IotProject2022

Francesco Palmisano 1004694 francesco.palmisano2@studio.unibo.it
Primiano Arminio Cristino 1004689 primiano.cristino@studio.unibo.it
Università degli studi di Bologna

***Abstract*—The implementation of an IoT-based project, according to the Internet of Things (IoT) pipeline, consists of data generation and processing on edge devices, data acquisition through communication protocols, data management via a database and data visualization on a dashboard, data processing and data analytics. This project aims to monitor indoor environmental parameters such as temperature, humidity and gas concentration with ESP32 devices. Then, through different protocols these data are processed, collected and visualized on a proxy server. Eventually, data are sent to InfluxDB and Grafana dashboard.**

I. Introduction

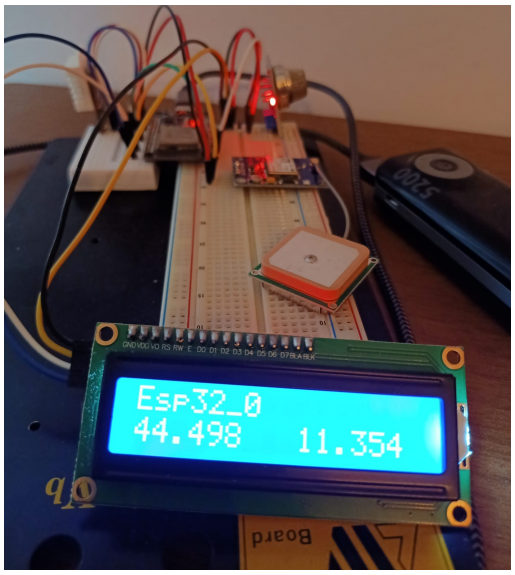

Fig. 1. Full connected ESP32 circuit

In the last years, there has been an increasing interest concerning the IoT solutions aiming to collect, monitor and analyze sensory data in constrained environments. There are several technologies and techniques that can be used, according to the IoT pipeline. This project has been designed for an energy-constrained system to monitor indoor environmental parameters. By exploiting different protocols such as CoAP, HTTP and MQTT IoT devices (in particular ESP32) are able to communicate with a Python proxy server. This server collects data such as temperature, humidity, gas concentration, Received Signal Strength Indicator (RSSI), Air Quality Monitoring (AQI) and the Media Access Control (MAC) address. These data are collected time by time (with a tunable sample frequency parameter) and stored on a time series database (InfluxDB). Eventually, data analytics are shown by the Grafana dashboard.
The rest of the paper is organized as follow. In Section II, we present our test-bed and implementation for ESP32, while in Section III we illustrate the Python middleware, located on a Raspberry Pi4. In Sections IV and V we show how data are stored and how the forecast model works. Finally, in Sections VI and VII conclude our work by showing our results and future progresses.

II. Devices

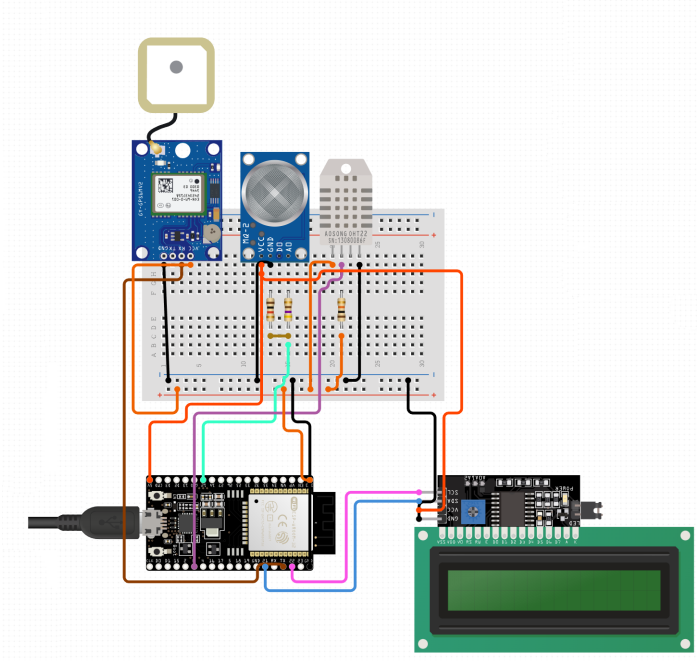

Fig. 2. ESP32 circuit scheme

Due to the low-power usage, we have used the ESP32 chip that represents a common open source Arduino-like board which is implemented in most of sensor IoT environments. Every ESP32 communicate with the following devices:

- DHT22 module for reading temperature and humidity [1].
- MQ2 module for reading the raw gas concentration.
- NEO-6M module for retrieving the GPS coordinates[2].
- LCD display for showing useful information [3].

According to the option selected by the proxy server, the ESP32 chip uses different libraries to communicate with the proxy server:

- HTTPClient for HTTP [4].
- PubSubClient for MQTT [5].
- coap-simple for CoAP [6].

coap-simple library.

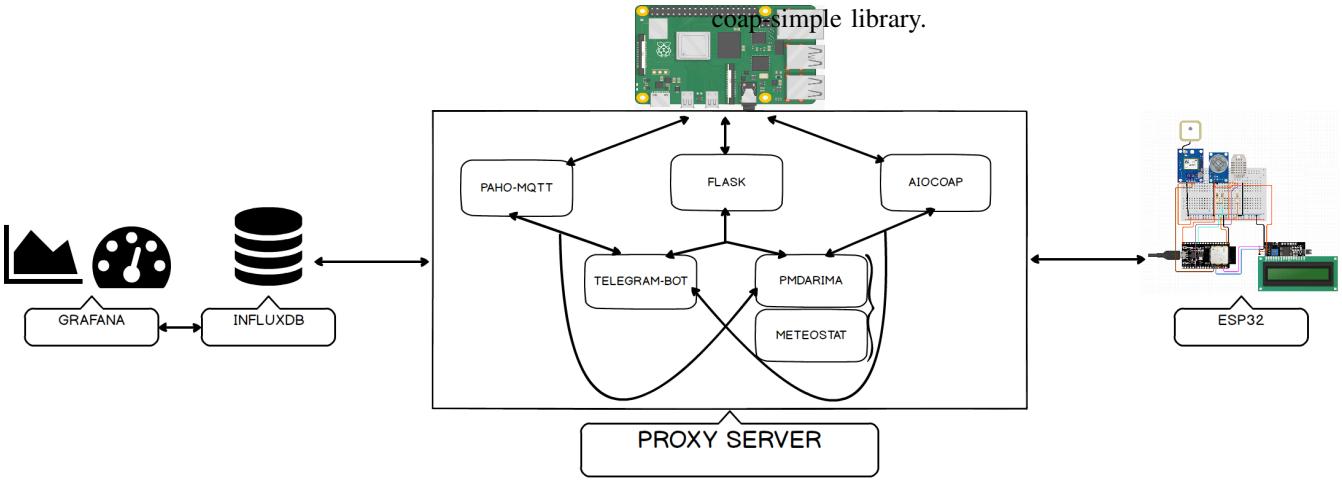

Fig. 3. Middleware architecture

III. SERVER

IoT devices communicate with a python middleware located in a raspberry Pi4, turned also as a router. It collects data received by ESP32 chips along different protocols (HTTP, CoAP and MQTT). On one hand, HTTP and CoAP server are included in the python middleware respectively with Flask[7] and aiocoap[8] libraries. While, on the other hand, when communicating via MQTT, both the server[9] and each IoT device[5] communicate with an external broker (broker.emqx.io).

Each IoT device exchanges data with the proxy server. It provides a Flask-based web application to monitor locally the behaviour of IoT devices by visualizing continuous updates of each ESP32 as well as its own information such as GPS coordinates, MAC address and id.

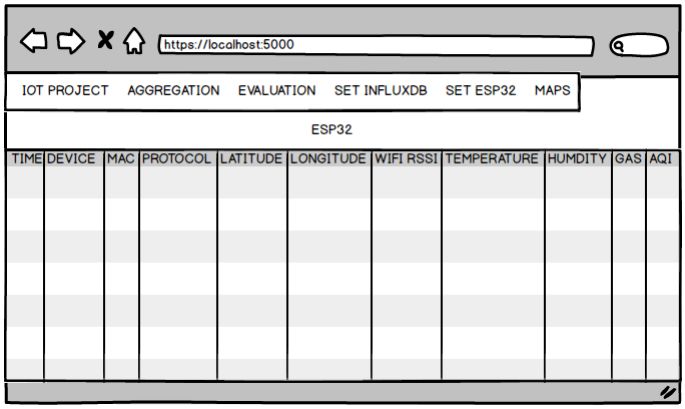

Fig. 4. Web User Interface to interact with ESP32 chips

Possibly, the ESP32 receives the configuration from the server modifying its internal parameters such as the current protocol, the sample frequency, maximum and minimum gas value for AQI calculation[10]. Each device has an own personal tunable configuration. They also support multiple protocols for the configuration change operations except for the CoAP device configuration that is still in HTTP due to issues in the ESP32

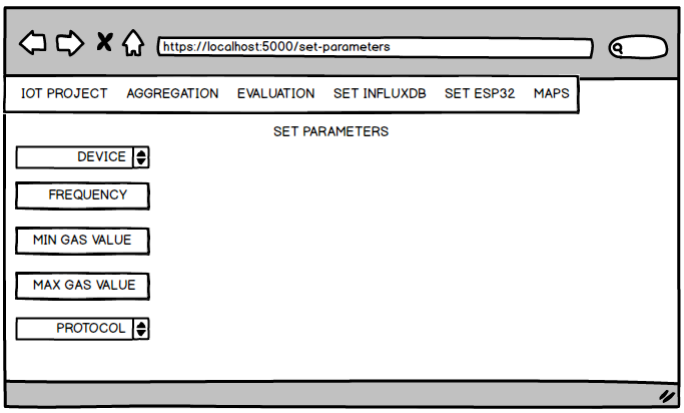

Fig. 5. Web view for data received by ESP32 chips

Moreover, the proxy server forecast future predictions of the sensory and meteostat data[11] via SARIMA[12]. These predictions are sent to InfluxDB[13] and eventually visualized both on Grafana dashboard and on the proxy server.

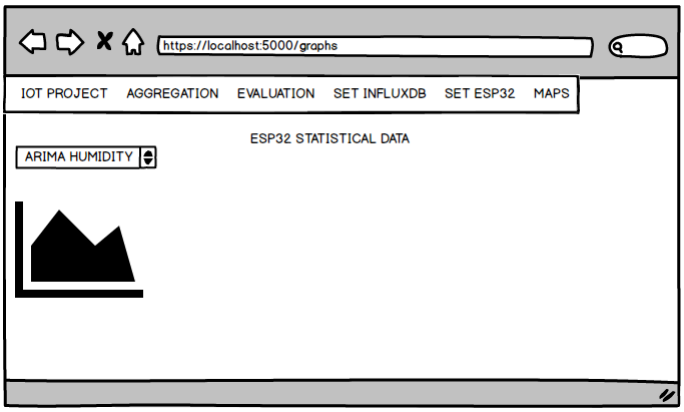

Fig. 6. Web page that show graphs not only about delay and PDR, but also for humidity, temperature and gas concentration (raw and predicted values).

Furthermore, the web application computes statistics over data by showing graphs of the delay, packet delivery ratio (PDR) and predictions, as well as data aggregation including the maximum, minimum, mean and standard deviation.

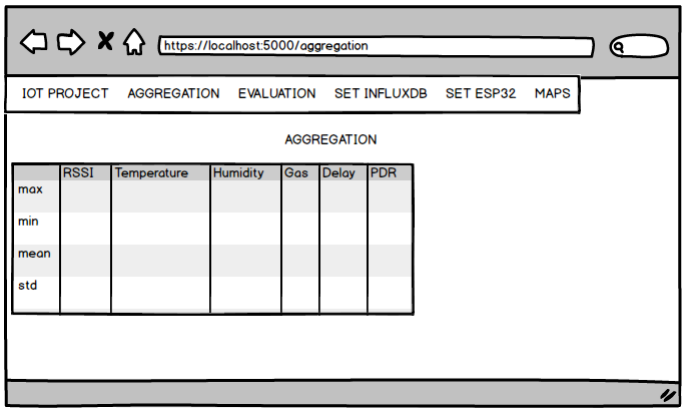

Fig. 7. Web page that show the statistics of parameters such as RSSI, temperature, humidity, gas concentration, Delay and PDR.

Each ESP32 device GPS coordinates are visualized via an Open Street Map. A red marker represents the current selected device, while the others have been set to a bluish color.

Fig. 8. Web view that show the GPS position of every ESP32 chips.

InfluxDB parameters can also be set server-side.

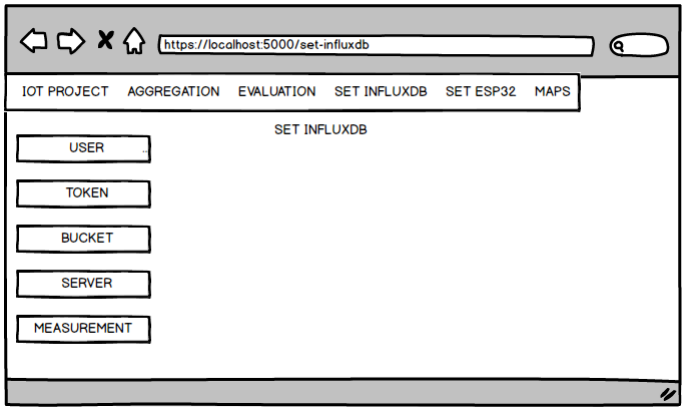

Fig. 9. Telegram bot group chat updates.

Eventually the proxy server send aggregate data to a telegram channel via python-telegram-bot library[14].

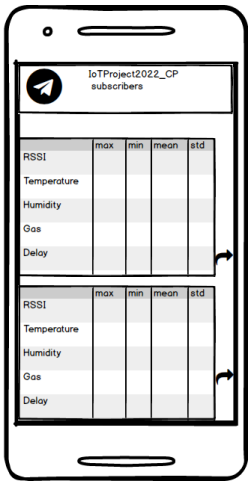

Fig. 10. Telegram bot group chat updates.

IV. DATABASE

The data are obtained by sensors, then collected by the proxy server and stored in an Influx Database as follows. The tags are the MAC address and GPS coordinates. The values are the sensory data and their predictions as well as RSSI and AQI. InfluxDB is not a great choice due to its inconsistency at writing data. Data collected in a short time are not fully stored correctly, even in a synchronous environment. At least 10 seconds should elapse between each update.

V. FORECASTING

The proxy server forecast future predictions via pmdarima library. ARIMA is not a great model for the received sensory data. Thanks to pmdarima library, seasonality can be added to the computation. This model is called SARIMA. First of all, the library tunes p and q (order) values automatically, as well as pm and qm (seasonal order). While, d and dm are respectively computed by their null hypothesis. Moreover, predictions have been calculated both on sensory data and meteostat library values. The default window is about 5 measurement for sensory data and an entire year for meteostat.

Fig. 11. Meteostat

VI. RESULTS

First we have performed multiple tests with different range distances and obstacles to analyze the performance (packets delay and PDR) of MQTT, HTTP and CoAP protocols.

Indoor values	MSE	Interval Confidence
Temperature	0	[27.74, 27.92]
Humidity	0.016	[38.67, 39.73]
Gas	9.2	[183.52, 203.21]

TABLE I. COMPARISON OF PERCENTAGES OF THE WINDOW OF 5 MEASUREMENTS.

To calculate PDR, the Network Time protocol has been used for clock synchronization, both on ESP32 devices and the server.

RSSI = [-89db, -79db]			
	HTTP	COAP	MQTT
Delay	0.64	0.56	1.13
PDR	0.95	1.0	1.0

TABLE II. DELAY AND PDR RESULTS WITH AT LONG DISTANCE WITH THE ROUTER.

RSSI = [-77db, -73db]			
	HTTP	COAP	MQTT
Delay	0.55	0.54	0.84
PDR	0.95	1.0	1.0

TABLE III. DELAY AND PDR RESULTS WITH AT MEDIUM DISTANCE WITH THE ROUTER.

RSSI = [-42db, -26db]			
	HTTP	COAP	MQTT
Delay	0.57	0.59	0.82
PDR	1.0	0.99	1.0

TABLE IV. DELAY AND PDR RESULTS WITH AT SHORT DISTANCE WITH THE ROUTER.

In each case HTTP and CoAP packets delay are very similar. Nevertheless in MQTT, the delay is slightly higher than the protocols above, due to its communication with the external broker. Moreover, it's interesting to see that the PDR is pretty much the same whatever the range distance is.
Thanks to the SARIMA model the predicted results of each sensory data, set by default to 5 future predictions, are not so far to the real one.
A Grafana alert has been initialized to be fired when the AQI level is above 1.

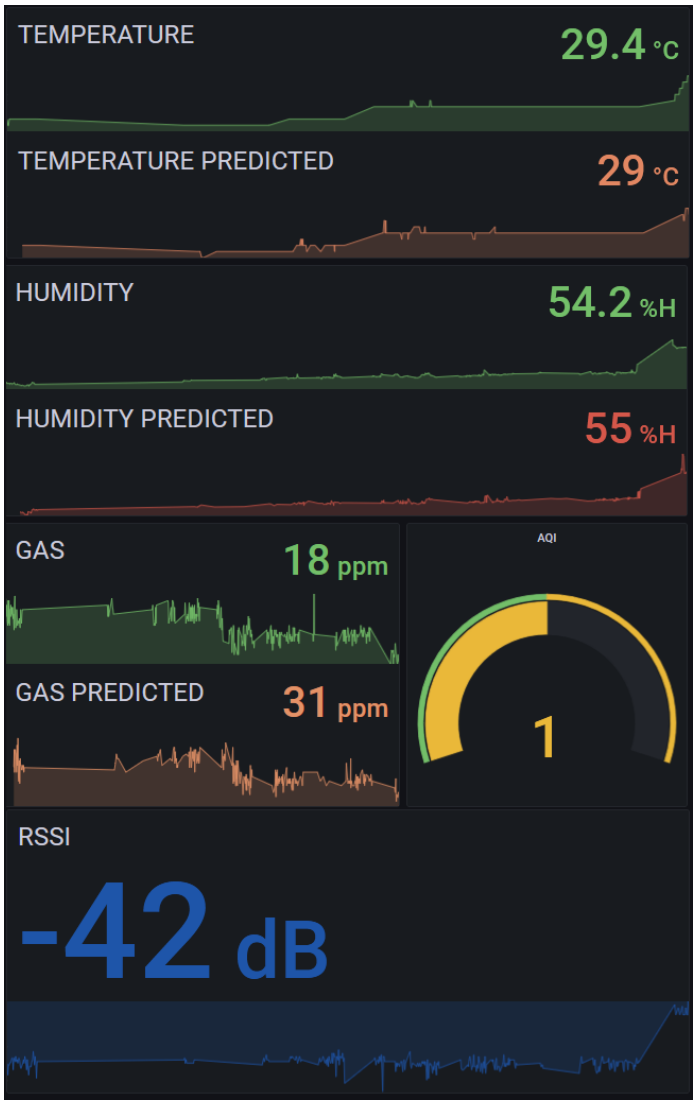

Fig. 12. Grafana dashboard.

VII. CONCLUSION

Even with a low budget, it is possible to implement and review a complete IoT systems, from the collection of sensory data to the prediction of them via custom models.
Suggested further improvements could be:

- Use more complete CoAP library for ESP32 (CoAP simple was the only choice available).
- Use more reliable sensors. Some of them, especially MQ2 chips, were malfunctioning very frequently.
- Use IoT devices with integrated clock chip to replace the NTP library[15], because the NTP server is sometimes down.
- Use a different database than InfluxDB, due to inconsistency data storing.

REFERENCES

[1] *DHT esp32 library*. https://github.com/adafruit/DHT-sensor-library.
[2] *GPS esp32 library*. https://github.com/mikalhart/TinyGPSPlus.

[3] *LCD esp32 library*. https://github.com/fdebrabander/Arduino-LiquidCrystal-I2C-library.
[4] *HTTP library esp32*. https://github.com/amcewen/HttpClient.
[5] *MQTT library esp32*. https://github.com/knolleary/pubsubclient.
[6] *CoAP esp32 library*. https://github.com/hirotakaster/CoAP-simple-library.
[7] *Flask python library*. https://github.com/pallets/flask.
[8] *CoAP python library*. https://github.com/chrysn/aiocoap.
[9] *MQTT python library*. https://github.com/eclipse/paho.mqtt.python.
[10] *Meanfilter esp32 library*. https://github.com/luisllamasbinaburo/Arduino-Meanfilter.
[11] *Meteotstat python library*. https://github.com/meteostat/meteostat-python.
[12] *Pmdarima python library*. https://github.com/alkaline-ml/pmdarima.
[13] *InfluxDB python library*. https://github.com/influxdata/influxdb-client-python.
[14] *Telegram python library*. https://github.com/python-telegram-bot/python-telegram-bot.
[15] *NTP python library*. https://github.com/remh/ntplib.