University of Pisa

Department of Information Engineering

Internet of Things

SPARK-GRID

Smart Power and Adaptive Real-Time Knowledge Grid

Student

Davide Vigna

Academic Year: 2023-2024

Contents

1 Introduction

A smart grid is an advanced electrical system that exploits the Internet of Things (IoT) to monitor and manage the generation, distribution, and consumption of electricity inside a city. It can be seen as a virtual grid operating in tandem with the physical electrical infrastructure, continuously monitoring each member's access to the collective resource of electricity. The advent of smart grids has revolutionized the energy sector, providing considerable benefits in energy management and savings increasing the efficiency in electricity delivery to end-consumers and reducing the transmission-related waste. In today's climate-change landscape, where the energy sector faces increasing pressure to curb fossil fuel use, smart grids play an important role for the effective management of renewable resources. Traditionally, electricity was transmitted from production plants to end-users through dispatching systems. Today, individuals can actively contribute to energy production through renewable energy systems, such as photovoltaic installations. The consumers, now known as a *prosumers*, directly use the energy they generate, while any surplus energy is fed back into the public electricity grid. This shift has required a bi-directional energy flow management within the grid and a continuous balance between the energy produced and the energy demanded is needed at any given time. A smart grid is typically divided into different areas within a city, each with its own priority level, which affects how the system should responds in case of faults or power outages. Some areas, like residential neighborhoods, can handle short down-times without serious consequences. However, other areas, such as hospitals, cannot afford any electrical interruptions. As a result, these critical areas are managed with higher priority to ensure they always have electricity, even in emergency situations. This kind of prioritization allows the smart grid to have different behaviour depending on the situation, so, it is become normal having a periodic variation in the flow of electricity. For instance, during summer, high temperatures can lead to a surge in air conditioner usage, causing increased demand and potential blackouts in the city. Conversely, on particularly sunny days, in which the consumption is very limited, there may be an excess of energy production from photovoltaic systems increasing the voltage on transformers, so it is required an intelligent management to handle these situations in order to preserve the life of the grid. This makes a continuous monitoring of the grid health and its components a critical priority to ensure a correct, reliable and efficient behaviour of the entire system.

Errors are an unavoidable aspect of any system, and the costs associated with addressing them can be significant. In certain situations, human intervention can be challenging and hazardous, often leading to obstacles and delays. This situations are particularly costly in a smart grid where time is gold. One effective strategy to minimize or prevent these errors is to leverage artificial intelligence whenever possible, leaving the human intervention only for essential task. In the context of a smart grid, there are numerous applications for AI. For instance, it can facilitate the quick identification of faults in specific sections of the network using data from dedicated sensors, allowing for autonomous corrective actions. Additionally, AI can perform in-depth analyses and forecasts regarding future energy consumption or demand from users, enhancing overall system efficiency.

1.1 Spark-IoT: project objectives and motivations

The aim of this project is to develop an IoT application focused on a custom smart grid use case that is detailed in the next section.

The acronym SPARK stands for *Smart Power and Adaptive Real-Time Knowledge*, emphasizing the grid's capability to intelligently manage power consumption and production data. It also highlights the system's ability to make autonomous decisions, dynamically configuring the grid based on real-time data received from sensors.

Power is a crucial quantity to calculate and measure, as it forms the basis for electricity billing. The system enables prosumers to manage their bills by providing detailed control over their consumption and production, while also allowing energy providers to efficiently monitor and manage the smart grid. It is also able to alert the prosumers when the energy consumption is too high or too low, providing suggestions on how and when to adjust their energy load for a more efficient usage.

Another key objective for the smart grid manager is to monitor voltage and current levels across the entire grid. Areas near transformers, which regulate voltage to ensure homes receive electricity within safe and proper ranges, are often more prone to fluctuations. Maintaining voltage and current within specific safety thresholds is essential for the reliable operation of the network, preventing transmission losses and overloads, while ensuring safety for users. Therefore, it is crucial that the smart grid in this project can automatically disconnect part of the grid in real-time in case of a fault that poses a safety risk, while still ensuring uninterrupted power supply to areas requiring continuous energy usage. For this task, the integration of artificial intelligence is essential, minimizing the need for human intervention to only rare or exceptional cases.

Finally, it is crucial to allow the smart grid manager to remotely control all of the previous aspects, allowing for adjustments to be made without the need for physical intervention on the place.

2 Use case description

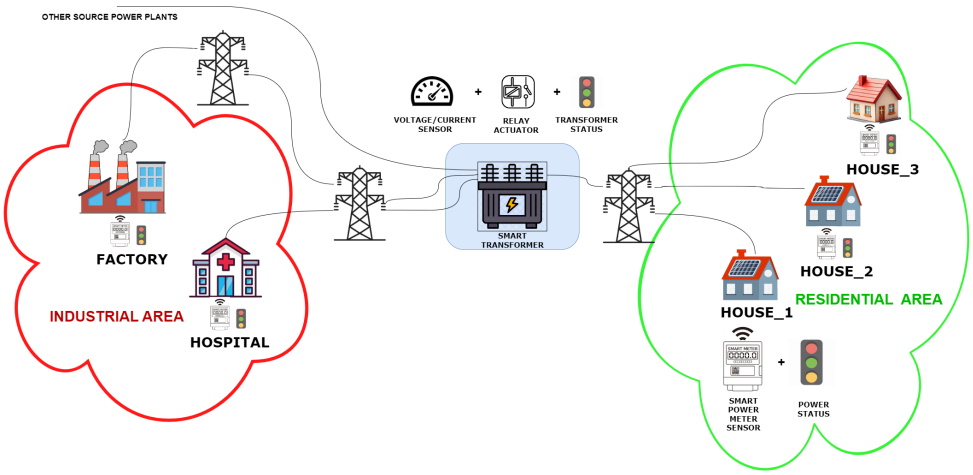

Figure 1: Smart Grid structure

The image illustrates a real-world scenario where the smart grid could be implemented. For simplicity, not the entire city is shown, but only the key components necessary to explain the overall behavior and use cases. The grid is divided into two main areas:

- **Industrial Area** (in red) where critical activities are executed during the day, so no interruption of energy is tolerated; in the figure are present a factory where 24/7 working is performed and an hospital which often human lifes depends on some machineries that requires continuous powered.
- **Residential Area** (in green) where daily activities take place, and temporary outages are more tolerable. The image highlights that some houses, like house_3, are traditional and lack renewable energy systems, while others have energy production systems, acting as prosumers.

Each building in the city is equipped with a smart power meter that periodically measures the voltage and current consumed or generated, calculating the corresponding power. This data is also interpreted by an actuator, which could be integrated into a smart home system within the same building. The actuator notifies the prosumer when the building is approaching its maximum contracted power consumption or generating renewable energy, helping to optimize energy usage (the semaphore in the picture). This power information is regularly transmitted to a remote application, offering a dual benefit both for prosumer and energy provider.

At the center of the two subgrids is present a smart transformer, responsible for ensuring the safe distribution of electricity to both sections. This transformer features a three-phase design and is equipped with a smart meter to monitor voltage and current levels on each phase. Additionally, it houses a load-balancing relay that acts as an actuator. The aim of the relay is to use the data coming from the sensor and exploing artificial intelligence to identify the possible fault types, taking autonomous decisions to balance the load of work and restoring the transformer to its normal operating state. It has also the capability to disconnect parts of the grid (the less constrained one - residential area) in case of an hazardous fault and next, once the problem is solved, reconnect it autonomously. The measured values are regularly transmitted to a remote application, where the data are recorded and can be analyzed to help prevent future overloads or black-outs.

3 System Design and Components

This project leverages a wireless sensor network (WSN) to develop the use case described in the previous section. To achieve this, three Nordic Semiconductor **NRF52840 Dongle** chips are employed to implement the various components of the system in the following way.

1. **Border Router Device** It connects the IoT sensor network with the Internet and therefore enables communication with the remote user application.

2. **Smart Power Device** It is designed to be integrated in a Smart House equipped with a photovoltaic system (like House_1 in the previous image). The device acts as a smart meter **sensor** for measuring voltage, current, and power factor related to the domestic user's consumption, while also detecting the current produced by a photovoltaic system. It calculates the actual instantaneous power consumption in Watt and transmits this data to remote application. Since power is a derived quantity, the device calculates it using the following formula:

$$P = \sqrt{3} \cdot V \cdot I \cdot FP$$

where V is the voltage, I is the total current, and FP is the power factor.The total current I is given by:

$$I = (I_{\text{consumed}} - I_{\text{produced}})$$

where the current produced by the photovoltaic system is also taken into account. The power P will be negative in the case of energy re-injection into the grid.
It is assumed that this device can also serve as an **actuator** to notify the user when specific scenarios occur. The following scenarios are possible, each of which is indicated by a specific LED configuration:

- YELLOW LED (on) - *home electrical system deactivated*; this can happen if the electricity grid management company decides to voluntarily deactivate access to the building's electricity grid (remotely) e.g. due to non-payment or temporary fault;
- GREEN LED (on) - *green energy*; the energy produced is greater than the energy consumed; it suggests the user to insert loads to optimize the use of the energy produced which would otherwise be redistributed to the grid at a lower price than the distribution company would pay for it;
- BLUE LED (on) – *normal operation*; energy consumption exceeds production but remains within the predefined range (less than (MAX_POWER − 1) kW). The user is consuming energy normally and requires additional power from the external grid to operate.
- RED LED (blinking) - *power limit warning*; energy consumption is in the range [(MAX_POWER − 1) kW, MAX_POWER kW).The user is alerted that if the energy demand continues to rise, the electrical system may shut down.
- RED LED (fixed on) - *maximum power exceeded*; the power load has surpassed the maximum allowable limit, resulting in the deactivation of the home system. The user must manually restore the internal network by disconnecting the connected loads. Both production and consumption are put at 0 kW.

To simulate user interaction, the button on the Dongle is used as follows:

- pressing the button increases the household load by elevating the current consumption
- holding the button down for 5 seconds disconnects all the previously attached loads taking the current consumed to 0

3. **Smart Transformer Device** It was designed to be integrated into a distribution station on a three-phase transformer used by the distribution company for energy dispatching. The device works as a voltage and current **sensor** for each phase of the transformer, periodically sending the measured values to the remote application. The device operates also as **actuator** which incorporates a neural network to minimize human intervention. It can detect different types of failures, determining which of the three phases is affected, and attempt to resolve the issue by adjusting parameters via a simulated Load Balancing Relay. The device's LEDs indicate the status of the transformer based on the following logic:

    - GREEN LED (on) – *normal status*; no anomalies detected; all three-phase transformer parameters are within normal ranges;
    - YELLOW LED (on) – *soft anomaly detected*; a potential issue has been identified, and the system is attempting an automatic repair by adjusting the relevant parameters;
    - BLUE LED (blinking) – *hard anomaly detected*; the detected issue is complex and may take longer to resolve simply adjusting parameters, but it is still repairable;
    - RED LED (on) – *critical anomaly detected*; the issue cannot be resolved automatically unless all residential utilities are disconnected to prevent severe damage to the transformer or a potential neighborhood-wide blackout. The three-phase transformer parameters are then corrected autonomously to the safe levels, allowing industrial utilities to continue operating without any interruption and later, the power is restored to the residential area.

    To simulate the generation of anomalies at the transformer level, the button on the Dongle is used in the following way:

    - by holding down the button, the system can alter the voltage and current values across the transformer's various phases, simulating an ongoing anomaly. For simplicity, for each described scenario a static range of values has been selected to be let correctly interpret by the neural network on the actuator.

It is important to note that on these devices run the Contiki-NG operating system, which is well-suited for low-power and lossy devices. Additionally, they natively support the 802.15.4 protocol at the link layer, as the communication range between these devices is limited to less than 50 meters. However, in this specific scenario of a smart grid, where obstacles are common and the distance between devices may exceed the communication range, a long-range communication protocol such as LoRaWAN should be implemented to ensure reliable connectivity and different devices should be used. For the purposes of the demonstration, and considering the available devices, the default configuration has been maintained.
The system uses the Constrained Application Protocol (CoAP) for optimized communication, leveraging observe relationships to decrease latency and boost responsiveness. The devices work as CoAP client/server in order to implement the functionalities of the use case.
On the cloud side, the system manages smart device registration and hosts an application that enables remote users to interpret sensor data, which is accurately stored in a database, and issue commands to various actuators. Users can also access these data through Grafana, a web application for graphical visualization and data analysis.

The following page shows an illustration, summarizing the entire system described. In the next section are presented in detail the resources and the functionalities offered by each mentioned component.

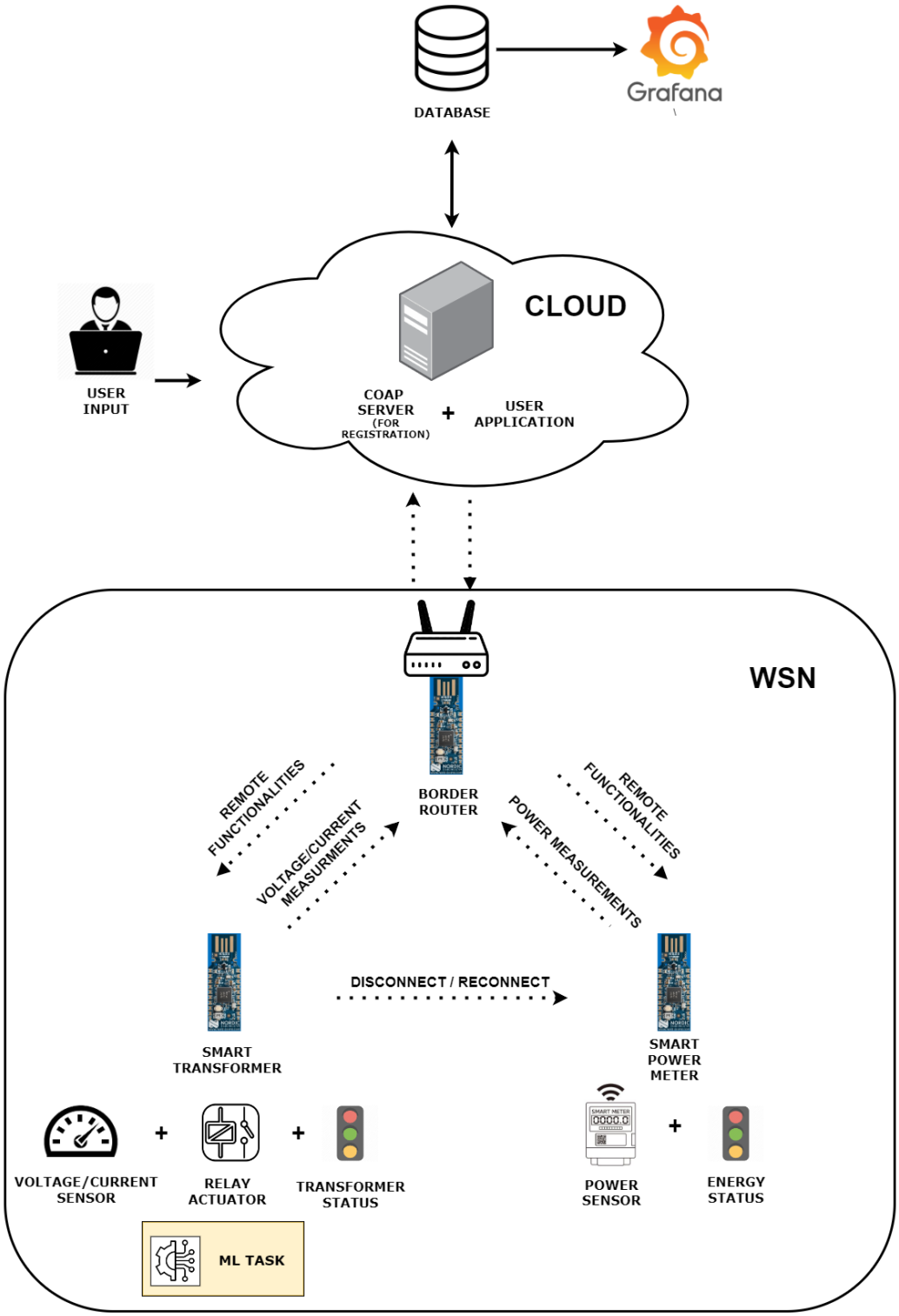

Figure 2: System Architecture

3.1 Smart Power Meter

It serves as a CoAP server for the remote application and offers the following resources:

Resource	Methods	Observable	Description
/power	GET	False	Instant power value from the sensor
/power_obs	GET	True	Power value updates every minute or with significant change ($\geq$ 1 kW)
/status	PUT	False	Status (enabled/disabled) of the home's electrical system, modifiable remotely
/max_power	PUT	False	Maximum power capacity of the electrical system, adjustable remotely; triggers alerts for excessive power usage

Table 1: Resources exposed by the Smart Power Meter

3.2 Smart Transformer

It serves as a CoAP server to remote application and as CoAP client for the Smart Power Meter, in case of *critical anomaly detected* status, in order to disconnect remotely the residential area (House_1). It also incorporates a neural network to identify the types of faults, which will be detailed in the next section. It offers the following resources:

Resource	Methods	Observable	Description
/transformer_state_obs	GET	True	Current and voltage measurements values for the three phases of the transformer, updated every 2 seconds for real-time visibility
/transformer_settings	PUT	False	The adjustments to be applied to each phase of the transformer through the Load Balancing Relay.

Table 2: Resources exposed by the Smart Transformer

3.3 Border Router

Functions as a key link between the wireless sensor network (WSN) and the internet. It allows users to remotely access and control the system, turning local sensor data into useful information on the remote application. This connection ensures smooth communication, keeping everything in sync. It consists of the existing border router implementation found in the Contiki-NG example folder. No customization or additional features are implemented.

3.4 Remote CoAP Server

It resides in the cloud and operates continuously, monitoring the measurements sent periodically by sensors. Each time a new measurement is received, it is promptly stored in a remote database. It only exposes one resource:

Resource	**Methods**	**Observable**	**Description**
/device_registration	POST	False	It registers the IoT device on the database before it starts working

Table 3: Resources exposed by the Remote Coap Server

The Smart Power Meter and Smart Transformers initially work as clients. When an IoT device becomes active, it first sends a registration request to the remote CoAP Server using the specified resource. If the device is not already registered, the CoAP Server creates a new record in the database, otherwise it retrieves the last registration record. Before sending a confirmation, the CoAP server initiates an observation relationship on a specific resource exposed by the device requesting registration. Finally it answers with a confirmation message and for a Smart Power Meter, it also includes two additional information:

1. the building's status (enabled/disabled)
2. the maximum allowed power for that building

3.5 User Application

This is a simple Java Console Application that allows users to remotely manage the functionalities of the use case scenario. The following is a list of permitted operations.

1. Control Energy Supply to House_1: this simple function allows to connect/disconnect by remote the specific building thought the resource /status of the Smart Power Meter.
2. Adjust Maximum Power for House_1: this function gives to the user the possibility to change his/her maximum power achievable contracted with the energy provider thanks to the /max_power resource of the Smart Power Meter.
3. Real-Time Power Monitoring for House_1: this functionality is designed to have an instantaneous view of the power produced/consumed by the building; it does not use the database, instead it contacts directly the IoT device thought the /power resource.
4. Daily consumption for House_1: it providers a detailed summarization of the power measurements observed during the current day on that specific building; it uses just the information stored on database in combination with specific queries to extract daily useful information.
5. Current Status of Smart Transformer: this function retrieves the most recent status measurement of the transformer from the database. Given the short interval between updates, the information accurately reflects the transformer's current condition without needing to directly query the device.
6. Changing Smart Transformer Relay Settings: this allows to the energy provider to indicate the variations of current and voltage to be applied on the Load Balancing Relay actuator in order to solve remotely some crucial situation (e.g. correct manually a fault).
7. Exit: terminate the user application in a safety way.

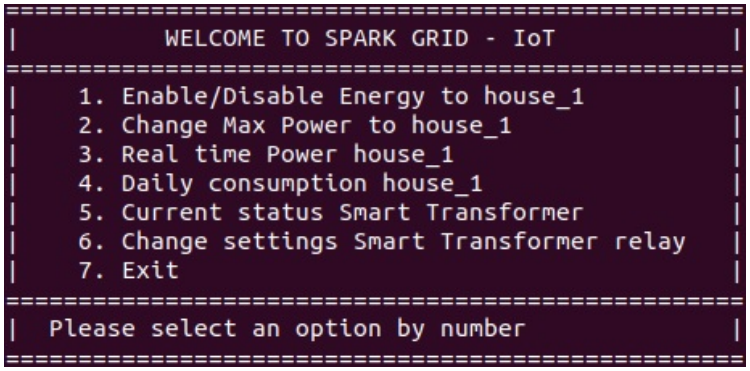

Figure 3: Screenshot of the user application

3.6 Database

This project employs a MySQL relational database to store sensor measurements, ensuring data is well-organized and accessible. Proper data structuring is crucial for efficient retrieval and integration with other applications such as Grafana, optimizing performance and usability. Only three tables are required to manage all the information efficiently.

3.6.1 Table: IoT Devices

This table stores information for all registered IoT devices. Although it would be more accurate to use separate tables for different sensor types, all sensor data is consolidated here, distinguished by the "type" attribute. The structure of the table is as follows.

Field	Data Type	Constraints
ID	INT	AUTO_INCREMENT, PRIMARY KEY
FULL_NAME	VARCHAR(30)	UNIQUE, NOT NULL
TYPE	INT	NOT NULL, DEFAULT 1
ALIAS	VARCHAR(30)	NOT NULL
IP_ADDRESS	VARCHAR(42)	NOT NULL
STATUS	BOOLEAN	NOT NULL, DEFAULT TRUE
MAX_POWER	DECIMAL(7,2)	NOT NULL, DEFAULT 0

Table 4: Structure of the iot_devices Table

Description:

- Id: unique device ID
- Full_name: full name of the device
- Type: device type identifier, 1 for Smart Power Meter, 2 for Smart Transformer
- Alias: device alias name, used just for a more human readable name
- Ip_address: device IP address
- Status: building status (active=true/inactive=false), true for Smart Transformer
- Max_Power: maximum power allowed (kW), always 0 for Smart Transformer

3.6.2 Table: Smart_Power_Meter_Measurements

This table records the details of a single measurement captured by a smart power meter installed in a specific building.

Field	Data Type	Constraints
ID	INT	AUTO_INCREMENT, PRIMARY KEY
ID_DEVICE	INT	NOT NULL, FOREIGN KEY (iot_devices.ID)
POWER	DECIMAL(7,2)	NOT NULL, DEFAULT 0
TIMESTAMP	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP

Table 5: Structure of the smart_power_meter_measurements Table

Description:

- Id: unique measurement ID
- Id_device : reference to the device ID
- Power: measured power in kW
- Timestamp: the time of the storage of the measurement in the database

3.6.3 Table: Smart_Transformer_Sensor_Measurements

This table contains the information related to a single measure executed by a smart meter on a specific three-phase transformer. This table plays a crucial role in maintaining a comprehensive history of various measurements over time. By storing this data, it not only enables tracking and analysis of trends related to the transformer's performance but also facilitates the improvement of the artificial intelligence learning model deployed on the IoT node. Having the latest measurements data ensures that whenever the model will be updated, it remains accurate and reliable, allowing for better fault detection, enhancing the efficiency and longevity of the IoT infrastructure.

Field	Data Type	Constraints
ID	INT	AUTO_INCREMENT, PRIMARY KEY
ID_DEVICE	INT	NOT NULL, FOREIGN KEY (iot_devices.ID)
STATE	INT	NOT NULL, DEFAULT 0
IA	DECIMAL(7,2)	DEFAULT 0
IB	DECIMAL(7,2)	DEFAULT 0
IC	DECIMAL(7,2)	DEFAULT 0
VA	DECIMAL(7,2)	DEFAULT 0
VB	DECIMAL(7,2)	DEFAULT 0
VC	DECIMAL(7,2)	DEFAULT 0
TIMESTAMP	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP

Table 6: Structure of the smart_transformer_sensor_measurements Table

Description:

- Id: unique measurement ID
- Id_device: reference to the device ID
- State: type of fault in the transformer; 0 indicates no faults, while any other value signifies a fault condition
- Ia: current measurement for phase A
- Ib: current measurement for phase B
- Ic: current measurement for phase C
- Va: voltage measurement for phase A
- Vb: voltage measurement for phase B
- Vc: voltage measurement for phase C
- Timestamp: the time of the storage of the measurement in the database

3.7 Grafana

Grafana is a powerful data visualization tool that pulls information from the MySQL database to create real-time dashboards, allowing operators to monitor current system conditions and analyze historical data using a variety of graph types. The last section outlines the implemented dashboards that meet the needs of both prosumers and energy grid provider.

4 Artificial Intelligence Model

In this project, artificial intelligence is used on the Smart Transformer node to detect potential fault scenarios, identifying the root cause, and adjusting the transformer parameters to restore it to a safe operating condition, when possible. The dataset used to build a model can be found at link `[1]`. This dataset contains data of a Distribution Transformer similar to a generic three-phase transformer. The attributes of this dataset are the values of voltage and current for each phases and four additional columns (G,C,B,A) representing different kind of fault with the following meaning:

- **[0 0 0 0]** - No Fault
- **[1 0 0 1]** - LG fault (Between Phase A and Gnd)
- **[0 0 1 1]** - LL fault (Between Phase A and Phase B)
- **[1 0 1 1]** - LLG Fault (Between Phases A,B and ground)
- **[0 1 1 1]** - LLL Fault(Between all three phases)
- **[1 1 1 1]** - LLLG fault(Three phase symmetrical fault)

4.1 Preprocessing

The following operations are applied on the entire dataset in order to achieve a better accuracy in the final model.

1. Data Cleaning: all the record with missing values in any columns are removed.
2. Data Transformation: the four additional columns are merged in one only column called *FaultType* which assigns a value of 0 for no fault and uses increasing numbers to indicate the severity of the fault. LL fault case is never present in the dataset.
3. Data Analysis and Outliers Removal: many values in each class for each attribute fall outside the expected ranges; outliers are detected and removed to ensure the classifier operates with clearer and more defined value ranges, improving its performance.
4. Data Re-balancing: to fix the dataset imbalance from earlier steps, under-sampling is applied to ensure each class has an equal number of records, preventing the classifier from favoring one class over others.
5. Data Splitting: to ensure consistent data for testing the model, the dataset is split into 80% training and 20% testing, while maintaining a balanced distribution of classes in each set.

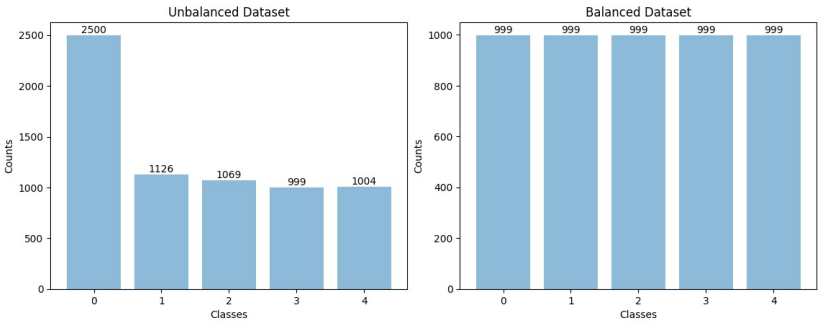

Figure 4: Dataset after re-balancing

4.2 Machine Learning models: Decision Tree and Random Forest

The model will be deployed on an edge device and must operate autonomously within the limited capacity of an NRF52840 Dongle, so the first model analyzed is a simple decision tree. This model adopts a statistical approach to generate a set of decision rules from given data, which can also be used for fault correction after detection (data explainability). In order to exploit all the training set, a 10-Fold-Cross Validation procedure is executed in the training phase and the average value is used to compare it with the evaluation of test data in order to understand if there is too much difference between test and training.
The accuracy on test dataset is about 83% and the generated tree structure is quite complex.

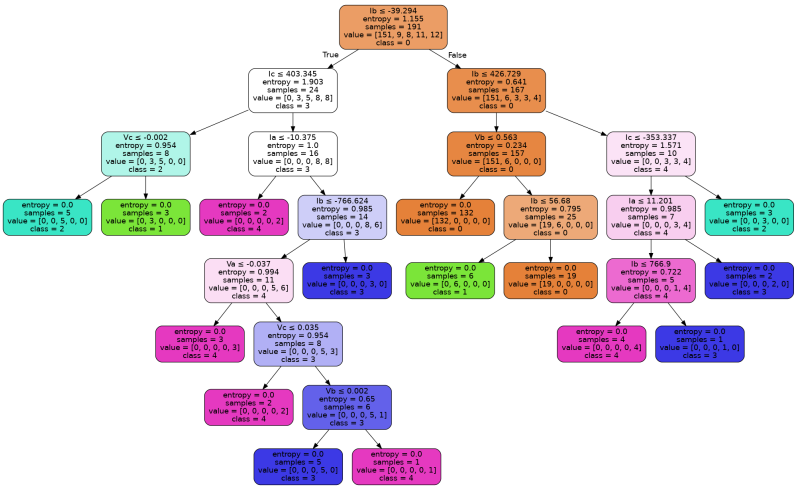

Figure 5: This picture shows just a part of the generated decision-tree

To enhance the accuracy, the Random Forest ensemble technique is employed, combining 100 different decision trees (estimators). However, this approach sacrifices some explainability, as decisions are based not only on a single tree but a forest of 100 trees. The previous procedure is then repeated. The result obtained is worse then the previous one with a value of accuracy equal of 81%. It also requires significantly more storage space, occupying approximately 9.5 MB compared to just 120 KB for the simpler decision tree model.

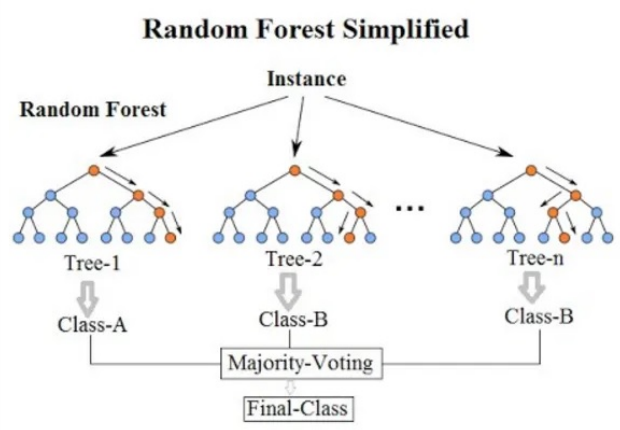

Figure 6: Random forest generic representation

4.3 Deep Learning model: Feed Forward Neural Network

An 83% accuracy is a solid result, but it's still not enough. This level of accuracy suggests that some faults may go undetected, potentially compromising the smart transformer's lifespan and leading to more frequent human intervention than necessary. To achieve a better performance a deep neural network is employed though its primary drawback is its "black box" nature, limiting interpretability. To determine the optimal neural network configuration for this problem, a hyper-parameter tuning through random search is conducted on the following components:

Parameter	**Values**
Neurons in the first layer	32 to 256 (in steps of 32)
Neurons in the second layer	256 to 512 (in steps of 32)
Learning rate	{1e-2, 1e-3, 1e-4, 5e-5, 1e-5}
Activation function	{'relu', 'tanh', 'sigmoid'}

Table 7: Hyper-parameter Search Space

The best configuration obtained is the following one:

Parameter	**Value**
Neurons in the first layer	64
Neurons in the second layer	384
Learning rate	1e-4
Activation function	tanh

Table 8: Hyper-parameter Best Configuration

With this configuration and 100 epochs, the model achieves about 99% accuracy and occupies just 350 KB, three times the size of a decision tree.

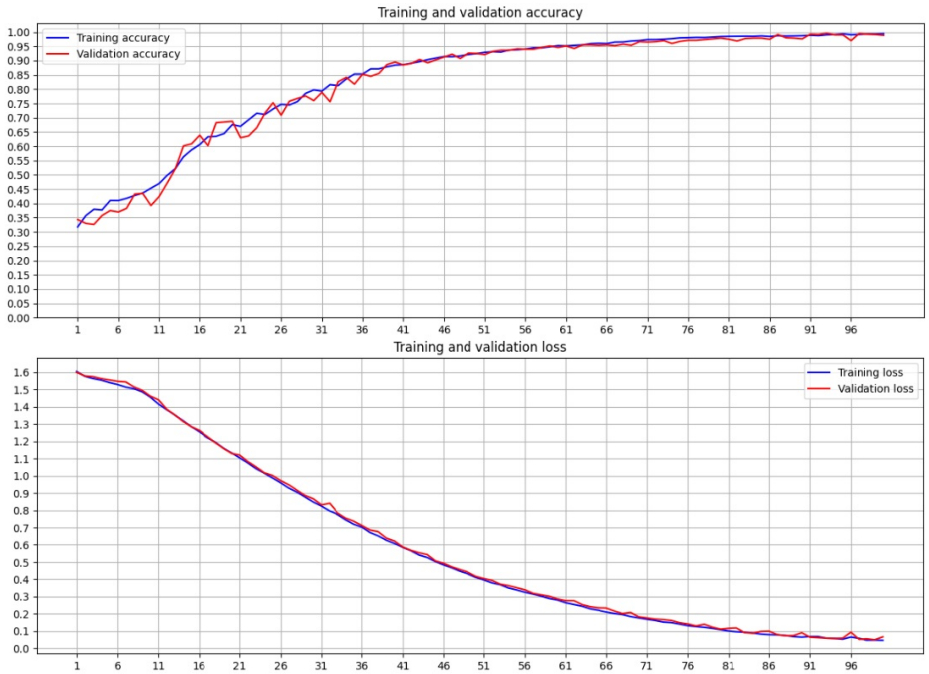

Figure 7: The training curves over the 100 epochs, on both accuracy and loss

4.4 Implementation on IoT Device

The model selected to be implemented on the IoT Smart Transformer device is of course the one with neural network. It is used the library **emlearn** to convert the Python model to C code. The model is then used practically inside the business logic of the device to infer the faults type once the simulated actuator uses the sensor value produced.

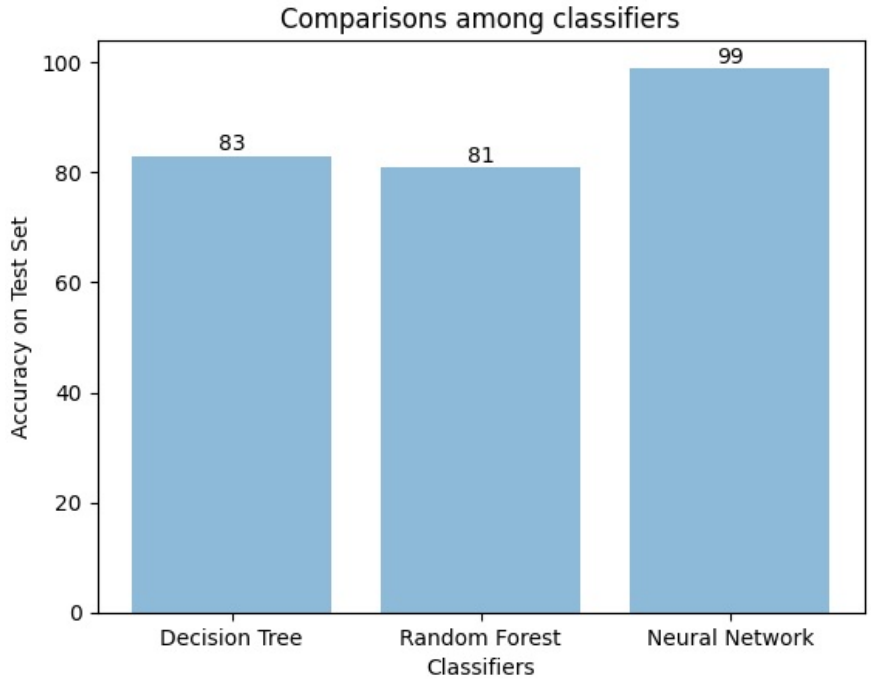

Figure 8: Comparison on accuracy among different models

Accordingly to the logic of the smart transformer device the fault types inferred by the neural network are mapped to the following led configuration:

STATUS	**FAULT TYPE**
Normal Status	0
Soft Anomaly Detected	1 and 2
Hard Anomaly Detected	3
Critical Anomaly Detected	4

Table 9: Status and Fault Type Mapping

The automatic repair procedure for the smart transformer's configuration is triggered only for fault types 1, 2, or 3. This process involves a gradual adjustment of the current values Ia, Ib, Ic within a safe range, simulating each modification with a random variation. Additionally, it ensures that the voltage values remain within predefined limits. The lack of explainability in the neural network has constrained the safe ranges, which have been derived from prior work on the decision tree model. In case of fault type 4, the operations are different. Once this fault is triggered the Smart Transformer sends a message to the Smart Power Meter device of House_1 ordering to disconnect "temporaneally" the power thought the resource /status. After, it resumes the original safety values and orders to House_1 to reactivate its power thought the /status resource. This is done avoiding to completely disconnect all the smart grid, limiting on the residential area.

5 Sensor Data Representation: SenML

In this project, an efficient and flexible data representation is employed to provide an horizontal approach to guarantee an easier future collaboration with other external applications. SenML (Sensor Measurement Lists) is a standardized data format designed for representing simple sensor measurements. It is particularly useful in the context of IoT (Internet of Things) applications, where sensors frequently communicate small amounts of data that need to be captured and transmitted efficiently. A great benefit of SenML is that it is a standardized way to represent sensor data eliminating most of the usual integration problems allowing a better interoperability. This measurements can be encoded using different encoding languages such as JSON (JavaScript Object Notation) or XML (eXtensible Markup Language).

In this project, JSON is used to encode system messages due to several key advantages.

- **Simplicity**: it has a simple syntax based on key-values pairs that results easy to be correct interpreted for both humans and machines.
- **Compactness**: it is less redundant compared to other encoding formats like XML. This reduces bandwidth usage and improves performance, particularly in systems with limited resources.
- **Flexibility**: it doesn't require a strict schema, which allows for more flexibility in handling varying or dynamic data structures respect to XML which usually used a predefined schema (XSD), adding rigidity and complexity, especially when changes are needed and in IoT context it requires lower battery usage.
- **Integration**: since JSON is a subset of JavaScript, it integrates seamlessly with web technologies and is the natural choice for front-end and back-end data interchange in web applications.

By employing a standard such as SenML using JSON encoding language, it is ensured that the sensors within the system can be integrated into a broader ecosystem. This supports the IoT paradigm, where interoperability among devices from different manufacturers and platforms is crucial for seamless operation.

5.1 SenML: General Structure

SenML organizes data into records, with each record typically containing key details about the measurements to be send. All the fields are optional unless explicitly specified in the generic fields or within the specific measurement fields.

- **Base Name** (`bn`): The common name or identifier used for the measurements in the group, often representing the sensor ID or resource name.
- **Base Time** (`bt`): The base timestamp, usually expressed in Unix epoch format, indicating when the measurements started.
- **Base Unit** (`bu`): The default unit of measurement used for all sensor readings, such as "Celsius", "meters", etc.
- **Version** (`ver`): The version number of media type format, default 1.
- **Entries** (`e`): The array of sensor measurements, where each entry consists of:
  - **Name** (`n`): The specific name or identifier of the measurement.
  - **Unit** (`u`): The name of unit for the specific measurement.
  - **Value** (`v`): The actual sensor reading or measurement value.
  - **Time** (`t`): The time offset from the base time, expressed in seconds.

In this project some conveniences are adopted.

- In order to avoid treating floating-point numbers all the floating-point values are encoded as integers by multiplying them by 100. This approach preserves a precision of two decimal places, ensuring that the sensor data remains accurate and reliable.
- The base time and time fields are not used inside the SenML message because these IoT devices lack a concept of global time, given their low-power and lossy nature. Consequently, each measurement received by the CoAP remote server is interpreted as being taken at the moment it arrives.

In order to simplify the code, enhance readability, maintainability, and minimize the likelihood of encoding errors, the **CJSON** external library has been added to the IoT Smart Device source code. The library offers a straightforward API that simplifies the process of working with JSON data in C, which is beneficial for developers who need to integrate JSON functionality into their applications without extensive overhead. This library provides a set of functions to create JSON objects and arrays programmatically, enabling the generation of JSON strings from C data types. It also includes functions for managing memory, ensuring that allocated memory for JSON objects can be properly freed to avoid memory leaks. On the other side, the remote Java application incorporates the external dependency **org.json**, which, similar to CJSON, provides a comprehensive set of APIs for handling JSON objects and arrays at a higher abstraction level.

5.2 Example of Smart Power Meter Measurement

A typical Smart Power Meter measurement includes some key fields. Here is an example:

```
{
    "bn": "urn:dev:mac:F4CE3667DB318653:/",
    "bu": "W",
    "ver": 1,
    "e": [
        {
            "n": "power",
            "v": -241024
        }
    ]
}
```

The string urn:dev:mac:F4CE3667DB318653: indicates that it uses a Uniform Resource Name (URN) format, likely based on the device's MAC address. This helps to uniquely identify the specific meter within the smart grid network. The unit of measure is "W" (watts), which is a standard unit for measuring power. The list of measurements consists in just one sample. The measurement name is "power", indicating that the entry pertains to power consumption or generation. The value associated with the measurement is -241024, that corresponds to -2410,24 Watts, and it is the amount of power measured. The negative value indicate that the power is fed back into the grid (production is greater then consumption).

5.3 Example of Smart Transformer Meter Measurement

A typical Smart Transformer Meter measurement includes several key fields, reflecting the complete status of the overall transformer. Here is an example:


```
{
    "bn": "urn:dev:mac:F4CE365387DF5630:/",
    "ver": 1,
    "e": [
        {
            "n": "state",
            "u": "type_fault",
            "v": 0
        },
        {
            "n": "current_A",
            "u": "MA",
            "v": -314
        },
        {
            "n": "current_B",
            "u": "MA",
            "v": -2662
        },
        {
            "n": "current_C",
            "u": "MA",
            "v": 2681
        },
        {
            "n": "voltage_A",
            "u": "MV",
            "v": 54
        },
        {
            "n": "voltage_B",
            "u": "MV",
            "v": -25
        },
        {
            "n": "voltage_C",
            "u": "MV",
            "v": -28
        }
    ]
}
```


The Smart Transformer device sends a set of different measurements, this is the reason why no attribute “bu” is present instead for each individual measure is present the field “u”. The first measure in the list indicates the state of the Smart Transformer, so it specifies the presence or not of faults in that specific moment. The other 6 measurements indicate respectively the current and the voltages on the three-phases (A,B,C) and their unit of measure is respectively “Mega Ampere” and “Mega Volt”. The values should be divided by 100 to obtain the effective measure value.

6 Grafana Web Application

This section presents all the dashboards created using Grafana, which are valuable for the proposed use case scenario. These charts help the user to intuitively monitor its consumptions and to take care of the smart transformer's healt by the energy provider.

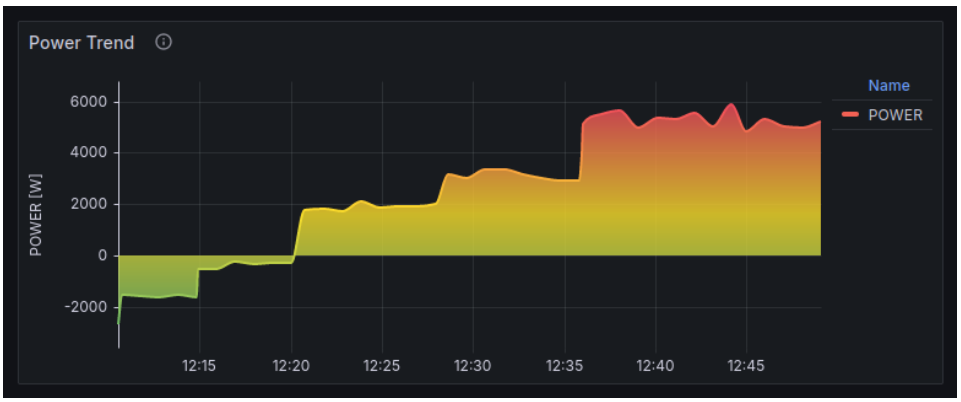

Figure 9: The figure displays the power trend for House_1. Values below 0 (shown in green) indicate energy production, while values above 0 represent energy consumption from the public electrical grid, which incurs costs.

Figure 10: The figure illustrates the trends of voltage and current over time. These values provide detailed insights, which are essential for maintaining a clear understanding of the transformer's status to effectively monitor and prevent faults.

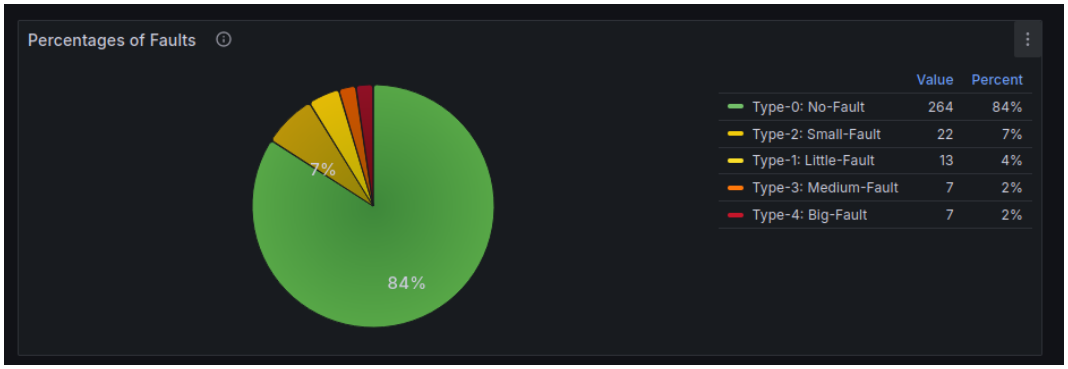

Figure 11: The figure illustrates the percentage of faults that occurred within a specific time frame. It highlights that Fault-Type 0 is prevalent, indicating that the transformer operates without errors most of the time. However, there are also small percentages of fault occurrences present.

7 Project Code Organization

The source code can be found at this link [2]
The project implementation source code is structured in 5 main folders.

1. **Project_Border_Router**: This contains the same unmodified files related to a border router implementation found in the "example" Contiki-NG directory. These files are essential for implementing a generic border router that connects to the external internet and interacts with the wireless sensor network.

2. **Project_JavaApplication**: This includes both the CoAP Server Application and the User Application, as described previously. Each is structured as a Maven project for streamlined dependency management. Both applications implement a HikariCP connection pool for efficient MySQL database access.

3. **Project_SmartPowerMeter**: This contains all the necessary files to implement the business logic of a Smart Power Meter that functions as both a sensor and an actuator, as previously described. It also includes the code for exposing its internal resources.

4. **Project_SmartTransformer**: This contains all the required files to implement the business logic of a Smart Transformer that acts as both a sensor and an actuator, as outlined earlier. It also includes the code to expose its internal resources, and a file named `smart_transformer_fault_detection.h`, which is the C representation of a trained neural network, generated using the emlearn library.

5. **Project_Utilities**: This contains shared utility files used across different parts of the code. Specifically:

   (a) **CJson Library**: used for parsing and converting strings in C.

   (b) **Global_Constants**: a collection of common constants used by both IoT devices.

   (c) **Math_Utilities**: functions for simulating the generation of sensor values within specific ranges.

   (d) **Printing_Floats**: functions that allow printing floating-point sensor measurements with a custom layout.

   (e) **SenML-JSON**: provides the general measurement representation in accordance with the SenML standard.

   (f) **Smart_Power_Meter_Utilities**: functions for initializing and managing Smart Power Meter measurement values, as well as handling specific electrical calculations and formulas.

   (g) **Smart_Transformer_Utilities**: contains logic for autonomously detecting and resolving transformer faults, as well as procedures for deactivating remote Smart Power Meters.

7.1 Project configuration

In this final section, a brief guide is provided to explain how to configure the entire setup, either for implementation on real NRF52840 Dongle devices or for simulation using the Cooja Simulator.

1. Download the Contiki-NG repository from https://github.com/contiki-ng/ and save the folder
2. Install a MySQL database on your Linux machine (e.g., using sudo apt-get install mysql-server), configure it for proper access, and run the schemaDB.sql script located in the database folder. This will create all the necessary tables to store the data.
3. Install and configure the Grafana web tool, then import the dashboard configuration file (.json) located in the Grafana folder. Follows the step provided here [3]
4. Unpack the provided Implementation folder in the /contiki-ng/example directory.
5. In a separate terminal, go to "Project_JavaApplication/JavaApplication/Server/target" and run the following command to start the CoAP server, which will receive data from sensors:

```
java -jar Server-0.0.1-SNAPSHOT.jar
```

7.1.1 Cooja Simulation

1. Navigate to the `/contiki-ng/tools/cooja` folder and run the command:

```
./gradlew run
```

   to launch the Cooja simulator.
2. Load the simulation file `Spark_Iot.csc` located in the `Cooja_Simulation` directory.
3. Move to the `Project_BorderRouter` directory and execute the following command:

```
make TARGET=cooja connect-router-cooja
```

4. Start the simulation in Cooja.
5. In a separate terminal, navigate to `Project_JavaApplication/JavaApplication/UserApplication/target` and run the following command to start the User Application, which allows managing actuators and smart grid information:

```
java -jar UserApplication-0.0.1-SNAPSHOT.jar -cooja
```

7.1.2 Real NRF52840 Dongle Devices

1. Plug the first dongle device into your computer and navigate to the `Project_BorderRouter` directory. Flash the device using the following command:

   ```
   make TARGET=nrf52840 BOARD=dongle border-router.dfu-upload
   PORT=/dev/ttyACM0
   ```

2. Run the tunslip6 connection by executing:

   ```
   make TARGET=nrf52840 BOARD=dongle PORT=/dev/ttyACM0 connect-router
   ```

3. Plug the second dongle device into your computer. Open a new terminal, navigate to the `Project_SmartPowerMeter` directory, and flash the device with the command:

   ```
   make TARGET=nrf52840 BOARD=dongle SmartPowerMeter.dfu-upload
   PORT=/dev/ttyACM1
   ```

4. Log in and view debugging messages using the command:

   ```
   make login TARGET=nrf52840 BOARD=dongle PORT=/dev/ttyACM1
   ```

5. Plug the third dongle device into your computer. Open a new terminal, navigate to the `Project_SmartTransformer` directory, and flash it using the following command:

   ```
   make TARGET=nrf52840 BOARD=dongle SmartTransformer.dfu-upload
   PORT=/dev/ttyACM2
   ```

6. Log in and view debugging messages using the command:

   ```
   make login TARGET=nrf52840 BOARD=dongle PORT=/dev/ttyACM2
   ```

7. In a separate terminal, navigate to `Project_JavaApplication/JavaApplication/UserApplication/target` and run the following command to start the User Application, which allows managing actuators and smart grid information:

   ```
   java -jar UserApplication-0.0.1-SNAPSHOT.jar
   ```

8 References

1. Dataset reference:
   https://www.kaggle.com/code/pythonafroz/transformer-fault-prediction-with-99-auc/input?select=classData.csv

2. GitHub reference:
   https://github.com/AlterVigna/Spark_Grid_IoT_Project

3. Grafana reference:
   https://grafana.com/docs/grafana/latest/installation/debian/