiFiUdp, coap-simple, HTTPClient, ArduinoJson.

3.1.1 Air Quality Index (AQI)

```
int num_gas = 0;
bool check5GasValue = false;
float gas0=0;
float gas1=0;
float gas2=0;
float gas3=0;
float gas4=0;

void loop()
    ...
    if (num_gas == 0){
          gas0 = gassensorAnalog;
        } else if (num_gas == 1){
          gas1 = gassensorAnalog;
        } else if (num_gas == 2){
          gas2 = gassensorAnalog;
        } else if (num_gas == 3){
          gas3 = gassensorAnalog;
        } else if (num_gas == 4){
          gas4 = gassensorAnalog;
        }

        float avgGas;
        if (check5GasValue == false){
          if (num_gas > 3) {
            check5GasValue = true;
          }
          avgGas = (gas0 + gas1 + gas2 + gas3 + gas4)/ (num_gas+1);
        } else {
          avgGas = (gas0 + gas1 + gas2 + gas3 + gas4)/5;
        }

        int AQI;
        if (avgGas >= MAX_GAS_VALUE){
          AQI = 0;
        } else if ((avgGas >= MIN_GAS_VALUE) && (avgGas <
    MAX_GAS_VALUE)){
          AQI = 1;
        } else {
          AQI = 2;
        }

        num_gas = num_gas+1;
        if (num_gas == 5) {
          num_gas = 0;
        }
```

```
    ...
    }
}
```

Listing 1: ESP32 code - AQI

The code above allows to determine the AQI value taking into account the average of the last 5 values obtained by the MQ-2 sensor.

3.1.2 Change protocol

```
int led_state = LOW;
int button_state;
int last_button_state;

void loop() {
    ...
    if (last_button_state == HIGH && button_state == LOW) {
      Serial.println("Change protocol");
      led_state = !led_state;
      digitalWrite(LED_BUILTIN, led_state);
      delay(1000);
    }

     if (!led_state){
      // MQTT
    } else {
      // CoAP + HTTP
    }
    ...
}
```

Listing 2: ESP32 code - Change protocol

The above code allows you to change the protocol used by the ESP32 by keeping the button pressed. If the active protocol is the MQTT, then the led will be off, otherwise the active protocol will be Coap and HTTP and the led will be on.

3.1.3 MQTT

```
void connectmqttserver() {
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    if (client.connect("ESP32-01"), "username", "password") {
      Serial.print("connected");
      client.subscribe("MAX_GAS_VALUE");
      client.subscribe("MIN_GAS_VALUE");
      client.subscribe("SAMPLE_FREQUENCY");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      delay(5000);
    }
  }
```

```
}

void callback(char* topic, byte* message, unsigned int length) {
  Serial.print("Message arrived on topic: ");
  Serial.print(topic);
  Serial.print(".Message ");
  String messageTemp;

  for(int i=0; i<length; i++) {
    Serial.print((char)message[i]);
    messageTemp += (char)message[i];
  }

  if(String(topic) == "MAX_GAS_VALUE") {
    MAX_GAS_VALUE = messageTemp.toFloat();
  }
  if(String(topic) == "MIN_GAS_VALUE") {
    MIN_GAS_VALUE = messageTemp.toFloat();
  }
  if(String(topic) == "SAMPLE_FREQUENCY") {
    SAMPLE_FREQUENCY = messageTemp.toFloat();
  }
}

void loop() {
    ...
    if(!client.connected()){
        connectmqttserver();
      }
      client.loop();
      client.publish("esp32",DataAsJson);
    ...
}
```

Listing 3: ESP32 code - MQTT protocol

The code above can be divided into 3 parts:

- connectmqttserver: It allows the device to connect to the broker mosquitto and subscribe to three topics: "MAX_GAS_VALUE", "MIN_GAS_VALUE", "SAMPLE_FREQUENCY".
- callback: Allows the device to receive messages relating to the topics to which it is subscribed. Furthermore, based on the topic of the message, the relative parameter in ESP32 is changed.
- loop: It allows the device to publish data (which have the Json format) with the topic "esp32".

3.1.4 CoAP

```
// CoAP server endpoint url callback
void callback_temperature(CoapPacket &packet, IPAddress ip, int
    port);
...
```

```
void callback_allData(CoapPacket &packet, IPAddress ip, int port);

// CoAP server endpoint URL
void callback_temperature(CoapPacket &packet, IPAddress ip, int
    port) {
  Serial.println("[Temperature]");

  // send response
  char p[packet.payloadlen + 1];
  memcpy(p, packet.payload, packet.payloadlen);
  p[packet.payloadlen] = NULL;
  coap.sendResponse(ip, port, packet.messageid, tempConv);
  Serial.println(tempConv);
}

...

void callback_maxGasValue(CoapPacket &packet, IPAddress ip, int
    port) {
  Serial.println("[Max Gas Value]");
  // send response
  char p[packet.payloadlen + 1];
  memcpy(p, packet.payload, packet.payloadlen);
  p[packet.payloadlen] = NULL;
  String message(p);
  MAX_GAS_VALUE = message.toFloat();
  if (MAX_GAS_VALUE == message.toFloat()) {
    coap.sendResponse(ip, port, packet.messageid, "OK");
  } else {
    coap.sendResponse(ip, port, packet.messageid, "ERROR");
  }
}

void setup() {
    coap.server(callback_temperature, "temperature");
    ...
    coap.server(callback_maxGasValue, "set/maxGasValue");
    coap.start();
}

void loop)= {
    ...
    coap.loop();
    ...
}
```

Listing 4: ESP32 code - CoAP protocol

The code related to CoAP can also be divided into 3 parts:

- declaration of CoAP server endpoind URL callback (ex. *void callback_temperature (CoapPacket packet, IPAddress ip, int port)*;
- the actual CoAP server endpoind URL. In particular, there are two types of endpoints. The first responds to GET requests (eg. temperature), while the second responds to PUT requests (eg. set/maxGasValue). In the latter case, the present parameters of the device are modified. For example: if

the device receives a PUT request at the set/maxGasValue endpoint, then the MAX_GAS_VALUE parameter is updated with the value present in the payload of the message sent by the client.

- coap.loop: allows the device to listen and respond to any requests from the client.

3.1.5 HTTP

```
void loop() {
    ...
    HTTPClient http;
    http.begin("http://xxx.xxx.x.xxx:xxxx/getValues");
    http.addHeader("Content-Type", "application/json");
    int httpResponseCode = http.PUT(DataAsJson);
    if(httpResponseCode>0){
      String response = http.getString();
      Serial.println(httpResponseCode);
      Serial.println(response);
    }else{
      Serial.print("Error on sending PUT Request: ");
      Serial.println(httpResponseCode);
    }
    http.end();
}
```

Listing 5: ESP32 code - HTTP protocol

The above code allows the IoT device to periodically send the various data (with Json format) to Telegraf via the HTTP protocol.

3.2 Forecasting

```
query = ' from(bucket:"ProgettoIoT")\
    |> range(start: -10m)\
    |> filter(fn: (r) => r["ID"] == "ESP32Client-01")\
    |> filter(fn: (r) => r["_field"] == "TEMPERATURE")'
mqttTopic='TemperatureForecast'
result = query_api.query(org=org, query=query)
results = []
for table in result:
    for record in table.records:
        results.append(( record.get_value(), record.get_time()))

df = pd.DataFrame(data=results)

df[1]= pd.to_datetime(df[1], infer_datetime_format=True)
df = df.set_index(1, inplace=False)

train = df[0]

history = [x for x in train]
predictions = list()

```

```
client1 = paho.Client("control1")  # create client object
client1.on_publish = on_publish  # assign function to callback
client1.username_pw_set(username, password)
client1.connect(broker, port)  # establish connection

for t in range(num):
    model = ARIMA(history, order=(1,1,1))
    model_fit = model.fit()
    output = model_fit.forecast()
    yest = output[0]
    predictions.append(yest)
    print('predicted=%f' % yest)
    ret = client1.publish(mqttTopic, yest)
    time.sleep(int(delay))
```

Listing 6: Python code - Forecasting

The above code allows to obtain all the data relating to a field (in this case temperature). Subsequently these data are given as input to the ARIMA model, which will try to predict the next x values. Also, any predicted values will be sent to the mosquitto borker via mqtt. Since telegraf is subscribed to all topics used by the predictor, these data will be stored in influxdb.

3.3 Outdoor weather

```
api_key = "xxxxxxxxxxxxxxxxxx"
base_url = "http://api.openweathermap.org/data/2.5/weather?"
city_name = input("Enter city name : ")
a = 0

complete_url = base_url + "appid=" + api_key + "&q=" + city_name
response = requests.get(complete_url)
x = response.json()

if x["cod"] != "404":
    while a < 1:
        y = x["main"]
        current_temperature = y["temp"]
        current_temperature = round(current_temperature - 273.15,
    2);
        current_pressure = y["pressure"]
        current_humidity = y["humidity"]

        client1 = paho.Client("control1")  # create client object
        client1.on_publish = on_publish  # assign function to
    callback
        client1.username_pw_set("mqtt", "mqttpassword")
        client1.connect(broker, port)  # establish connection

        ret = client1.publish("TEMPERATURE", current_temperature)
        ret2 = client1.publish("HUMIDITY", current_humidity)
        ret3 = client1.publish("PRESSURE", current_pressure)
        time.sleep(5)
else:
```

```
28      print(" City Not Found ")
```

Listing 7: Python code - Outdoor Weather

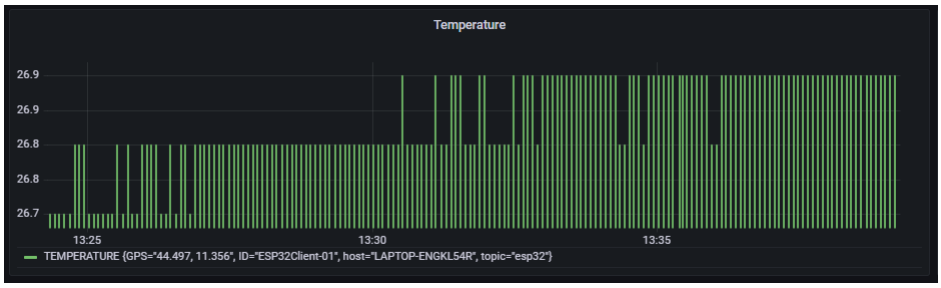
The code above makes a GET request to openWeather to get the temperature, humidity and pressure of a given city. Subsequently, this data is sent to the mosquitto broker via the mqtt protocol. Since telegraf is subscribed to the topics "current_temperature", "current_humidity", "current_pressure", these data are saved in influxdb.

3.4 Grafana

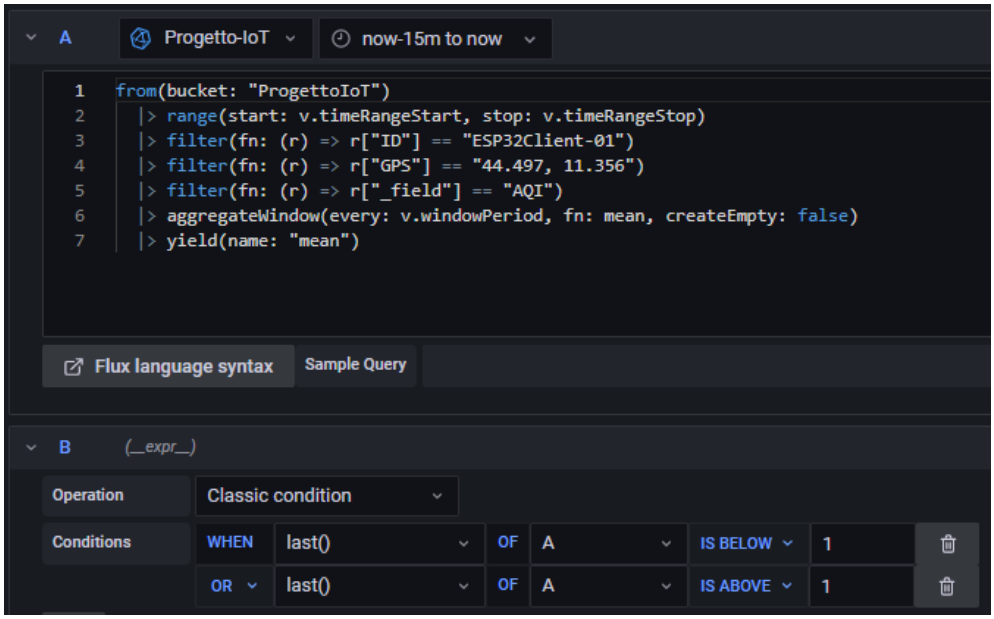
```
1 from(bucket: "ProgettoIoT")
2   |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
3   |> filter(fn: (r) => r["ID"] == "ESP32Client-01")
4   |> filter(fn: (r) => r["GPS"] == "44.497, 11.356")
5   |> filter(fn: (r) => r["_field"] == "TEMPERATURE")
6   |> aggregateWindow(every: v.windowPeriod, fn: mean, createEmpty:
      false)
7   |> yield(name: "mean")
```

Listing 8: Grafana code - Temperature

The above query allows grafana to get the data from influxdb. In this case, all temperature data are extracted. Subsequently, a graph is automatically created with this data, such as the graph below.



Below is how the alert was created and activated in case the AQI is equal to 0 or 2.
First, the query is created to get the AQI data. Then the condition is added:
WHEN last() OF A IS BELOW 1 OR last() OF A IS ABOVE 1
In order to send such alerts on telegram, a telegram bot has been created, which has been placed on a group. Subsequently, a contact point is created on grafana by setting the following parameters:
- Contact point type: Telegram
- BOT API Token: xxxxx
- Chat ID: xxx
Finally, through the "Notification policies" option, the alert was connected to the contact point created.



4 Results

4.1 Packege delay and delivery ratio

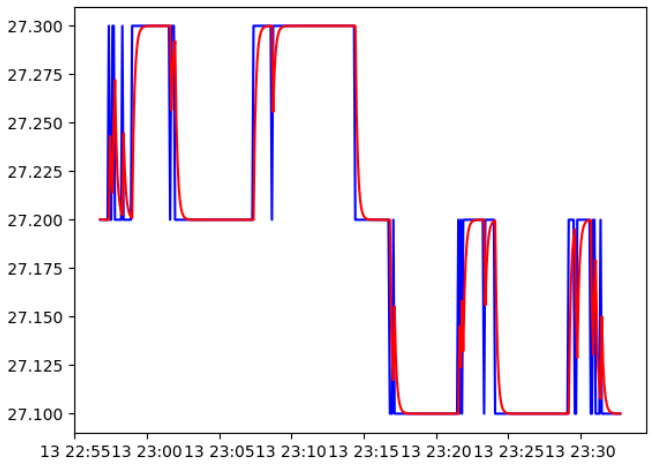
The following tables show the results obtained by sending 100 packets.

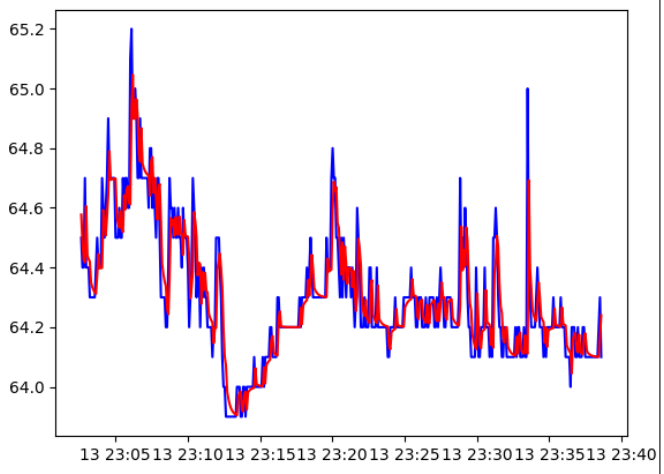
MQTT		
	average delay	packet delivery ratio
ESP32 →Mosquitto →Inluxdb	∼ 700 ms	100/100 = 1
ESP32 →Mosquitto →Client	∼ 50 ms	100/100 = 1
Client →Mosquitto →ESP32	∼ 9,6 s	100/100 = 1

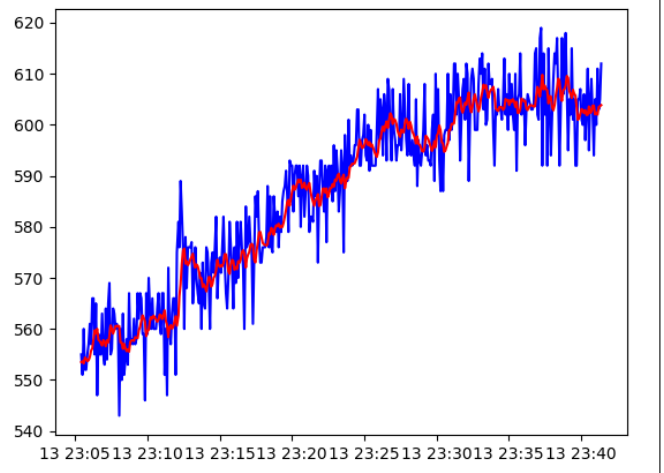
CoAP		
	average delay	packet delivery ratio
Client →ESP32 (GET)	∼ 11,44 s	100/100 = 1
Client →ESP32 (PUT)	∼ 8,52 s	100/100 = 1

HTTP		
	average delay	packet delivery ratio
ESP32 →Mosquitto →Influxdb	∼ 760 ms	100/100 = 1

4.2 Mean Square Error

TEMPERATURE	
	MSE: 0,001 RMSE: 0,026

HUMIDITY	
	MSE: 0,012 RMSE: 0,110

GAS	
	MSE: 43,278 RMSE: 6,579

LEGEND	
	Correct value
	Predictions