Sound Level Monitoring System

1st Igor Iurevici
Alma Mater Studiorum • University of Bologna
Master's Degree in Computer Science
Bologna, Italy
igor.iurevici@studio.unibo.it

***Abstract*—Most urban areas suffer of a growing noise pollution problem which affects the quality of life of millions of people worldwide. High noise levels are potentially harmful in regards of stress, sleep disturbances and reduced cognitive function. Considering the exponential growth of IoT systems and them being accessible to a wide range of users, with this project I want to explore and develope a prototype of a noise monitoring system in order to have a numerical and graphical feedback of the sound level of my home environment.**

***Index Terms*—communication, dashboard, esp32, forecast, influxdb, http, iot, mqtt, noise, proxy**

I. Introduction

This project aims to exploit modern IoT devices and software in order to develop a prototype of a Sound Level Monitoring System able to acquire, process and store sound samples in order to give an overall idea of the environment noise pollution and alert critical cases. In particular *esp32*, a power-efficient microcontroller, and a *GY-MAX4466* are going to be explored and used for this project.
In addition to that a *Flask* app is also developed in order to manually configurate the following parameters:

- `Sampling_rate`: interval between consecutive sensor readings;
- `Noise_threshold`: only values higher than a threshold are stored;
- `Alarm_level`: threshold to generate an alarm;
- `Alarm_counter`: trigger an alarm even in case the noise level exceeds `Alarm_level` for a consecutive number of samples at least equal to `Alarm_counter`.

Other than hosting the configuration webpage, the Flask app serves as a proxy and forwards new configurations to the esp32 board via MQTT protocol and receives sensor data via HTTP persistent connection. Both with alarm level and noise values, also WiFi rssi value are acquired and send to the proxy which stores everything in a InfluxDB time-series database. The database is periodically queried and a graphical representation of the data is made on a Grafana dashboard.
Finally a Prophet model utilizes the existing data in order to forecast the future trend of the noise values.

II. Project's Architecture

The previous prototype introduction and description is well described in figure 1.

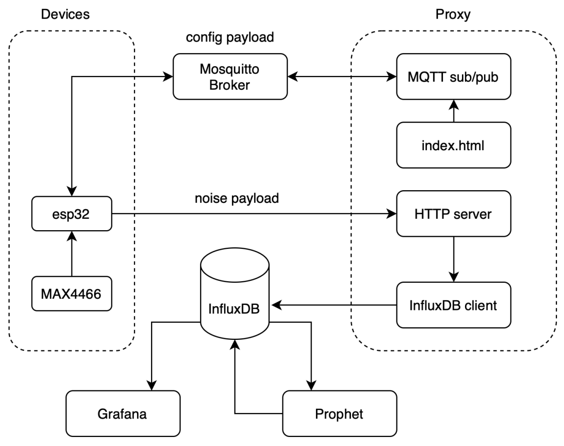


Fig. 1. System architecture.

III. Project's Implementation

In this section we dive in the implementation of each aspect present in the system architecture.

A. Data acquisition

The audio signal are periodically acquired by the MAX4466 sensor through an ADC process. A single analog read consists in an integer between 0-4095 using a 12bit resolution, but since a single sample doesn't give a meaningful representation of the noise level, the sampling is done in a fixed window of 50ms (20Hz) where the difference between the maximum positive peak and maximum negative peak of the window wave is computed, which describes the amplitude of the audio signal, also called **peak-to-peak**. Given the MAX4466 sensitivity of -44 +-2 dBV/Pa, its gain range of x25 to x125, we can convert peak to peak to voltage and then decibel using the following formula:

$$20 \times \log\left(\frac{V_1}{V_0}\right)$$

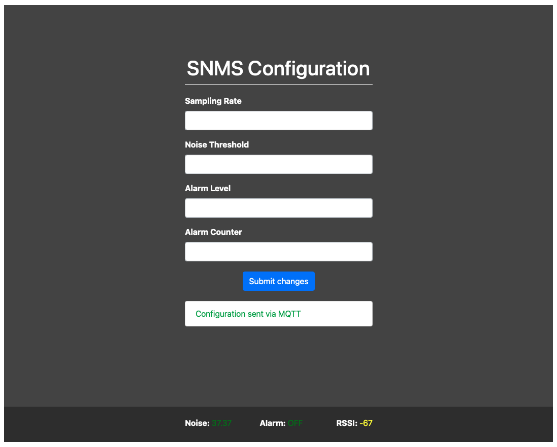
Since I use the sinusoidal amplitude peak as reference, the obtained value isn't very accurate, but still a good approximation to get an overall idea of the noise level.
A fixed denoising costant is applied to peak-to-peak value in order to eliminate the internal noise.
The decibel value is then processed through threshold, the alarm flag is updated and the result is packed with the WiFi RSSI value as a comma separated string.

B. Data communication

The communication is done exploiting two protocols: MQTT and HTTP.

1) Payload via HTTP: A persistent HTTP connection is established between esp32 (client) and Flask app (proxy) where the payload is periodically send via POST request. The proxy prepares and forwards the received data to the InfluxDB database as a new Point.

2) Configuration via MQTT: The Flask app hosts a simple web page where the user is able to type the desired values for the esp32 configurable parameters which are locally stored in a json file, and published on the configuration topic of the Mosquitto, broker after a consistensy check (fig. 2).


SNMS Configuration

Sampling Rate

Noise Threshold

Alarm Level

Alarm Counter

Submit changes

Configuration sent via MQTT

Noise: 37.37 Alarm: OFF RSSI: -67

Fig. 2. Configuration web page.

The proxy is also subscribed to a discovery topic which is used to receive an "awake" message from a fresh connected esp32 in order to send it the current configuration.

C. Data storage

A InfluxDB cloud account has been setup using the free plan, which is enough for the purpose of the project. The received payload is validated and unpacked by the proxy and a Point object is prepared to be send to the InfluxDB (Cloud) bucket. In particular 3 fields are arranged:

- `noise`: dB value of the sample;
- `alarm`: boolean status of the alarm at the given sample;
- `rssi`: WiFi strength expressed as dBA.

A second bucket is created for a downsampling process performed via tasks using an moving average approach of a 10 seconds window. In this way the data storage is optimized by retaining single samples for the last 12 hours, and the aggregation values for 7 days.

D. Forecasting

A Prophet model is trained on the last 6 hours of data and forecasts 10 seconds in the future. Prophet is a prediction model designed for time-series data, allowing to predict its future trend based on history data. In this particular use-case I expect it to anticipate possible high peak of noise in the near future.

The model is updated periodically with new data points since in my enviornment noise pattern is evolving over time.

A window of 1h of data has been used to validate the model on a test set of 30%. The graphical result can be appreciated on the fig. 3.

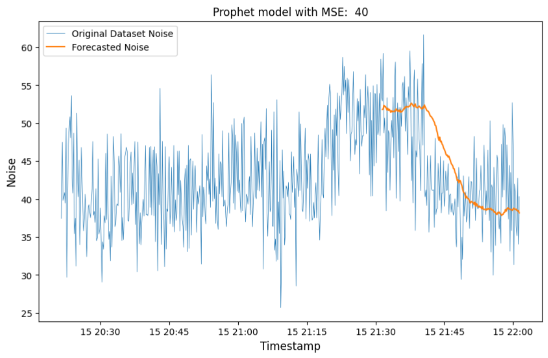


Fig. 3. Model validation.

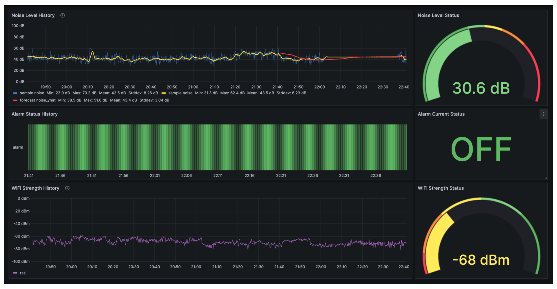
E. Dashboard

The InfluxDB aggregations and the forecasted data is visualized on a Grafana dashboard which allows to observe the time-series data in a clear format obtained by recurrent queries. The dashboard (fig. 4) shows data as following:

1) Noise values: The noise data shows the current value, aggregated mean value, window average value over 10 samples and the forecasted data.

2) Alarm status: The current alarm status is show together with the aggregated alarm values. The aggregation of alarm values consists in a sum of status over that 10 second window.

3) WiFi RSSI values: RSSI aggregation mean values over 10 second window.



sample noise Min: 23.9 dB Max: 70.2 dB Mean: 43.5 dB Stddev: 8.26 dB
sample noise Min: 31.3 dB Max: 62.4 dB Mean: 43.5 dB Stddev: 6.23 dB
forecast noise_yhat Min: 38.5 dB Max: 51.6 dB Mean: 43.4 dB Stddev: 3.04 dB

Fig. 4. Grafana dashboard.

IV. Results

The overall result of the project is solid and it performs in an excellent way the main job of detecting high noise values over a certain amount of time. Personal testing of the dashboard allowed to visually observe and detect patterns over daily interval in an indoor environment. The sampling rate has been kept for about 1-2 seconds for the entire testing process, but it resulted to be consistent even when stressed under 0.5

second.
The esp32 microcontroller was able to perform the task, even though it is not the best device for analog reads since its affected from periodical fluctuations. A possible future work could consist in improving the hardware aspect introducing resistors or trying different boards. Also a cluster of devices could be used to monitor different rooms of the indoor environment. Scaling would also require an edit on the security aspect.