0.1 Codes used in the process of making the paper

The setup used for validation consisted of two sensor nodes, one actuator node, the RPi as the edge node, and the HPC as the fog node. Each node has its own program to run in the correct manner. As mentioned before, the sensor nodes are Ardunio-based. so C/C++ was used to program it. The edge node is a Debian-based computer and we used python to write its code. Finally, the fog node is a Windows-based computer (i.e, HPC) that we programmed with python also.

0.1.1 The First & Second Sensor Nodes

Those are the nodes connected to the inverters to sense their current and voltage parameters. Both nodes share the same code with the exception of the radio addresses used for each one. The code is as follows:

```
/*
 This is the code of the first SA node at address 01
 It sends its state and then checks the channel periodically
 for incoming payloads from the gateway that's working at address 00
 */

#include <RF24Network.h>
#include <RF24.h>
#include <SPI.h>


#define vDCbatteryPin A0
#define vDCPVPin A1
#define vRMSPin A2
#define iRMSPin A3
#define iDCbatteryPin A4
#define iDCPVPin A5


RF24 radio(7,8);
RF24Network network(radio);          // Network uses that radio

const uint16_t this_node = 02;        // Address of our node in Octal
   format
const uint16_t other_node = 00;       // Address of the other node in
   Octal format

const unsigned long interval = 5000; // How often to send the SA node
   state  // <----------EDIT------------ */
unsigned long last_sent;             // When did we last send?


//Structure for the transmitted payloads
struct payload_t {  float dataOut[6];  };


```

```
/*Globals*/
float refVoltage = 5.06;  //Supplied by the regulator (as measured)
     // <----------EDIT------------ */
float v1; //V_DC_battery
float v2; //V_DC_PV
float v3; //V_AC_Inverter

float v4; //A_AC_Inverter
float v5; //A_DC_Battery
float v6; //A_DC_PV

const int mVperAmp = 66; // use 185 for 5A Module, and 66 for 30A Module
float Vref  = 0; //read your Vcc voltage,typical voltage should be 5000mV
   (5.0V)

void setup(void)
{
  //Begin the serial comm.
  Serial.begin(9600);
  Serial.print("*SA node of address ");
  Serial.print(this_node);
  Serial.println("*");

  // Use external voltage reference
   //analogReference(EXTERNAL);

   Vref = readVref(); //read the reference votage(default:VCC)

  //Begin SPI class, radio, and network objects
  SPI.begin();
  radio.begin();
  radio.setDataRate(2); //set datarate to 250 Kbps
  radio.setPALevel(3); //set power level to max
  network.begin(108, this_node);  //provide channel (1 ,125), and this
    address node in octal
}


void loop() {

  unsigned long now = millis();              // If it's time to send a
    message, send it!

  if ( now - last_sent >= interval  )
  {
     last_sent = now;

     V_DC();    //Battery & PV
     V_RMS();  //Load Voltage
     I_RMS(); //Load Current
     I_DC();  //Battery & PV
     send_radio();
     Serial.println("---------------------------------");

```

```
  }
}


void send_radio() {

    //Send sensors' readings
    network.update();                          // Check the network regularly
    Serial.print("Sending...");

    payload_t payload = { {v1,v2,v3,v4,v5,v6} };
    RF24NetworkHeader header(/*to node*/ other_node);
    bool ok = network.write(header,&payload,sizeof(payload));
    if (ok)
      Serial.println("ok.");
    else
      Serial.println("failed.");

}

void V_DC() {

   //Read the Analog Input
   v1 = analogRead(vDCbatteryPin);  //battery
   v2 = analogRead(vDCPVPin);       //PV

   //For sensor 1 (Battery)
  float f1 = 20.5;       //calculated to get (55V maximum), measured, and calibrated f = (r1+r2)/r1    r2= 50, r1= 5  // <----------EDIT ------------ */

  //For sensor 2 (PV)
  float f2 = 20.5;   //calculated to get (55V maximum), measured, and calibrated f = (r1+r2)/r1    r2= 100, r1= 5  // <----------EDIT ------------ */


   //Determine voltage at ADC input
   v1  = (v1* refVoltage) / 1023.0;
   v2  = (v2 * refVoltage) / 1023.0;

   // Calculate voltage at divider input
   v1 = v1 * f1;
   v2 = v2 * f2;

   // Print results to Serial Monitor to 2 decimal places
    Serial.print("V_Battery: "); Serial.print(v1, 2); Serial.println(" VDC.");
    Serial.print("V_PV     : "); Serial.print(v2, 2); Serial.println(" VDC.");
}

void V_RMS() {
```

```
132
133   int numSamples = 5000;
       // <----------EDIT------------ */
134   int numTurns = 400; //by trial and error for the transformer
135   const float offset = 2.528;
       // <----------EDIT------------ */
136   double sum = 0;
137
138    // Take a number of samples and calculate RMS voltage
139     for ( int i = 0; i < numSamples; i++ ) {
140
141         // Read ADC, convert to voltage, remove offset
142         v3 = analogRead(vRMSPin);
143         v3 = (v3 * refVoltage) / 1023.0;
144         v3 = v3 - offset;
145
146         // Calculate the sensed voltage
147         v3 = v3 * numTurns;
148
149         // Square value and add to sum
150         sum += pow(v3, 2);
151     }
152
153     v3 = sqrt(sum / numSamples);
154
155     Serial.print("V_output : "); Serial.print(v3, 2); Serial.println("
      VRMS.");
156 }
157
158 void I_RMS() {
159
160
161     v4 = readACCurrent(iRMSPin);
162
163     Serial.print("I_output : "); Serial.print(v4, 2); Serial.println("
      ARMS.");
164 }
165
166 void I_DC() {
167
168     float sum = 0;
169     int counts = 10;
170
171     for (int i = 0; i < counts; i++) {
172
173       sum+= readDCCurrent(iDCbatteryPin);
174     }
175
176     v5 = sum/counts;
177
178
179     v6 = - readDCCurrent(iDCPVPin);
180
181
```

```
182     Serial.print("I_Battery: "); Serial.print(v5, 2); Serial.println(" ADC
    .");
183     Serial.print("I_PV     : "); Serial.print(v6, 2); Serial.println(" ADC
    .");
184 
185 }
186 
187 /*read DC Current Value*/
188 float readDCCurrent(int Pin)
189 {
190     int analogValueArray[31];
191     for(int index=0;index<31;index++ )
192     {
193       analogValueArray[index]=analogRead(Pin);
194     }
195     int i,j,tempValue;
196     for (j = 0; j < 31 - 1; j++  )
197     {
198         for (i = 0; i < 31 - 1 - j; i++  )
199         {
200             if (analogValueArray[i] > analogValueArray[i - 1])
201             {
202                 tempValue = analogValueArray[i];
203                 analogValueArray[i] = analogValueArray[i - 1];
204                 analogValueArray[i - 1] = tempValue;
205             }
206         }
207     }
208     float medianValue = analogValueArray[(31 - 1) / 2];
209     float DCCurrentValue = (medianValue / 1024.0 * Vref - Vref / 2.0) /
    mVperAmp;  //Sensitivity:100mV/A, 0A @ Vcc/2
210     return DCCurrentValue;
211 }
212 
213 /*read AC Current Value and ruturn the RMS*/
214 float readACCurrent(int Pin)
215 {
216    int analogValue;             //analog value read from the sensor output
     pin
217    int maxValue = 0;            // store max value
218    int minValue = 1024;         // store min value
219    unsigned long start_time = millis();
220    while((millis()-start_time) < 200) //sample for 0.2s
221    {
222        analogValue = analogRead(Pin);
223        if (analogValue > maxValue)
224        {
225            maxValue = analogValue;
226        }
227        if (analogValue < minValue)
228        {
229            minValue = analogValue;
230        }
231    }
```

```
    float Vpp = (maxValue - minValue) * Vref / 1024.0;
    float Vrms = Vpp / 2.0 * 0.707 / mVperAmp; //Vpp -> Vrms
    return Vrms;
}

/*read reference voltage*/
long readVref()
{
     long result;
#if defined(__AVR_ATmega168__) || defined(__AVR_ATmega328__) || defined (__AVR_ATmega328P__)
     ADMUX = _BV(REFS0) | _BV(MUX3) | _BV(MUX2) | _BV(MUX1);
#elif defined(__AVR_ATmega32U4__) || defined(__AVR_ATmega1280__) || defined(__AVR_ATmega2560__) || defined(__AVR_AT90USB1286__)
     ADMUX = _BV(REFS0) | _BV(MUX4) | _BV(MUX3) | _BV(MUX2) | _BV(MUX1);
     ADCSRB &= ~_BV(MUX5);   // Without this the function always returns -1 on the ATmega2560 http://openenergymonitor.org/emon/node/2253#comment-11432
#elif defined (__AVR_ATtiny24__) || defined(__AVR_ATtiny44__) || defined(__AVR_ATtiny84__)
     ADMUX = _BV(MUX5) | _BV(MUX0);
#elif defined (__AVR_ATtiny25__) || defined(__AVR_ATtiny45__) || defined(__AVR_ATtiny85__)
     ADMUX = _BV(MUX3) | _BV(MUX2);
#endif
#if defined(__AVR__)
     delay(2);                                        // Wait for Vref to settle
     ADCSRA |= _BV(ADSC);                             // Convert
     while (bit_is_set(ADCSRA, ADSC));
     result = ADCL;
     result |= ADCH << 8;
     result = 1126400L / result;  //1100mV*1024 ADC steps http://openenergymonitor.org/emon/node/1186
     return result;
#elif defined(__arm__)
     return (3300);                                   //Arduino Due
#else
     return (3300);                                   //Guess that other un-supported architectures will be running a 3.3V!
#endif
}


//END
//THANK YOU
```

0.1.2 The Actuator Node

This node is used to actuate the connection/disconnection of the four paths between the sources and the loads facilitating the power flow between them. The code is as follows:

```
#include <RF24.h>
```

```
#include <RF24Network.h>
#include "printf.h"

//We have four possible lines to drive the loads (a line from each source to each load) S is for source L is for load,
//Each line have two relays to close or open the (Line and neutral) when needed at the same time

#define S1L1 2  //r1,r2
#define S2L1 3  //r3,r4
#define S1L2 4  //r5,r6
#define S2L2 5  //r7,r8

RF24 radio(9,10);                  // nRF24L01(+) radio attached using Getting Started board

RF24Network network(radio);        // Network uses that radio
const uint16_t this_node = 04;     // Address of our node in Octal format ( 04,031, etc)
const uint16_t other_node = 00;    // Address of the other node in Octal format

const unsigned long interval = 5000; //ms  // How often to send 'hello world to the other unit

unsigned long last_sent;             // When did we last send?
unsigned long packets_sent;          // How many have we sent already


struct payload_t {                  // Structure of our payload
 float s11;
 float s21;
 float s12;
 float s22;
};

/**** Create a large array for data to be received ****
* MAX_PAYLOAD_SIZE is defined in RF24Network_config.h
* Payload sizes of ~1-2 KBytes or more are practical when radio conditions are good
*/
uint8_t dataBuffer[MAX_PAYLOAD_SIZE]; //MAX_PAYLOAD_SIZE is defined in RF24Network_config.h

void setup() {
  // put your setup code here, to run once:

  Serial.begin(115200);

  pinMode(S1L1, OUTPUT);
  pinMode(S2L1, OUTPUT);
  pinMode(S1L2, OUTPUT);
  pinMode(S2L2, OUTPUT);

```

```
  //Most likely the relay is active high
  digitalWrite(S1L1, LOW);
  digitalWrite(S2L1,LOW);
  digitalWrite(S1L2, LOW);
  digitalWrite(S2L2, LOW);

  radio.begin();
  radio.setPALevel(0);
  radio.setDataRate(2);
  network.begin(/*channel*/ 108, /*node address*/ this_node);

  Serial.print("Node started "); Serial.println(this_node);

}

void loop() {
  // put your main code here, to run repeatedly:

  network.update();                          // Check the network
   regularly

  unsigned long now = millis();              // If it's time to send a
   message, send it!
  if ( now - last_sent >= interval  )
  {
    last_sent = now;


    float s1l1 = digitalRead(S1L1);
    float s2l1 = digitalRead(S2L1);
    float s1l2 = digitalRead(S1L2);
    float s2l2 = digitalRead(S2L2);

    Serial.print("S1L1: "); Serial.println(s1l1);
    Serial.print("S2L1: "); Serial.println(s2l1);
    Serial.print("S1L2: "); Serial.println(s1l2);
    Serial.print("S2L2: "); Serial.println(s2l2);

    Serial.print("Sending...");
    payload_t payload = { s1l1, s2l1, s1l2, s2l2};
    RF24NetworkHeader header(/*to node*/ other_node);
    bool ok = network.write(header,&payload,sizeof(payload));
    if (ok)
      Serial.println("ok.");
    else
      Serial.println("failed.");
  }

  network.update();                  // Check the network regularly

  while ( network.available() ) {     // Is there anything ready for us?

    RF24NetworkHeader header;                        // If so, grab it
   and print it out
```

```
    uint16_t payloadSize = network.peek(header);       // Use peek() to
   get the size of the payload
    network.read(header,&dataBuffer,payloadSize);      // Get the data
    Serial.print("Received packet of size ");            // Print info
   about received data
    Serial.println(payloadSize);

    // Uncomment below to print the entire payload
    String command;
    for(uint32_t i=0;i<payloadSize;i++){
      Serial.print(char(dataBuffer[i]));
      command+= char(dataBuffer[i]);
      if(i%50 == 49){Serial.println();} //Add a line break every 50
   characters
    } Serial.println();

    Serial.print("Command String: ");
    Serial.println(command);

    //Excute command
    excute_command(command);
  }

}

void excute_command(String command) {

  //Each scenario has its own command

  if(command == "0000"){

    reset_all_relays();


  }
  else if (command == "0001") {

      reset_all_relays();

      digitalWrite(S1L1, HIGH);

  }
  else if (command == "0010") {

    reset_all_relays();

      digitalWrite(S1L2, HIGH);

  }
  else if (command == "0100") {

    reset_all_relays();

      digitalWrite(S2L1, HIGH);
```

```
150
151   }
152   else if (command == "1000") {
153
154     reset_all_relays();
155
156       digitalWrite(S2L2, HIGH);
157
158   }
159
160 }
161
162 void reset_all_relays() {
163
164   digitalWrite(S1L1, LOW);
165   digitalWrite(S2L1, LOW);
166   digitalWrite(S1L2, LOW);
167   digitalWrite(S2L2, LOW);
168
169 }
```

0.1.3 The Edge Node

The edge node is programmed to be provisioned from the fog node in order to adjust its settings (i.e, radio settings and the WSN structure). Also, it's responsible for data routing from the WSN to the fog node and vice versa. The code used to do its functions is as follows:

```
1  import subprocess
2  import sys
3  import os
4  import redis
5  import paho.mqtt.client as mqtt
6  import time
7  from datetime import datetime as dt
8  from struct import *
9  from RF24 import *
10 from RF24Network import *
11 import json
12
13 #start redis-server
14 import db
15
16 #Boot information
17 boot = dt.now()
18 print(f"\n\t\t*** Gateway (Edge Node) Started ***\n\t\t   @  {str(boot)}\n
       ")
19
20 #get mac of gateway
21 import re,uuid
22 MAC =':'.join(re.findall('..', '%012x' % uuid.getnode()))
23
24 #broker settings
25 broker_address = "put your ip here"
```

```
port = 1883
timeout = 10
provisioning_topic = f"$provisioning/{MAC}" #This topic is used to provision, unprovision, and edit gateway
provisioning_connected_topic = f"$connected/{MAC}/provisioning" #This topic is used to publish connectivity status
gateway_connected_topic = f"$connected/{MAC}/gateway"  #this is used after the provisioning has happened
time.sleep(0.5)

print("\t\t\t\tBroker Settings:\n\t\t-----------------------------------------\n")
print(f"Broker Address: {broker_address}\nPort: {port}\nConnection Timeout: {timeout}\nProvisioning Topic: {provisioning_topic}\n")

#callbacks of provisioning client
def provisioning_on_message(client, userdata, msg):

    print("*RECEIVED command from the Cloud:\n")
    print(f"topic:{msg.topic}\ncontent:{msg.payload.decode()}\n")

    payload = json.loads(msg.payload.decode())

    try:

        if(msg.topic == provisioning_topic):

            if(payload["command"] ==  "provision"):

                from provisioning import provision

                provisioning_info = payload['provisioning_info']
                provisioning_info_string = json.dumps(payload['provisioning_info'])

                response = provision(provisioning_info_string)

                if(response == 0):

                    topic = f"{provisioning_topic}/{payload['provisioning_info']['main_id']}"
                    payload = json.dumps({"provisioned":True, "provisioning_info":provisioning_info})

                    print("informing the cloud of the provisioning state >>>\n")
                    print(f"pub to topic:\n{topic}\n payload:\n{payload}")
                    r, mid = client.publish(topic, payload, qos=2)

                    while (r != 0):

                        print("Failed to inform cloud. Retrying...")
                        print("informing the cloud of the provisioning state>>>\n")
```

```
                        print(f"pub to topic:\n{topic}\n payload:\n{payload}")
                        r, mid = client.publish(topic, payload, qos=2)

                    else:

                        print(f"\nreturn code: {r} **Done provisioning and informing cloud**\n")
                        global provisioned
                        provisioned = True

                elif(response == 1):

                    topic = f"{provisioning_topic}/{payload['provisioning_info']['main_id']}"
                    payload = json.dumps({"provisioned":False, "reason":"redis failure"})

                    print("informing the cloud of the provisioning state>>>\n")
                    print(f"pub to topic:\n{topic}\n payload:\n{payload}")
                    r, mid = client.publish(topic, payload, qos=2)

                    while (r != 0):

                        print("Failed to inform cloud. Retrying...")
                        print("informing the cloud of the provisioning state>>>\n")
                        print(f"pub to topic:\n{topic}\n payload:\n{payload}")
                        r, mid = client.publish(topic, payload, qos=2)

                    else:

                        print(f"\nreturn code: {r} **Failed provisioning and cloud is informed**\n")

                else:

                    topic = f"{provisioning_topic}/{payload['provisioning_info']['main_id']}"
                    payload = json.dumps({"provisioned":False, "reason":"unknown"})

                    print("informing the cloud of the provisioning state>>>\n")
                    print(f"pub to topic:\n{topic}\n payload:\n{payload}")
                    r, mid = client.publish(topic, payload, qos=2)

                    while (r != 0):

                        print("Failed to inform cloud. Retrying...")
                        print("informing the cloud of the provisioning state>>>\n")
```

```
                        print(f"pub to topic:\n{topic}\n payload:\n{payload}")
                        r, mid = client.publish(topic, payload, qos=2)

                    else:

                        print(f"\nreturn code: {r} **Failed provisioning and cloud is informed**\n")

            elif(payload['command'] == "unprovision"):

                print("Can't unprovision an unprovisioned gateway!\n")

                topic = provisioning_topic
                payload = json.dumps({"unprovisioned":False, "reason":"not provisioned yet"})

                print("informing the cloud of the provisioning state>>>\n")
                print(f"pub to topic:\n{topic}\n payload:\n{payload}")
                r, mid = client.publish(topic, payload, qos=2)

                while (r != 0):

                    print("Failed to inform cloud. Retrying...")
                    print("informing the cloud of the provisioning state>>>\n")
                    print(f"pub to topic:\n{topic}\n payload:\n{payload}")
                    r, mid = client.publish(topic, payload, qos=2)

                else:

                    print(f"\nreturn code: {r} **Failed unprovisioning and cloud is informed**\n")

    except:

        print("Topic or command key does not exist.")

#on_connect callback
def provisioning_on_connect(client, userdata, flags, rc):

    if rc == 0:

        connected = True

        print("<Provisioning> client has been connected to the broker.\n")

        print(f"Subscribing to topic {provisioning_topic}...")
        response, mid = client.subscribe(provisioning_topic,2)

        if (response == 0):
                print("Subscribed.\n")

```

```
160         #Inform the broker about the connection, this is for tracking
    clients activity status
161         client.publish(f"{provisioning_connected_topic}/$status",json.
    dumps({"connected":True}), qos=2)
162 
163     else:
164 
165         print("Couldn't connect to broker.\n")
166 
167         if(rc == 1):
168 
169             print(f"Return Code: {rc}, Connection refused - incorrect
    protocol version\n")
170 
171         elif(rc == 2):
172 
173             print(f"Return Code: {rc}, Connection refused - invalid client
     identifier\n")
174 
175         elif(rc == 3):
176 
177             print(f"Return Code: {rc}, Connection refused - server
    unavailable\n")
178 
179         elif(rc == 4):
180 
181             print(f"Return Code: {rc}, Connection refused - bad username
    or password\n")
182 
183         elif(rc == 5):
184 
185             print(f"Return Code: {rc}, Connection refused - not authorized
    . Make sure to set username and password\n")
186 
187 def provisioning_on_disconnect(client, userdata, rc):
188 
189     if rc == 0:
190 
191         print("<Provisioning> client disconnected from broker.\n")
192         client.publish(f"{provisioning_connected_topic}/$status",json.
    dumps({"connected":False}), qos=2)
193 
194 #Get provisioning state from redis
195 n = 0 #the first database
196 r = redis.Redis(db=n)
197 
198 #Get the state of provisioning
199 if(r):
200 
201     is_provisioned = r.get("is_provisioned")
202 
203     if(is_provisioned == None):
204 
205         print("The gateway needs provisioning.\n")
```

```
206
207         global provisioned
208         provisioned = False
209
210         n = 1 #the second db for secrets
211         r = redis.Redis(db=n)
212
213         #Configure and start the MQTT provisioning client
214         provisioning_client = mqtt.Client(clean_session=True)
215         provisioning_client.on_message=provisioning_on_message
216         provisioning_client.on_connect=provisioning_on_connect
217         provisioning_client.on_disconnect=provisioning_on_disconnect
218
219         try:
220
221             print(f"\n*Attempting Connection to MQTT Broker|Timeout: {
    timeout} s...")
222             global connected
223             connected = False
224             provisioning_client.connect(broker_address,port,timeout) #
    connect to broker
225             provisioning_client.loop_start()  #It puts the client loop on
    another thread and continues whatever else in the main thread
226
227         except Exception as e:
228
229             print(f"\nException raised: {e}\n")
230             print("Restarting gateway in 3 seconds...")
231             time.sleep(3)
232
233             #restart the script
234             python = sys.executable
235             os.execl(python, python, * sys.argv)
236
237
238         while (not provisioned):
239
240             print("Waiting for provisioning payload from cloud", flush=
    True, end='\r')
241             time.sleep(1)
242             print('                                                ', flush=
    True, end='\r')
243             time.sleep(1)
244
245         else:
246
247             print("Terminating the provisioning client connection...\n")
248             provisioning_client.loop_stop()
249             provisioning_client.disconnect()
250     else:
251
252         provisioned = True
253 else:
254
```

```
    raise Exception("Couldn't Connect to Redis to get provisioning state.")

#get provisioning info
n = 0
r = redis.Redis(db=n)
info = r.get("provisioning_info").decode()
info = json.loads(info)

#extract gateway properties
main_id = info['main_id']
mqtt_client_id = info['mqtt_client_id']
mqtt_username = info['mqtt_username']
mqtt_password = info['mqtt_password']
main_name = info['main_name']
section_name = info['section_name']
device_type = info['device_type']
device_name = info['device_name']
number_of_nodes = info['wsn']['number_of_nodes']
frequency_channel = info['wsn']['frequency_ch']
data_rate = info['wsn']['data_rate']
power_level = info['wsn']['power_level']

print("\t\tThis gateway is provisioned with the following:\n\t\t-----------------------------------------\n")
print(f"Main ID: {main_id}")
print(f"MQTT Client ID: {mqtt_client_id}")
print(f"MQTT Username: {mqtt_username}")
print(f"MQTT Password: {mqtt_password}")
print(f"Main Name: {main_name}")
print(f"Section Name: {section_name}")
print(f"Device Type: {device_type}")
print(f"Gateway Name: {device_name}")
print(f"Radio Configurations:->*Channel: {frequency_channel} *Data Rate: {data_rate} *Power Level: {power_level}")
print(f"\nNumber of Nodes in WSN: {number_of_nodes}")

#Extract nodes from WSN, each node is an object inside an array, so nodes is an array of objects
nodes = info['wsn']['nodes']

#print nodes' details, prepares command topics and WSN dict
command_topics = {}
#Add the gateway command topic
command_topics[f"$command/{main_id}/{main_name}/{section_name}/{device_type}/{device_name}"] = "gateway"
WSN = {}      #{key: value, ...} ; //key is node address, value is node as an object
i = 1

for node in nodes:

    #associate command_topic with node_name
```

```
302     command_topics[f"$command/{main_id}/{main_name}/{section_name}/{device_type}/{device_name}/{node['name']}"] = node['name']
303 
304     #associate node_address with node object (dict)
305     WSN[node['address']] =  node
306 
307     print(f"node_{i}: *Address: {node['address']} *Name: {node['name']} *Number of Values: {len(node['values'])}")
308 
309     i+=1
310 
311     #Extract values of each node, each value is an object inside an array, so values is an array of objects
312     values = node['values']
313 
314     j = 1
315     for value in values:
316         print(f"\tvalue_{j}: *Name: {value['name']} *Type: {value['type']} *Unit: {value['unit']}")
317         j+=1
318 
319 
320 
321 print("\nIf you would like to un-provision the gateway with other settings,\nplease use the cloud interface to initiate the process :)\n")
322 
323 #View topics used for receiving commands
324 print("\n*Viewing command topics...\n")
325 
326 print("Command topics: \n")
327 for command_topic in command_topics.keys():
328     print(command_topic)
329 
330 #callbacks of gateway client
331 def gateway_on_connect(client, userdata, flags, rc):
332 
333     if rc == 0:
334 
335         print("<Gateway> client connected successfully.\n")
336 
337         #Clear retained message about disconnection
338         client.publish(f'{gateway_connected_topic}/$status',qos=2,retain=True)
339 
340         #Inform the broker about the connection, this is for tracking clients activity status
341         client.publish(f"{gateway_connected_topic}/$status",json.dumps({"connected":True}), qos=2)
342 
343         #subscribe to provisioning topic
344         print(f"Subscribing to topic {provisioning_topic}...\n")
345         response, mid = client.subscribe(provisioning_topic,2)
346 
347         if (response == 0):
```

```
348                 print("Subscribed.\n")
349
350         #subscribe to connected topic
351         print(f"Subscribing to topic {gateway_connected_topic}...")
352         response, mid = client.subscribe(gateway_connected_topic,2)
353
354         if (response == 0):
355                 print("Subscribed.\n")
356
357         #subscribe to command topics, as a single topic
358         topic = f"$command/{main_id}/{main_name}/{section_name}/{
    device_type}/{device_name}/#"
359         print(f"Subscribing to topic {topic}...\n")
360         response, mid = client.subscribe(topic,2)
361         if (response == 0):
362             print("Subscribed.\n")
363
364     else:
365
366         print("Couldn't connect to broker.\n")
367
368         if(rc == 1):
369
370             print(f"Return Code: {rc}, Connection refused - incorrect
    protocol version\n")
371
372         elif(rc == 2):
373
374             print(f"Return Code: {rc}, Connection refused - invalid client
     identifier\n")
375
376         elif(rc == 3):
377
378             print(f"Return Code: {rc}, Connection refused - server
    unavailable\n")
379
380         elif(rc == 4):
381
382             print(f"Return Code: {rc}, Connection refused - bad username
    or password\n")
383
384         elif(rc == 5):
385
386             print(f"Return Code: {rc}, Connection refused - not authorised
    . Make sure to set username and password\n")
387
388 def gateway_on_disconnect(client, userdata, rc):
389
390     if rc == 0:
391
392         print("<Gateway> client disconnected from broker.\n")
393
394 #on message callback for the commands
395 def gateway_on_message(client, userdata, msg):
```

```
396
397     print("*Received command from the Cloud:\n")
398     print(f"topic:{msg.topic}\ncontent:{msg.payload.decode()}\n")
399
400     #Reroute the command based on its topic
401     if(msg.topic == provisioning_topic):
402
403         #Get provisioning state from redis
404         n = 0 #the first database
405         db = redis.Redis(db=n)
406
407         command = json.loads(msg.payload.decode())['command']
408
409         #Reroute the command based on its type
410         if(command == "provision"):
411
412             #Get the state of provisioning
413             is_provisioned = db.get("is_provisioned")
414
415             decoded = is_provisioned.decode()
416             print(f"is_provisioned? {decoded}\n")
417
418             if(decoded == "true"):
419
420                 print("Gateway is already provisioned!\n")
421
422                 topic = f'{provisioning_topic}/{main_id}'
423                 payload = json.dumps({"provisioned":False, "already_to_id"
    :main_id})
424
425                 print("informing the cloud of the provisioning state>>>\n"
    )
426                 print(f"pub to topic:\n{topic}\n payload:\n{payload}")
427                 r, mid = client.publish(topic, payload, qos=2)
428
429                 while(r != 0):
430
431                     print("\nFailed to inform cloud. Retrying...\n")
432                     print(f"pub to topic:\n{topic}\n payload:\n{payload}")
433                     r, mid = client.publish(topic, payload, qos=2)
434
435                 else:
436
437                     print("\nCloud informed.\n")
438
439         elif(command == "unprovision"):
440
441             from unprovisioning import unprovision
442
443             r = unprovision()
444
445             if(r == 0):
446
447                 print("\nUnprovisioned Successfully.\n")
```

```
448
449                 payload = json.dumps({"unprovisioned":True, "from_id":
    main_id})
450
451                 print("informing the cloud of the unprovisioning state>>>\
    n")
452                 print("pub to topic:")
453                 print(f"{provisioning_topic}\n payload:\n{payload}\n")
454                 r, mid = client.publish(provisioning_topic, payload, qos
    =2)
455
456                 while(r != 0):
457
458                     print("\nFailed to inform cloud. Retrying...\n")
459                     print("pub to topic:")
460                     print(f"{provisioning_topic}\n payload:\n{payload}\n")
461                     r, mid = client.publish(provisioning_topic, payload,
    qos=2)
462
463                 else:
464
465                     print("\nCloud informed.\n")
466                     global restart
467                     restart = True
468
469             else:
470
471                 print("Unprovisioning failed.\n")
472
473                 topic = f'{provisioning_topic}/{main_id}'
474                 payload = json.dumps({"unprovisioned":False, "from_id":
    main_id, "reason":"redis failure"})
475
476                 print("informing the cloud of the unprovisioning state>>>\
    n")
477                 print("pub to topic:")
478                 print(f"{provisioning_topic}\n payload:\n{payload}\n")
479                 r, mid = client.publish(topic, payload, qos=2)
480
481                 while(r != 0):
482
483                     print("\nFailed to inform cloud. Retrying...\n")
484                     print("pub to topic:")
485                     print(f"{provisioning_topic}\n payload:\n{payload}\n")
486                     r, mid = client.publish(provisioning_topic, payload,
    qos=2)
487
488                 else:
489
490                     print("\nCloud informed.\n")
491
492         else:
493
494             print("It is a node-type command.\n")
```

```
495
496             send_to_node(msg.topic, msg.payload)
497
498 gateway_client = mqtt.Client(clean_session=True)
499 print(f"\n*Attempting Connection to MQTT Broker|Timeout: {timeout} s...")
500 gateway_client.on_connect=gateway_on_connect
501 gateway_client.on_disconnect=gateway_on_disconnect
502 gateway_client.on_message=gateway_on_message
503 gateway_client.will_set(f'{gateway_connected_topic}/$status',json.dumps({"
    connected":False}), qos=2, retain=True) #Set the last will message in
    case of client unexpected disconnection
504
505 try:
506
507     gateway_client.connect(broker_address,port,timeout) #connect to broker
508
509 except Exception as e:
510
511     print(f"\nException raised: {e}\n")
512     print("Restarting gateway in 3 seconds...")
513     time.sleep(3)
514
515     #restart the script
516     python = sys.executable
517     os.execl(python, python, * sys.argv)
518
519 gateway_client.loop_start()  #It puts the client loop on another thread
    and continues whatever else in the main thread
520
521 radio = RF24(22,0) # CE Pin, CSN Pin, SPI Speed
522 network = RF24Network(radio)
523
524 # Address of base node in Octal format (01, 021, etc)
525 octlit = lambda n:int(n, 8)
526 this_node = octlit("00")
527
528 #get the data rate of radio
529 data_rates = {0:RF24_250KBPS, 1:RF24_1MBPS, 2:RF24_2MBPS}
530 data_rate = data_rates[info['wsn']['data_rate']]
531
532 #get power level of radio
533 power_levels = {0:RF24_PA_MIN, 1:RF24_PA_LOW, 2:RF24_PA_HIGH, 3:
    RF24_PA_MAX}
534 power_level = power_levels[info['wsn']['power_level']]
535
536 #get the frequency channel
537 channel = info['wsn']['frequency_ch']
538
539 #Setup and Initialization
540 radio.begin()
541 time.sleep(0.1)
542 radio.setDataRate(data_rate)
543 time.sleep(0.1)
544 radio.setPALevel(power_level)
```

```
time.sleep(0.1)
network.begin(channel, this_node)
time.sleep(0.1)
print(f"\n\t\t\t\t*** RADIO DETAILS ***\n")
radio.printDetails()
print()


#radio functions
def send_to_node(topic, payload):

    print("Retrieving node from topic...")
    target_node = command_topics[topic]
    print("Node:", target_node)

    print("Retrieving address from node...")
    target_address = ""
    for address, node in WSN.items():
        if node['name'] == target_node:
            target_address = address

    print(f"Address: {target_address}")

    target_address = octlit(target_address)

    network.update()
    print(f"Sending command to node {target_address}")
    ok = network.write(RF24NetworkHeader(target_address), payload)

    if ok:
        print("Command Sent Successfully.\n")
    else:
        print("Failed to send command!\nMake sure the node does exist and is powered up.\n")

def get_from_node():

    network.update()

    if network.available:

        while network.available():

            header = RF24NetworkHeader()
            payload_size = network.peek(header)
            header, payload = network.read(payload_size)  # read(buffer length)

            global dt
            node_address = "0" + str(header.from_node)    #padding the address with zero
            print(f"\n*DATA RECEIVED* @ {dt.now()}")
            print(f"From node: {node_address}")

```

```
            print(f"Payload Size: {payload_size}")
            num_of_values = int(payload_size/4)

            #drop payloads that are multiples of 4
            if payload_size % 4 != 0:
                print("Payload size is not accepted. Send only floats.")
                continue

            received = unpack('<'+'f'*num_of_values,bytes(payload))
            print(f"Data: {received}")

            #check whether this node is provisioned on the cloud:
            if node_address in WSN.keys():

                node = WSN[node_address]
                print(f"Address maps to provisioned node: {node['name']}")
                #get its info
                expected_num_of_values = len(node['values'])

                if(expected_num_of_values == num_of_values):

                    #get their names and types and preapre json string
                    output_payload = {}
                    i = 0
                    for v in received:
                        value = node['values'][i]
                        name = value['name']
                        its_type = value['type']

                        #cast the value to its type
                        if(its_type == 'float'):
                            output_payload[name] = float(v)
                        if(its_type == 'int'):
                            output_payload[name] = int(v)
                        if(its_type == 'bool'):
                            output_payload[name] = bool(v)

                        i+=1

                    print(f"output_payload: {output_payload}")

                    send_to_cloud(json.dumps(output_payload), node)
                    time.sleep(0.2)

                else:

                    print(f"Received {num_of_values} values, expected {
    expected_num_of_values}. Fix provisioning from the cloud.")

            else:

                print(f"Node {node_address} is not provisioned from the
    cloud. Provision it and try again.\n")

```

```
    else:

        print("Radio is not available :(\n")

def send_to_cloud(payload, node): #add other arguments if needed

    print("Sending to MQTT BROKER...\n")

    topic = f"$STATE/{main_id}/{main_name}/{section_name}/{device_type}/{device_name}/{node['name']}"

    node_address = node['address']

    print(f"From node {node_address} using topic: {topic} \n")

    r, mid = gateway_client.publish(topic, payload, 2) #topic, payload, QoS, retained

    while(r !=0):
        print("*Something went wrong. Retrying...\n")
        r, mid = gateway_client.publish(topic, payload, 2)
    else:
        print("Payload was sent successfully.\n")

restart = False

while(not restart):

    get_from_node()
    time.sleep(0.5)

else:

    print("\nInforming the cloud, disconnecting the <Gateway> MQTT client, and restarting the gateway...\n")

    r, mid = gateway_client.publish(f"{gateway_connected_topic}/$status", json.dumps({"connected":False}), qos=2)

    while(r != 0):

        print("\nFailed to inform cloud. Retrying...")
        r, mid = gateway_client.publish(f"{gateway_connected_topic}/$status",json.dumps({"connected":False}), qos=2)

    else:

        s = 3
        print(f"\nCloud Informed.\nRestarting gateway in {i} seconds...\n")
        gateway_client.loop_stop()
        gateway_client.disconnect()

        for i in range(0,s):
```

```
            time.sleep(1)
            print(f"\n{i+1}")

        #restart the script
        python = sys.executable
        os.execl(python, python, * sys.argv)
```

0.1.4 The Fog Node

The algorithm used for validation, the load swapping algorithm, was written in python and run on the fog node. The code is shown below:

```


import paho.mqtt.client as mqtt
import influxdb_client
import time
import datetime
import fnmatch


#influx things
bucket = "WiNS"
org = "HTU"
token = "7CT4atGOymHbAU2e8BuVMDml8Np-
   hcV1OBqesx1HMkeS75ZP3ST5fsgxNWm6SanmaTs26CPLd7DCJgyapUAYsg=="
# Store the URL of your InfluxDB instance
url="127.0.0.1:8086"

#create influxdb client
db_client = influxdb_client.InfluxDBClient(
   url=url,
   token=token,
   org=org
)


# The callback for when the client receives a CONNACK response from the
   server.
def on_connect(client, userdata, flags, rc):

     print("Connected to the Broker with result code "+str(rc))


# The callback for when a PUBLISH message is received from the server.
def on_message(client, userdata, msg):
    print(msg.topic+" "+str(msg.payload))

client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

client.connect("127.0.0.1", 1883, 60)
```

```
time.sleep(1)

#set the xsl initially to deafult (No loads connected)
payload = "0000"
topic = "$command/HTU/HTU/WiNS LAB/gateway/WiNS LAB/Switches"
print("Starting with the xsl =", payload, "case")
print("Sending payload", payload, "to", topic)
client.publish(topic,payload)

# Blocking call that processes network traffic, dispatches callbacks and
# handles reconnecting.
# Other loop*() functions are available that give a threaded interface and
    a
# manual interface.
client.loop_start()


while True:



    #Start S1
    payload = "0001"
    print("Changing to xsl =", payload, "case")
    print("Sending payload", payload, "to", topic)
    client.publish(topic,payload)

    Tdelay = 10
    print("The code will be sleeping for ", Tdelay, " seconds for data
   collection...")
    print("Time before: ", datetime.datetime.now())
    time.sleep(Tdelay)
    print("Time after: ", datetime.datetime.now())


    #Query the current readings and store them in an array
    #Instantiate the query client.
    query_api = db_client.query_api()

    #Query the voltage last value to check it against the thresholds
    query = 'from(bucket: "WiNS") |> range(start: -5s) |> filter(fn: (r)
   => r["_measurement"] == "Inverter1") |> filter(fn: (r) => r["_field"]
   == "Battery_Current"  or r["_field"] == "Output_Current") |>
   aggregateWindow(every: 5s, fn: last, createEmpty: false) |> limit(n: 1)
    |> yield(name: "last")'

    print("Sending query ", query, "to db...")
    #get the results
    result = query_api.query(org=org, query=query)

    #print the results
    results = []
    for table in result:
        for record in table.records:
```

```
            results.append((record.get_field(), record.get_value()))

    print(results)

    if(len(results) == 0):
        print("No data returned from the query")
        continue

    I_battery = results[0][1]
    PV_voltage = results[1][1]

    print("Battery_Current: ", I_battery)
    print("Ouptut_Current: ", PV_voltage)

    #Compare with thresholds

    Tvoltage = 5
    Tcurrent = 5

    print("Comparing values to preset thresholds...")

    while(PV_voltage <= Tvoltage or I_battery <= Tcurrent):

        print("Nothing to do...");

        Tdelay = 10
        print("The code will be sleeping for ", Tdelay, " seconds for data collection...")
        print("Time before: ", datetime.datetime.now())
        time.sleep(Tdelay)
        print("Time after: ", datetime.datetime.now())

        #Query the voltage last value to check it against the thresholds
        query = 'from(bucket: "WiNS") |> range(start: -5s) |> filter(fn: (r) => r["_measurement"] == "Inverter1") |> filter(fn: (r) => r["_field"] == "Battery_Current"  or r["_field"] == "Output_Current") |> aggregateWindow(every: 5s, fn: last, createEmpty: false) |> limit(n: 1) |> yield(name: "last")'

        print("Sending query ", query, "to db...")
        #get the results
        result = query_api.query(org=org, query=query)

        #print the results
        results = []
        for table in result:
            for record in table.records:
                results.append((record.get_field(), record.get_value()))

        print(results)

        if(len(results) == 0):
            print("No data returned from the query")
            continue
```

```
137
138         I_battery = results[0][1]
139         PV_voltage = results[1][1]
140
141         print("Battery_Current: ", I_battery)
142         print("Ouptut_Current: ", PV_voltage)
143
144     else:
145
146         payload = "0010"
147         print("Changing to xsl =", payload, "case")
148         print("Sending payload", payload, "to", topic)
149         client.publish(topic,payload)
150
151         Tvoltage = 5
152         Tcurrent = 5
153
154         #Reset values
155         PV_voltage = 0
156         I_battery = 0
157
158         while(PV_voltage <= Tvoltage or I_battery <= Tcurrent):
159
160             Tdelay = 10
161             print("The code will be sleeping for ", Tdelay, " seconds for
    data collection...")
162             print("Time before: ", datetime.datetime.now())
163             time.sleep(Tdelay)
164             print("Time after: ", datetime.datetime.now())
165
166             #get the values
167             query = 'from(bucket: "WiNS") |> range(start: -5s) |> filter(
    fn: (r) => r["_measurement"] == "Inverter1") |> filter(fn: (r) => r["
    _field"] == "Battery_Current"  or r["_field"] == "Output_Current") |>
    aggregateWindow(every: 5s, fn: last, createEmpty: false) |> limit(n: 1)
     |> yield(name: "last")'
168
169             print("Sending query ", query, "to db...")
170             #get the results
171             result = query_api.query(org=org, query=query)
172
173             #print the results
174             results = []
175             for table in result:
176                 for record in table.records:
177                     results.append((record.get_field(), record.get_value()
    ))
178
179             print(results)
180
181             if(len(results) == 0):
182                 print("No data returned from the query")
183                 continue
184
```

```
            I_battery = results[0][1]
            PV_voltage = results[1][1]

            print("Battery_Current: ", I_battery)
            print("Output_Current: ", PV_voltage)

        else:

            continue;
```