IoT-Based Indoor Air Quality Monitoring System

Samuele Evangelisti, Kevin Pes, Giovanni Pietrucci

1 Introduction

The system is an IoT installation focused on monitoring the indoor air quality. The sensors detect temperature, humidity and gas ppm. MCU-gateway connection RSSI is also detected directly by the MCU. Sensor data and MCU data are then sent to the proxy, collected, stored in a database and analyzed for forecasting purposes. Finally the system analyzes MCU-proxy packet traffic. The system also provides:

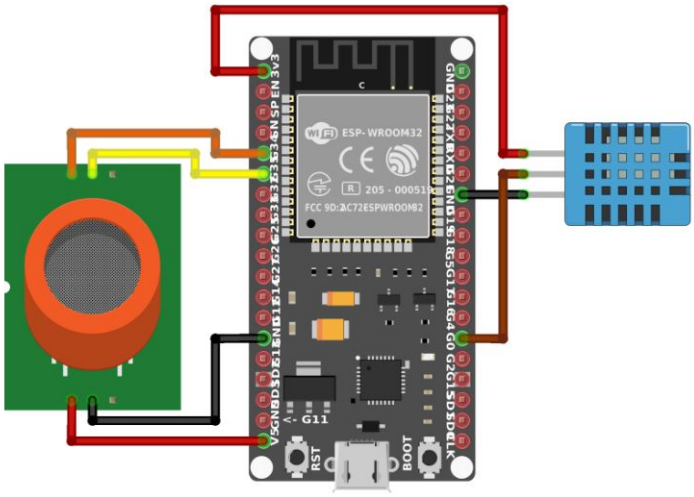
- Information about outside temperature
- A dashboard available for configuring the installation of IoT devices
- A telegram bot which can send alerts and periodic reports about data

2 Project's Architecture

The system is developed for a multisensor environment. Each sensor is implemented by the ESP32 MCU and it is connected to a proxy by a WiFi connection. The proxy is the actual machine where all the rest of the software runs (eg. database, broker, dashboard, forecasting, ...).

2.1 Device and Sensor

- Device:
  - ESP32 NodeMCU
- Sensors:
  - DHT11 (temperature and humidity)
  - MQ-2 (gas ppm)
- Schematics:
  - [DHT11] VCC -> 3v3 (red wire)
  - [DHT11] GND -> GND (black wire)
  - [DHT11] DATA -> G0 (brown wire)
  - [MQ-2] VCC -> 5v (red wire)
  - [MQ-2] GND -> GND (black wire)
  - [MQ-2] AO -> G34 (orange wire)
  - [MQ-2] DO -> G35 (yellow wire)



ESP32 wires configuration

The device needs to be connected to the same gateway of the proxy and it is able to send data to the proxy over three different protocols:

- MQTT (default)
- COAP
- HTTP

2.2 MQTT

MQTT is a protocol that follows the publish-subscribe paradigm. Generally publishers in an MQTT usage environment publish data to a broker (eg. Mosquitto). Subscribers listen for new data from the broker and use that data for internal logical tasks.

2.3 COAP

Coap is a low-cost computational messaging protocol. It is implemented in a request-response mechanism and it follows a REST architecture. Usually a sensor acts as server while the proxy acts as client requesting actual data from the sensor. In our system the ESP32 act as client and it sends data to the proxy (the server) as soon as new data is available.

2.4 HTTP

Http is the primary protocol for transmission of information across internet. Is a protocol for transporting information between a server and a client.

2.5 Forecasting

The proxy incorporates a Forcasting API call. Forecasting has been implemented through two distinct techniques: ARIMA and ProphetFB. The user must enter the ESP32 device id, then the forecasting methods will be applied and will be able to get forecasts in the next few minutes. The forecasting task is executed on single sensor data because the system is developed for a multisensor environment and the sensors could be in rooms with different air conditions.

2.6 Database

All the data related to the ESP32 are stored in the Influx database. Then the data is displayed by the configured Grafana dashboard. It is also possible to do write operations, more precisely the results related to the forcasting will be inserted in the database.

2.7 Other services and features

Other services need to be up and running to provide optional features:

- **weather-service.py**: Retrieves outside temperature from the nearest weather station and store it inside the Influx database
- **dashboard-service.js**: Provides a dashboard to configure the system IoT devices
- **telegram-bot.py**: Provides an automatic messaging system and an alert system relating to the exceeding of a minimum gas threshold.

3 Project's Implementation

The project is divided in different folder containing the source code of different aspect of the implementation:

- **ESP32**: Source code for the ESP32 firmware;
- **MQTT**: Configuration and start script for Mosquitto;
- **HTTP**: Configuration for the HTTP protocol backend;
- **Influx**: Configuration for the Influx backend;
- **Services**: Ad hoc libraries and services that need to be up and running;
- **Weather**: Configuration for the service providing outside temperature;
- **Dashboard**: Frontend and configuration for the dashboard web application;

3.1 ESP32

The MCU code is divided in libraries. Each library provides functionalities for specific aspect of the implementations:

- CoapUtils.h: Reverse COAP protocol;
- MqttUtils.h: MQTT protocol;
- HttpUtils.h: HTTP protocol and minimal HTTP web server;
- Dht11Utils.h: DHT11 sensor value reading;
- Mq2Utils.h: MQ-2 sensor value reading;
- WifiUtils.h: WiFi end point connections, Wifi access point; creations and RSSI value reading;
- JsonUtils.h: JSON reading and conversion;
- Globals.h: Global variables;
- Communication.h: Wrapper for the three communications modules;
- Firmware.ino: Firmware entry point.

3.2 MQTT

The MQTT communication protocol relies on the Mosquitto broker:

- **mosquitto.passwd**: Mosquitto password file;
- **mosquitto.conf**: Mosquitto configuration file;
- **mosquitto-start.***: Automated start script (available in .sh for bash or in .cmd for command prompt);
- **mqtt-service-config.json**: Proper configuration for the connection between Services/mqtt-service.js and the Mosquitto broker;

3.3 HTTP

- **http-service-config.json**: basic information about the local HTTP server

3.4 Influx

The data is stored in the Influx database and displayed using a Grafana dashboard. For the Influx and Grafana connection ports the default ones are used. The Influx database needs to be configured providing the following configurations:

- **Organization**:
  - "IoT"
- **Buckets**:
  - "IoT-sensor": sensor data and forecasting
  - "IoT-weather": outside temperature

In addition:

- **influx-config.json**: configuration file for the connection to the Influx database

3.5 Services

Services are the main core for the actual use of the various processes used by the system, and must be up and running.
As for messaging protocols, three different protocols have been implemented as explained above: COAP, MQTT and HTTP.
As for forecasting, it was chosen to implement both ARIMA and Prophet, two algorithms that work differently between them.
As for prophet, it resides in its special service that proceeds with a single training phase and forecasting phase making it compact and fast, its activation is periodically so that you can always analyze new fresh data.
Arima's service instead is divided in two phases: a phase of periodic training and an immediate forecasting. The choice to divide it lies in the fact that training a large amount of data can cause the system to slow down. So it was chosen to keep a periodic training active in the background, even here as in prophet, in order to recall the models saved in training and use them in the actual forecasting module.

3.6 Weather

weather-service-config.json: Latitude and longitude for finding the nearest meteorological station

3.7 Dashboard

The dashboard is a web application developed in Angular Material. It provides a simple management interface for the deployed MCUs. The interface is developed for computer screens in landscape mode and it is divided in two parts:

- **Search panel** (left part of the screen): Provides functionalities to find a deployed MCU, to add a found MCU or to directly add a deployed MCU knowing the IP of the MCU and the port of the MCU HTTP server.
- **Management panel** (right side of the screen): Provides a list of expansion panels, one for each MCU added to the list. Opening an expansion panel it is possible to configure the MCU connected to to panel

To avoid being blocked by CORS policy, every communication from the dashboard is intercepted and passed through the dashboard proxy developed inside Services/dashboard-service.js.

3.8 Telegram Bot

The telegram bot has been implemented thanks to the telegram.exe package. After setting up the tokens created in the telegram app, it was chosen to develop two distinct steps to facilitate user use. The service provides both an automatic recap of daily information every 5 hours and a recap requested by the user through interaction with the bot chat. You can also receive gas alerts. If the latter exceeds a certain threshold, then the bot will notify it in chat making possible the intervention of the user.

3.9 Forecasting

The system implements two forecasting techniques: Arima and Prophet. The development of the two follows different procedures that will be explained later. The results relating to the accuracy of the predicted results will also be compared.

3.9.1 ARIMA

ARIMA is a class of statistical model for analyzing and forecasting time series data. More precisely is an autoregressive integrated moving average models which predict future values based on past values. ARIMA model assumes time series data are set as stationary such that mean and variance are constant. In order to check if the time series is stationary, we're going to use the Dickey Fuller Test (ADF) which is a hypothesis test where the null hypothesis is if the time series is non stationary. The rejection of the null hypothesis is when p-value is less than 0.05. If the ADF doesn't reject the null hypothesis, a differencing technique can be applied to the data in order to transform a non-stationary time-series into a stationary time-series. This method must be applied until the null hypothesis is rejected. Once the time-series is finally stationary, we need to define the parameters for the ARIMA model.

The model works on the combination of the following three values:

- p is the number of autoregressive terms;
- d is the number of nonseasonal differences needed for stationarity;
- q is the number of lagged forecast errors in the prediction equation.

In order to obtain these parameters it is necessary to apply an autocorrelation and a partial-correlation test.

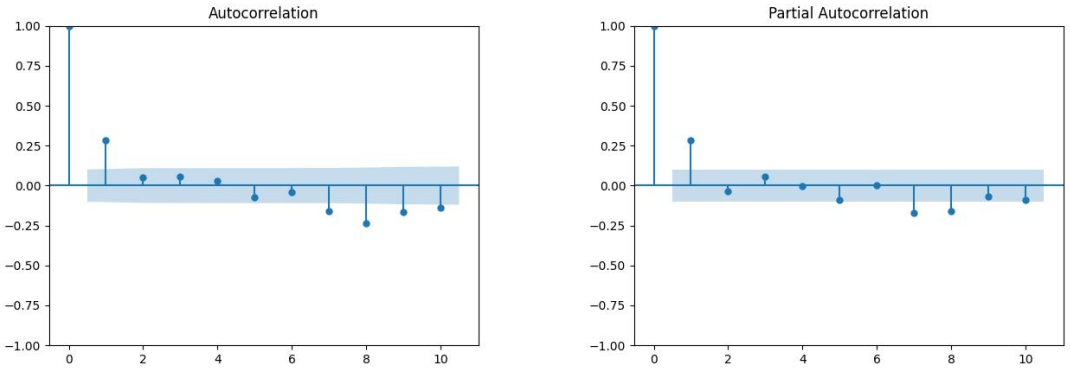


Figure 1: Autocorrelation and Partial autocorrelation tests

Since the system must be fast to return the predicted values, it was necessary to divide the training and forecasting phases.
As for the training, the user must enter the sensor id, then a query will be applied on the influx database to obtain the timeseries related to the sensor requested by the user. The training phase

related to temperature, humidity and gas will be applied. The obtained models will be saved in 3 separate files; entering the second phase, the previously saved models will be load and used by an effective forecasting method that will obtain the future timeseries.

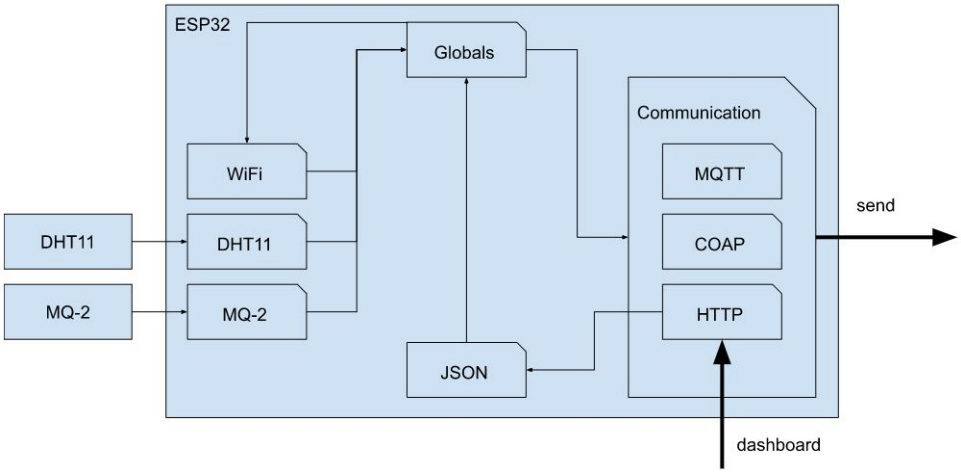
3.9.2 Prophet

Prophet is a forecasting tool for time series data based on an additive regression model. It works best on time series with strong seasonal effects. Prophet can automatically detect changes in trends and it's based on an automatic method where pre-processing and parameter tuning are not necessary. The implementation of prophet is therefore faster and more compact than ARIMA as the process is automated, however it is interesting to compare the results such as RMSE, the average values and the confidence intervals of both forecasting algorithms.
The training speed of prophet allowed to use directly a single file for the training phase and forecasting. Following the development of arima, the user is asked to enter the sensor id in order to make a targeted query on influx. As a result a method for predicting future timeseries is obtained.

3.10 General structure

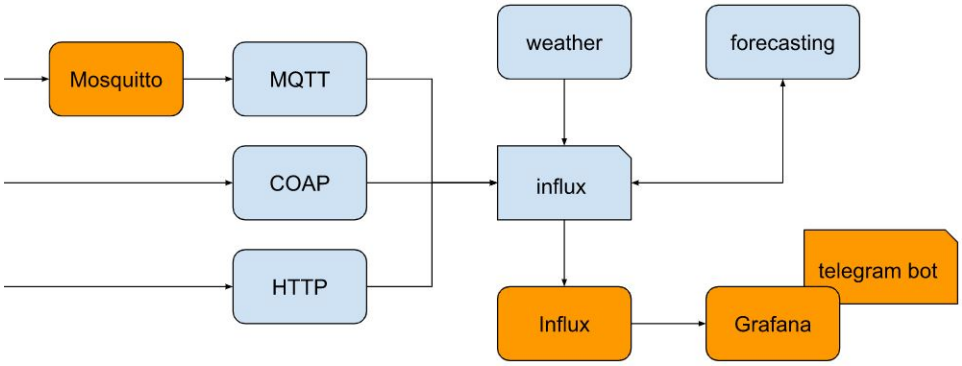
As explained before, here there is a graphical representation of the MCU structure and the system structure.

3.10.1 MCU



MCU Structure

3.10.2 System



System Structure

4 Results

4.1 Communication Protocols

Average Trasmission Delay		
MQTT	**COAP**	**HTTP**
1.84 ms	1.44 ms	485.95 ms

Packet Delivery Ratio		
MQTT	**COAP**	**HTTP**
100 %	98.66 %	100 %

4.2 Forecasting Algorithms

Prophet data analytics			
Sensor Type	**RMSE**	**Average**	**Confidence Interval**
Temperature	0.040	32.267	[31.621,32.913]
Humidity	0.546	29.966	[24.169 , 35.763]
Gas	0.296	200.130	[199.838 , 200.422]

Arima data analytics			
Sensor Type	**RMSE**	**Average**	**Confidence Interval**
Temperature	0.030	31.481	[31.073,31.889]
Humidity	0.196	33.221	[30.196 , 36.226]
Gas	0.287	200.056	[199.957,200.155]

4.2.1 Conclusions

As expected the fastest communication protocol is COAP but it is also the only one that can lose packages. The slowest communication protocol is HTTP but with full reliability. Finally the MQTT communication protocol has a good transmission delay and a full reliability.
As we can see the forecasting carried out on arima is more precise than prophet. the reason resides not only for the difference between the algorithms, but also for a better setting of the values relating to the verification and conversion of the stationarity of the timeseries.