UNIVERSITÀ DI PISA

Internet of Things

The SafeTunnels Service

Table of Contents

Service Introduction

The *SafeTunnels* is an IoT service belonging to the *Smart Transportation/Smart Traffic* application domain aimed at guaranteeing the safety and well-being of drivers in road tunnels (e.g. a motorway mountain underpass) by deploying in the tunnel, or operating environment:

- A set of sensors continuously monitoring the ambient CO_2 density and temperature, whose variations can be attributed to events such as traffic jams, car accidents and fires.
- A set of actuators consisting of industrial fans and emergency lights which, should either the CO_2 density or temperature cross a hierarchy of safety thresholds, are automatically activated in an attempt to restore the system to a safe operating condition as well as alerting transportation authorities and drivers that the tunnel should temporarily be closed or evacuated.

As the service represents a safety-critical, industrial system to meet its stringent *Reliability* and *Low Latency* requirements:

- The devices' states, including whether they are connected and thus reachable from an external network, as well as any error that may occur in their execution, must be promptly propagated to the rest of the system,
- The service includes a locally deployed *Control Module* which, in addition to automatically adjusting the actuators' states according to the current environmental conditions, allows a technician to monitor the overall system as well as manually override the actuators' states.
- While in the developed prototype the smart devices form a Wireless Sensor/Actuator Network (WSN) based on the 802.15.4 communication standard, the use of communication technologies more appropriate to an industrial setting are required in a real-world deployment, with a pre-existing road electrical grid conceived for lighting purposes that can be exploited to implement a Powerline Communication (PLC) system.

Finally, to enable data analysis and decision-supporting systems, all data collected in the operating environment should be propagated to a remote database and presented to operators via a user-friendly graphical interface.

System Operating States

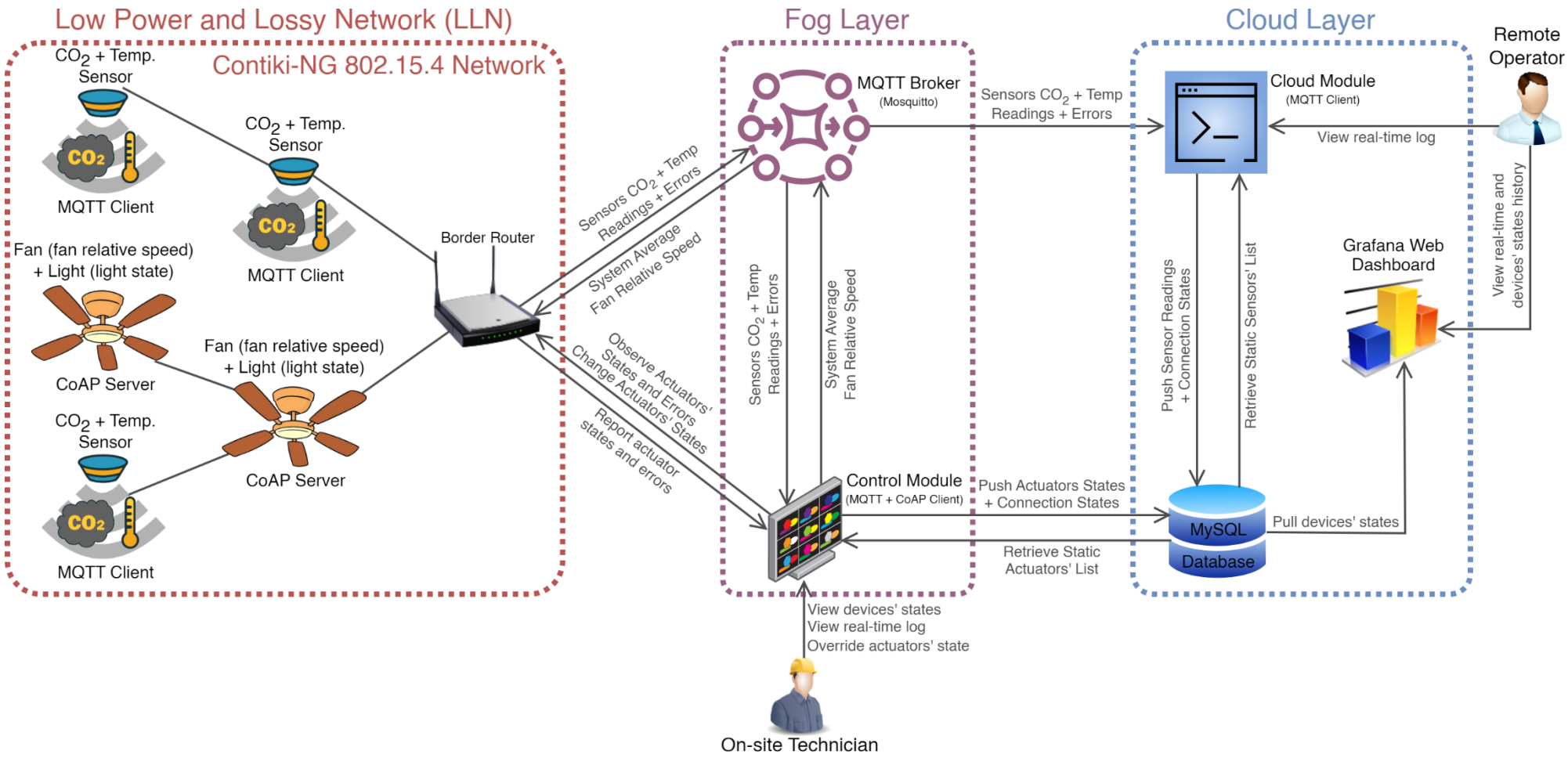
The hazard level to human health of the system's environmental conditions has been classified into four *operating states*, which are determined by the most severe among the latest sensor CO_2 density and temperature readings and are conveyed to operators and user via the actuators' emergency lights:

Operating State	Description	Sensors Quantities' Thresholds
NOMINAL	The environmental conditions are within their nominal levels – the tunnel can be accessed by users	CO_2 density ≤ **2000 ppm** Temperature ≤ **35°C**
WARNING	The environmental conditions are beyond their nominal but still within safe levels – the tunnel can still be accessed by users	**2000 ppm** < CO_2 density ≤ **5000 ppm** **35°C** < Temperature ≤ **40°C**
ALERT	The environmental conditions are at an unsafe level – traffic should be diverted from the tunnel	**5000 ppm** < CO_2 density ≤ **10000 ppm** **40°C** < Temperature ≤ **45°C**
EMERGENCY	The environmental conditions are at a critical level – the tunnel should be immediately evacuated	CO_2 density > **10000 ppm** Temperature > **45°C**

Service Architecture

The service architecture is divided into three layers:

- A *Low Power and Lossy Network (LLN)* layer consisting of the smart devices deployed in the tunnel, including:
  - A set of *"Sensor"* nodes which:
    - Continuously sample the environment's *CO_2 density* in parts-per-million (ppm) and *temperature* in degrees Celsius (°C).
    - Propagate their CO_2 and temperature readings to the rest of the service by publishing them on predefined *topics* using the *Message Queue Telemetry Transport (MQTT)* protocol.
    - Propagate any errors on a predefined MQTT topic.
    - Receive the average fan relative speed in the system for correlated quantities simulation purposes.
  - A set of *"Actuator"* nodes consisting of an industrial fan and a light bulb, which:
    - Expose their *fan relative speed* and *light* states as *resources* using the *Constrained Application Protocol (CoAP)*, allowing external actors, in particular the *Control Module*, to observe and manipulate them.
    - Expose an *"errors"* resource which can be observed by external actors, in particular the *Control Module*, to receive information on errors that have occurred.
  - A *Border Router* node connecting the *LLN* to the *Fog Layer* and thus the rest of the internet.
- A *Fog Layer* (or *Edge Layer*) deployed in a service area adjacent to the tunnel comprising:
  - A *Mosquitto MQTT Broker* routing published MQTT messages to their destinations, in particular:
    - Sensor readings and errors to the local *Control Module* as well as to the remote *Cloud Module*.
    - The average fan relative speed values in the tunnel from the *Control Module* to the sensors.
  - A *Control Module* application which:
    - Receives sensors' readings and errors updates from the local MQTT broker.
    - Observes via CoAP Clients the three resources (fan relative speed, light state, errors) exposed by all actuators in the tunnel.
    - Pushes actuators' state and connection updates into the remote MySQL database.
    - Automatically adjusts the actuators' states according to the tunnel current environmental conditions, or *operating state*.
    - Allows a technician to monitor the state of every device as well as manually override the actuators' states.
- A remotely deployed *Cloud Layer* consisting of:
  - A *Cloud Module* application which:
    - Receives sensors readings and errors updates from the local MQTT broker.
    - Pushes sensor readings and connection updates into the MySQL database.
  - A *MySQL database* storing the devices' connection updates and states time series.
  - A *Grafana Web Dashboard* module pulling data from the MySQL database and displaying it in a user-friendly web interface.


Low Power and Lossy Network (LLN)
Contiki-NG 802.15.4 Network
Fog Layer
Cloud Layer
Remote Operator
CO_2 + Temp. Sensor
MQTT Client
CO_2 + Temp. Sensor
MQTT Client
Fan (fan relative speed) + Light (light state)
CoAP Server
Fan (fan relative speed) + Light (light state)
CoAP Server
CO_2 + Temp. Sensor
MQTT Client
Border Router
MQTT Broker
(Mosquitto)
Sensors CO_2 + Temp Readings + Errors
System Average Fan Relative Speed
Observe Actuators' States and Errors
Change Actuators' States
Report actuator states and errors
Sensors CO_2 + Temp Readings + Errors
System Average Fan Relative Speed
Control Module
(MQTT + CoAP Client)
Sensors CO_2 + Temp Readings + Errors
Cloud Module
(MQTT Client)
View real-time log
View real-time and devices' states history
Grafana Web Dashboard
Push Sensor Readings + Connection States
Retrieve Static Sensors' List
Push Actuators States + Connection States
Retrieve Static Actuators' List
MySQL Database
Pull devices' states
View devices' states
View real-time log
Override actuators' state
On-site Technician

Service Components Descriptions

Low-Power and Lossy Network (LLN) Layer

Sensor

A sensor node's execution can be summarized as per the following state machine:

1) The sensor waits until the LLN's *RPL DODAG* has converged and thus for external hosts to be reachable.
2) The sensor's MQTT client attempts to connect with the MQTT broker and registers its *"last will"* message to be automatically published by the latter on the *"SafeTunnels/sensorsErrors"* topic should it detect the node to have disconnected, with the message being JSON-encoded and structured as:

```
{
 "MAC":     [sensorMACStr],
 "errCode": 0              // Disconnection error
}
```

3) The sensor attempts to subscribe on the *"SafeTunnels/avgFanRelSpeed"* on the MQTT broker to receive the tunnel's updated average fan relative speed values for correlated random sampled quantities generation purposes.
4) As soon as it is connected with the MQTT broker, the node starts sampling its operating environment's CO_2 density in parts-per-million (ppm) and temperature in degrees Celsius (°C) via two separate timers, publishing its readings in JSON format on the *"SafeTunnels/CO2"* and *"SafeTunnels/temp"* topics as:

```
{                                  {
 "MAC": [sensorMACStr],             "MAC":  [sensorMACStr],
 "C02": [C02ReadingInt]             "temp": [tempReadingInt]
}                                  }
```

Other sensor nodes functionalities include:

- For prototyping purposes the sampled CO_2 density and temperature values are randomly generated, with their probabilities at every sampling to increase, decrease, or remain unchanged as well as the variation from their current values taking into account:
  - Their current relative values $\in [0,100]$ with respect to their predefined maximum values.
  - A randomly generated and evolving *"road equilibrium point"* $\in [0,100]$ representing the percentage with respect to the quantities' maximum values at which they would be at the equilibrium without taking into account the tunnel's average fan relative speed.
  - The tunnel's average fan relative speed $\in [0,100]$, a fraction of which is subtracted from the *"road equilibrium point"* so as to determine the quantities' system equilibrium point.
- As a *data reduction energy conservation technique*, sensors avoid publishing a same sampled quantity up to the pre-established publishing timeout at which they would be considered disconnected by the MQTT broker.
- Comprehensive error recovery mechanisms have been implemented so as to manage and possibly recover from conditions such as the sensor being disconnected from the RPL DODAG or from the MQTT broker.
- Errors occurring while the sensor is connected with the MQTT broker are reported by publishing them on the *"SafeTunnels/sensorsErrors"* in the following JSON-encoded format:

```
{
 "MAC":          [sensorMACStr],
 "errCode":      [errCodeEnum],
 "errDscr":      [errDscrStr],        // Optional error description
 "MQTTCliState": [MQTTCliStateEnum]   // Current Sensor MQTT client state
}
```

- The two LEDs available on a sensor are driven as follows:

LED #	Name	Color	Semantic
LED1	**POWER_LED**	**GREEN**	• Enabled while the sensor is powered on.
LED2	**MQTT_COMM_LED**	**BLUE**	• Enabled while the sensor is connected with the MQTT broker. • Blinks when publishing a message to the MQTT broker.

- Pressing the sensor's button causes, at their next sampling, the CO_2 density and temperature to be set to their maximum values for the purpose of testing the system's negative feedback capability of restoring the environment to a safe operating condition.

Actuator

At startup an actuator node initializes a *CoAP Server* exposing to clients the following *observable* resources:

- A *"fan"* resource allowing the retrieval and/or observation via a GET request as well as the modification via a PUT request of the actuator's fan relative speed ∈ [0,100], whose state is returned in JSON format as:

```
{"fanRelSpeed": [fanRelativeSpeedInt]}
```

- A *"light"* resource allowing the retrieval and/or observation via a GET request as well as the modification via a PUT request of the actuator's emergency light state, which, as previously discussed, is normally associated with the current system's operating state and displayed via the *LIGHT_LED* (see later).

Light State	Associated Operating State	Description
LIGHT_OFF	**NOMINAL**	The emergency light is **OFF**.
LIGHT_ON	**WARNING**	The emergency light is **ON**.
LIGHT_BLINK_ALERT	**ALERT**	The emergency light is **blinking slowly**.
LIGHT_BLINK_EMERGENCY	**EMERGENCY**	The emergency light is **blinking fast**.

Light state that is represented as an integer and returned in JSON format as:

```
{"lightState": [lightStateEnum]}
```

- An *"errors"* resource allowing clients to be notified of errors associated with received CoAP requests, which are returned in the following JSON format:

```
{
 "errCode":   [errCodeEnum],
 "errDscr":   [errDscrStr],      // Optional error description
 "clientIP": [clientIPv6Addr]   // The IP address of the client
}                                  whose request caused the error
```

Other actuator nodes functionalities include:

- The CoAP responses status codes have been appropriately set depending on whether a request was successful (2.xx), failed due to a client error (4.xx) or a server error (5.xx).
- A human-readable description of the error raised by a CoAP request is included in the payload of the returned CoAP responses.
- The two LEDs available on an actuator are driven as follows:

LED #	Name	Color	Semantic
LED1	**LIGHT_LED/ POWER_LED**	**GREEN**	• Briefly blinks when the actuator is powered on. • As with the actuator's emergency light when connected with the RPL DODAG.
LED2	**FAN_LED/ COMM_LED**	**BLUE**	• Briefly blinks when the actuator is connected with the RPL DODAG. • Blinks at a fixed rate proportional to the current fan relative speed.

- Pressing the actuator's button simulates an external CoAP client's intervention changing both the actuator's fan relative speed and light state to a random, new value.

Fog Layer

MQTT Broker

The topics and semantics of MQTT messages exchanged in the service are summarized below:

Topic	Publishers	Subscribers	Semantics
SafeTunnels/CO2	Sensors	Control Module, Cloud Module	Report CO_2 density readings
SafeTunnels/temp	Sensors	Control Module, Cloud Module	Report temperature readings
SafeTunnels/sensorsErrors	Sensors	Control Module, Cloud Module	Report sensors errors, including their disconnection
SafeTunnels/avgFanRelSpeed	Control Module	Sensors	Send sensors the system's current average fan relative speed

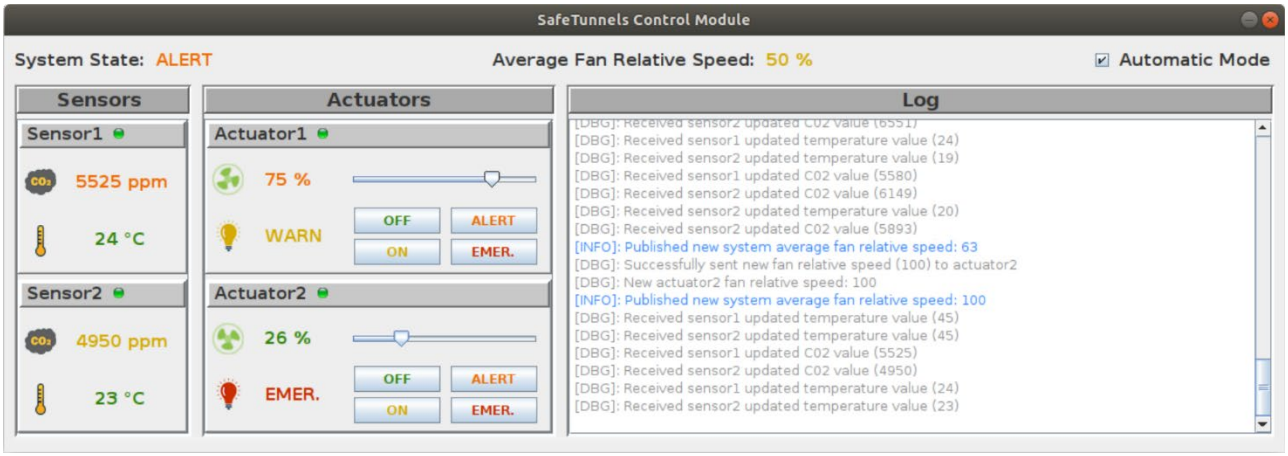
Control Module

Developed as a *Java 11 GUI* application using the *Swing* library, the Control Module's functionalities are summarized as:

- At startup it retrieves from the remote *MySQL Database* the static lists of *{MAC, sensorID}* and *{MAC, actuatorID}* of the sensors and actuators deployed in the tunnel.
- It instantiates a *PAHO MQTT Client* module and subscribes on the *MQTT Broker* on the sensors' quantities readings and errors topics.
- Updates the system's average fan relative speed computed among all connected actuators and publishes it on the MQTT *"SafeTunnels/avgFanRelSpeed"* topic.
- For every actuator it instantiates a *Californium CoAP Client* for observing and sending requests to each of its three exposed resources (*"fan", "light", "errors"*), clients that are also preliminarily and periodically used to check whether the actuator is online by sending it a *CoAP Ping*.
- Pushes every actuator connection and quantities' state update into the remote *MySQL Database*.
- Updates the system's *operating state* based on the most severe CO_2 and temperature sensor readings as previously discussed.
- By default, automatically adjusts the actuators' fan relative speeds and light states according to the current operating state as:

Operating State	Associated Light State	Associated Fan Relative Speed
NOMINAL	**LIGHT_OFF**	**0%**
WARNING	**LIGHT_ON**	**30%**
ALERT	**LIGHT_BLINK_ALERT**	**60%**
EMERGENCY	**LIGHT_BLINK_EMERGENCY**	**100%**

- Displays a graphical user interface allowing a local technician to:
  - View the system's current *operating state* and *average fan relative speed*.
  - View the connection and quantities' states of all sensors and actuators in the system.
  - Toggle the system's automatic actuators' adjustment mode.
  - View the real-time system log, whose messages are classified and can be filtered according to their severity (debug, info, warning, error, fatal).
  - Manually override the actuators' fan relative speeds and light states.



Cloud Layer

Cloud Module

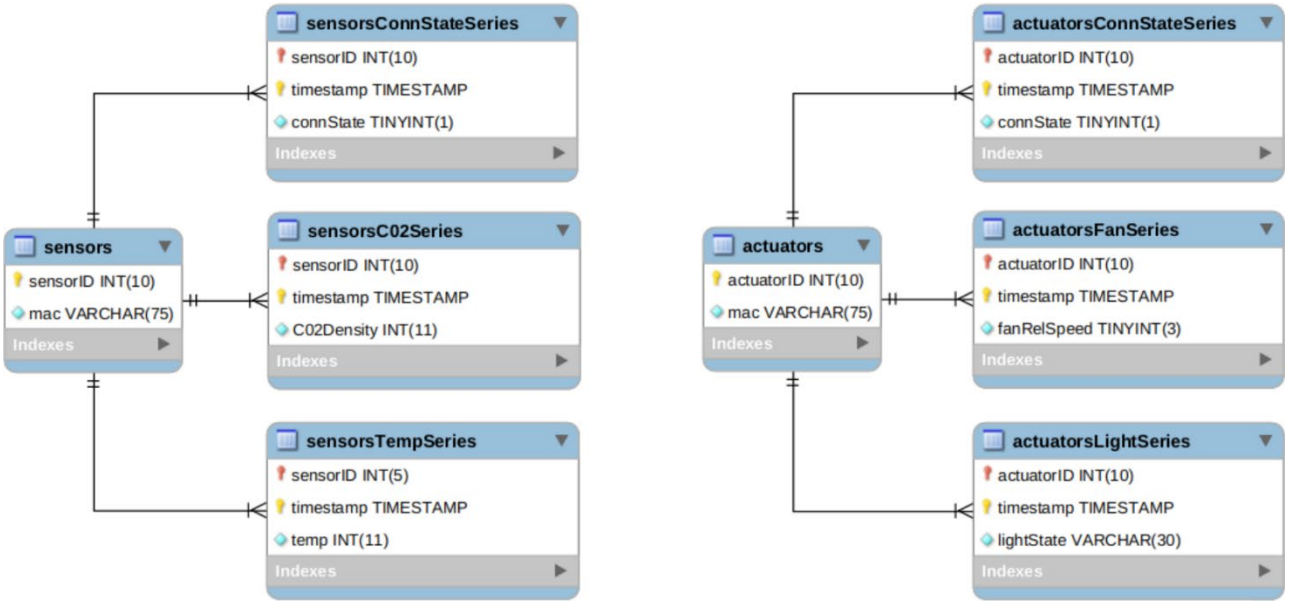
Developed as a *Java 11 command-line interface* application, the Cloud Module's functionalities are summarized as:

- At startup it retrieves from the *MySQL Database* the static lists of *{MAC, sensorID}* of the sensors deployed in the tunnel.
- It instantiates a *PAHO MQTT Client* module and subscribes on the *MQTT Broker* on the sensors' quantities readings and errors topics.
- Pushes every sensor connection and quantities' state update into the *MySQL Database*.
- Displays a real-time system log, whose messages are classified and can be filtered according to their severity (debug, info, warning, error, fatal).

```
[DBG]: 2 sensors were retrieved from the database <sensorID,MAC>:
[DBG]: |- <2,f4:ce:36:69:a9:6a:5b:65>
[DBG]: |- <1,f4:ce:36:ed:61:11:36:51>
[DBG]: Sensors MQTT client handler initialized
[DBG]: MQTT Client connected with the MQTT broker @tcp://127.0.0.1:1883, subscribing on topics
[DBG]: MQTT Client successfully subscribed on the sensors' topics
[INFO]: Cloud Module successfully initialized
[WARN]: sensor1 appears to be offline (pushed into the database)
[WARN]: sensor2 appears to be offline (pushed into the database)
[INFO]: sensor1 is now online (pushed into the database)
[DBG]: Pushed sensor1 updated C02 density (5142) into the database
[DBG]: Pushed sensor1 updated temperature (27) into the database
[DBG]: Pushed sensor1 updated C02 density (5744) into the database
[DBG]: Pushed sensor1 updated temperature (26) into the database
[DBG]: Pushed sensor1 updated C02 density (5948) into the database
[INFO]: sensor2 is now online (pushed into the database)
[DBG]: Pushed sensor2 updated C02 density (7152) into the database
[DBG]: Pushed sensor1 updated temperature (25) into the database
[DBG]: Pushed sensor2 updated temperature (17) into the database
```

MySQL Database

The MySQL database was designed to optimize the read and write operations of the sensors' and actuators' quantities as *time series* as follows:



SafeTunnels Database Tables				
Table Name	**Description**	**Attributes**		
		Name	**Type**	**Description**
sensors	Sensors' static list	*sensorID*	INT(10)	Sensor's unique ID
		mac	VARCHAR(75)	Sensor's unique MAC
sensorsConnStateSeries	Sensors' connection states time series	*sensorID*	INT(10)	Sensor's unique ID
		timestamp	TIMESTAMP	Reading timestamp
		connState	TINYINT(1)	Sensor connection state
sensorsCO2Series	Sensors' CO_2 readings time series	*sensorID*	INT(10)	Sensor's unique ID
		timestamp	TIMESTAMP	Reading timestamp
		CO2Density	INT(11)	CO_2 density reading
sensorsTempSeries	Sensors' temperature readings time series	*sensorID*	INT(10)	Sensor's unique ID
		timestamp	TIMESTAMP	Reading timestamp
		temp	INT(11)	Temperature reading
actuators	Actuators' static list	*actuatorID*	INT(10)	Actuator's unique ID
		mac	VARCHAR(75)	Actuator's unique MAC
actuatorsConnStateSeries	Actuators' connection states time series	*actuatorID*	INT(10)	Actuator's unique ID
		timestamp	TIMESTAMP	Reading timestamp
		connState	TINYINT(1)	Actuator connection state
actuatorsFanSeries	Actuators' relative fan speed time series	*actuatorID*	INT(10)	Actuator's unique ID
		timestamp	TIMESTAMP	Reading timestamp
		fanRelSpeed	TINYINT(3)	Actuator fan relative speed
actuatorsLightSeries	Actuators' light state time series	*actuatorID*	INT(10)	Actuator's unique ID
		timestamp	TIMESTAMP	Reading timestamp
		lightState	TINYINT(3)	Actuator light state

Graphana Web Dashboard

By fetching data from the *MySQL Database* the developed *Graphana* dashboard presents to the user both the latest and the time series in a customizable interval of all the sensors and actuators connection and quantities' states: