

Smart Ventilation System

An IoT-based project

Author: Gabriele Frassi

University of Pisa

Department of Information Engineering

Master degree in *Artificial Intelligence and Data Engineering*

prof. Giuseppe Anastasi, prof. Carlo Vallati, eng. Francesca Righetti

TABLE OF CONTENTS

1	INTRODUCTION	3
1.1	ADDRESSED PROBLEM	3
1.2	PROPOSED SOLUTION AND USE-CASE.....	3
1.3	WHY CoAP?	3
1.4	LIMITATIONS OF THE NRF52840 DONGLES WITH FLOATING-POINT OUTPUT	3
2	COMPONENTS OF THE SYSTEM	4
2.1	BORDER ROUTER	4
2.2	CO2 NODES	4
2.3	ACTUATOR NODES	5
2.4	CLOUD APPLICATION	5
2.5	USER APPLICATION.....	7
2.6	DATABASE	8
3	DEBUG AND TEST	9

1 INTRODUCTION

1.1 Addressed problem

We operate in large buildings, characterized by numerous rooms and a constant flow of people. In these environments, concentrations of CO₂ and other human-related indicators can rise quickly, creating conditions that are unhealthy and not ideal to productivity.

The goal is to effectively manage indoor air quality, ensuring comfort and safety for occupants while optimizing energy consumption, given that ventilation and air-conditioning systems are among the main contributors to global pollution.

1.2 Proposed solution and use-case

The development of a *Smart Ventilation System* is proposed, designed to optimize the management of ventilation and air conditioning systems within buildings, thus ensuring both occupant comfort and environmental sustainability. The system is based on two main type of devices:

- **CO2 Node.**
Equipped with CO₂ and humidity sensors, it can estimate the likelihood of people being present in the environment based on the values detected. These parameters are considered among the most reliable for presence detection. It did all measurements periodically.
- **Actuator Node.**
Connected to a component of the ventilation system, it receives the the estimated probability of presence and determines whether to activate the ventilation component. This is based on a configured threshold.

The devices enable the creation of a hierarchical system, where each component of the ventilation system is associated with a specific threshold. Low-power devices can be linked to lower thresholds and activated first. If their operation is insufficient to reduce CO₂ and humidity, the system will then activate more powerful devices, which consume more energy (and are associated with higher thresholds).

Devices can be arranged within the building according to its layout: each CO₂ node, responsible for a specific area (for example, a room), coordinates a set of actuator nodes that act upon the same area using the measured values.

1.3 Why CoAP?

The *Constrained Application Protocol* (CoAP) is adopted in this solution. CoAP is particularly suited for large-scale systems with a potential high number of devices. The system may be deployed in very large buildings, such as skyscrapers. Our goal is to manage interactions between CO₂ Nodes and Actuator Nodes locally, minimizing as much as possible the delay between detecting critical values and triggering the ventilation system components.

MQTT, on the other hand, is less suitable due to its limited flexibility, especially the requirement that every message must pass through an MQTT Broker.

1.4 Limitations of the nrf52840 dongles with floating-point output

Some values (thresholds and occupancy predictions of *Actuator Nodes*) are exchanged as integer values due to limitations of the nRF52840 dongles with floating-point output.

Each value is multiplied by 100 and converted into an integer by the sender; the integer is then sent to the receiver, which divides it by 100 and converts it back to a float.

Why 100? Two decimal places are sufficient for our application

2 COMPONENTS OF THE SYSTEM

2.1 Border Router

The *Contiki-NG border router* has been deployed to enable communication among the system's components. The router provides a list of the currently connected nodes through its web interface.

2.2 CO2 Nodes

The *CO2 Node* is a Smart Device able to periodically measure CO2 and humidity, according to a configured sampling rate. Using these values, the CO2 Node will compute an occupancy prediction (the probability of having people within a room) using a Machine Learning Model.

After powering on, the device will register itself with the *Cloud Application*. All periodic measurements will be made available to the *Cloud Application* and the *Actuator Nodes* with the activation of the */CO2* resource.

- **Exposed resources.**

The CO2 Node will provide two resources:

- ***/CO2 resource (Observable resource).***

Only the GET method is available. The resource returns the most recent measurements from the CO2 Node (CO2, humidity, and occupancy prediction).

- ***/sampling resource.***

- GET Method.

The current sampling rate of the CO2 Node is returned.

- PUT Method (non-safe, but idempotent).

The method allows changes in the CO2 Node's sampling rate. The *User Application* will provide the following JSON

```
{ "new_sr": "%i" }
```

where *new_sr* is the new sampling rate. The *Actuator Node* will check if the

sampling rate is a correct value: if not a *BAD_REQUEST* error will be returned;

otherwise, the sampling rate will be updated. The change will be effective after the next scheduled measurement.

- **Observed resources.**

The CO2 Node does not observe any resource.

- **Machine Learning Model.**

The CO2 Node uses a Machine Learning Model in order to predict the probability of having people within a room.

- The *Occupancy Detection* dataset¹ by Luis Candanedo has been adopted to train the model. The *Occupancy* column provides a ground-truth base for classifying possible set of values.
- A dense neural network was implemented using TensorFlow library, with the aim of estimating the probability that a room is occupied (Occupancy = 1) based on specific input values. The features used for training the network were CO2, Humidity, and Occupancy.
- The model has been trained according to procedure explained during lectures.
- emlearn library has been adopted in order to allow execution of the model on low power devices!
- The execution of the machine learning model is indicated by the blue LED.

¹ <https://archive.ics.uci.edu/dataset/357/occupancy+detection>

2.3 Actuator Nodes

The *Actuator Node* is a device connected to a component of the ventilation system. It can activate or deactivate the component based on the received occupancy predictions and the configured threshold.

After powering on, the device will perform the following actions: it will register with the *Cloud Application*, providing the node ID supplied by the user via the serial line²; it will then use the IP address and port returned by the *Cloud Application* to establish a connection with the CO2 Node and observe its `/CO2` resource.

- **Exposed resources.**

The *Actuator Node* exposes the resource `/threshold`.

- o GET Method.

The current threshold of the *Actuator Node* is returned.

- o PUT Method (non-safe, but idempotent).

The method allows changes in the *Actuator Node*'s threshold. The *User Application* will provide the following JSON

```
{"new_t": "%i"}
```

where `new_t` is the new threshold. The *Actuator Node* will check if the threshold is a correct value: if not a `BAD_REQUEST` error will be returned; otherwise, the threshold will be updated. Remember limitations explained in section 1.4.

- **Observed resources.**

The *Actuator Node* observes the `/CO2` resources exposed by the *CO2 Node* it is associated with.

- o Each measurements packet will be in the following form

```
{"c": "%i", "h": "%i", "op": %i}
```

where `c` is the measured CO2 (ppm), `h` is the measured humidity (%) and `op` is the computed occupancy prediction (by *TinyML*). Remember limitations explained in section 1.4.

- o The *Actuator Node* will consider only the occupancy prediction (`op`) value: if this value exceeds the current threshold, the component will be activated; otherwise, the component will be deactivated.

- **Representation of the active status of the component.**

The component of the ventilation system may be active or inactive, depending on the adopted threshold. If the device is active the blue LED on the dongle will be active. Otherwise, no LED will be on.

2.4 Cloud Application

The *Cloud Application* is responsible for the following activities:

- registering *CO2 Nodes*;
- registering *Actuator Nodes* and establishing associations between *Actuator Nodes* and *CO2 Nodes*;
- managing and updating all key information stored in the database (Information on registered *CO2 Nodes* and *Actuator Nodes*, *CO2 Node* measurements, status of each *Actuator Node*).

All functionalities have been implemented in Python, using *CoAPthon*³ and *MySQL Connector*⁴ libraries.

- **Exposed resources.**

The *Cloud Application* exposes one resource: `/registration`

- o Only one method is available: `POST` (the action is *non-safe* and *non-idempotent*).

² Node ID must be provided using serial line: command `make login` must be launched in the terminal in our case.

³ <https://github.com/Tanganelli/CoAPthon>

⁴ <https://pypi.org/project/mysql-connector-python/>

- The resource is used by both *CO2 Nodes* and *Actuator Nodes* for their registration.
 - *CO2 Nodes.*
 - The CO2 Node will provide the following JSON
`{"type": "S", "settings": {"rate": "%i"}}`
 Where `type` indicates that the node is a *CO2 Node* and `rate` specifies the adopted sampling rate.
 - The Cloud Application will check if information on the node (IP address and port of the sender) are already available: if so, the database will be updated (UPDATE query); otherwise, the new information will be inserted (INSERT query). Possible errors:
 - `BAD_REQUEST` in case of a wrong sampling rate or wrong type
 - `INTERNAL_SERVER_ERROR` in case of errors by MySQL connector
 - The Cloud Application will observe the `/CO2` resource provided by the newly registered CO2 Node.
 - *Actuator Nodes.*
 - The CO2 Node will provide the following JSON
`{"type": "A", "settings": {"node": "%i", "threshold": "%i"}}`
 Where `type` indicates that the node is an *Actuator Node*, `node` is the identifier of the *CO2 Node* and `threshold` is the adopted threshold. Remember limitations explained in section 1.4.
 - The Cloud Application will check if the CO2 Node has been registered: if not a `BAD_REQUEST` error will be returned.
 - The Cloud Application will check if information on the node (IP address and port of the sender) are already available: if so, the database will be updated (UPDATE query); otherwise, the new information will be inserted. Possible errors:
 - `BAD_REQUEST` in case of a wrong node id and/or wrong threshold
 - `INTERNAL_SERVER_ERROR` in case of errors by MySQL connector
 - If everything is okay, the *Cloud Application* will provide the *Actuator Node* with the IP address and port of the *CO2 Node*, allowing it to connect and receive all measurements.

- **Observed resources.**

The *Cloud Application* observes all `/CO2` resources exposed by the registered *CO2 Nodes*, allowing it to receive periodic measurements and occupancy predictions.

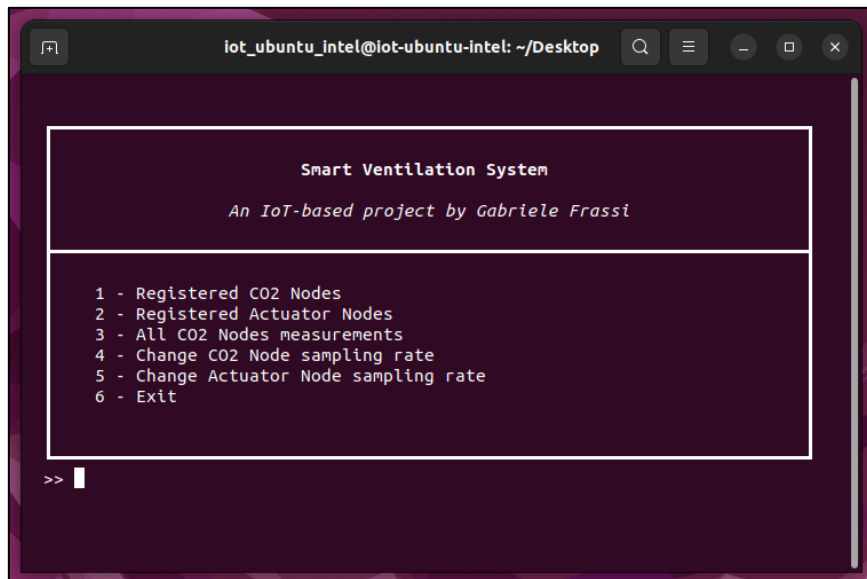
- Each measurements packet will be in the following form
`{"c": "%i", "h": "%i", "op": %i"}`
 where `c` is the measured CO2 (ppm), `h` is the measured humidity (%) and `op` is the computed occupancy prediction (by *TinyML*).
- The Cloud Application will store all measurements in the database.
- The Cloud Application updates the status of each *Actuator Node* by comparing its threshold with the computed occupancy prediction, without requiring any data exchange between the *Cloud Application* and the *Actuator Nodes*.

2.5 User Application

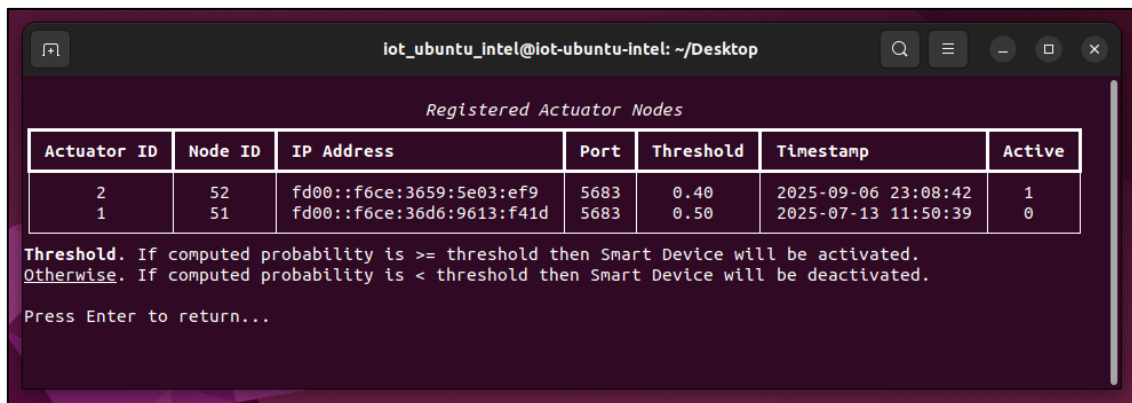
The *User Application* provides the following functionalities:

- view a list of all registered *CO2 Nodes*;
- view a list of all registered *Actuator Nodes*;
- view all stored measurements;
- modify the sampling rate of a *CO2 Node* (by using the resource `/sampling` of the node);
- modify the threshold of an *Actuator Node* (by using the resource `/threshold` of the node).

All functionalities have been implemented in Python, using *CoAPthon*⁵ and *MySQL Connector*⁶ libraries. Also, *console-menu*⁷ and *Rich*⁸ libraries to provide an elegant and simple user interface.



Main page of the User Application



Page with all registered actuator nodes

⁵ <https://github.com/Tanganelli/CoAPthon>

⁶ <https://pypi.org/project/mysql-connector-python/>

⁷ <https://pypi.org/project/console-menu/>

⁸ <https://rich.readthedocs.io/en/stable/introduction.html>

2.6 Database

A MySQL database is deployed in order to store all the necessary information for the functioning of the *Cloud Application* and the *User Application*.

Table CO2_Nodes			
Field	Type	Characteristics	Notes
id	int	PRIMARY KEY auto_increment	
ip_address	varchar(45)		
port	int unsigned		
sampling_rate	int unsigned		
timestamp	timestamp	CURRENT_TIMESTAMP as default value	When registration has been done.

Table Actuator_Nodes			
Field	Type	Characteristics	Notes
id	int	PRIMARY KEY auto_increment	
ip_address	varchar(45)		
port	int unsigned		
threshold	decimal(3, 2)		Value between 0 and 1
co2_node_id	int	FOREIGN KEY REFERENCES TO CO2_Nodes(id)	
timestamp	timestamp	CURRENT_TIMESTAMP as default value	When registration has been done.

Table CO2_Nodes Measurements			
Field	Type	Characteristics	Notes
id	int	PRIMARY KEY auto_increment	
node_id	int	FOREIGN KEY REFERENCES TO CO2_Nodes(id)	
CO2	float		u.o.m: ppm
humidity	float		u.o.m: %
occupancy_prediction	decimal(3, 2)		Value between 0 and 1
timestamp	timestamp	CURRENT_TIMESTAMP as default value	When measurements have been done.

3 DEBUG AND TEST

This project is a highly simplified implementation, as it does not employ real sensors and relies exclusively on the nRF52840 dongle. To account for this simplification and to facilitate debugging, the following functionalities have been developed:

- **Log printing.**

The `printf` (C) and `print` (Python) functions have been widely used to provide real-time feedback on node and application activities. All logs can be viewed in the terminal by running the `make login` command (nodes). The execution of this command is mandatory for *Actuator Nodes* because we have to define the parent CO2 Node.

The image displays four terminal windows from a Linux environment. The top-left window shows a series of log messages: `Measurements have been received from < fd00::f6ce:36a6:c68f:cb04 , 5683 > {'c': '75000', 'h': '2600', 'op': '73'}`, followed by similar messages for different CO2 levels and humidity values. The top-right window shows the process of connecting to a CO2 sensor, including the command `Please write the ID of the CO2 Sensor: 53` and the response `Sent ID: 53`, followed by a successful connection message and a list of smart device activation/deactivation events. The bottom-left window displays a menu titled "Smart Ventilation System" with options: `1 - Registered CO2 Nodes`, `2 - Registered Actuator Nodes`, `3 - All CO2 Nodes measurements`, `4 - Change CO2 Node sampling rate`, `5 - Change Actuator Node sampling rate`, and `6 - Exit`. The bottom-right window shows the output of the `enlearn` command, displaying input values (e.g., `Input: c -> 900 h -> 30`) and corresponding output values (e.g., `Output: op -> 76`).

- **Startup when a button is pressed.**

Both CO2 Nodes and Actuator Nodes begin their activities only after the button on the corresponding nRF52840 dongle is pressed.

- **On-demand sensing.**

In addition to scheduled sensing, the system allows on-demand sensing by pressing the button on the corresponding nRF52840 dongle. This functionality is not intended to represent a realistic scenario.

- **Fake sensing.**

A simple function called `fake_values`, implemented within *CO2 Nodes*, generates input values that are used by `enlearn` to make a prediction. These values are repeated cyclically.