Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2014, Article ID 506203, 15 pages
http://dx.doi.org/10.1155/2014/506203

Research Article

A Middleware for the Integration of Smart Grid Elements with WSN Based Solutions

Paulo Régis C. de Araújo,[1,2] Raimir Holanda,[1] Antonio Wendell de Oliveira Rodrigues,[1] André Luiz Carneiro de Araújo,[1] José de Aguiar Moraes Filho,[1] and João Paolo C. M. Oliveira[1]

[1]*Department of Computer Science (PPGIA), University of Fortaleza, Avenue Washington Soares 1321, Fortaleza, CE, Brazil*
[2]*Department of Telematics, Federal Institute of Ceará, Avenue 13 de Maio 2081, Benfica, Fortaleza, CE, Brazil*

Correspondence should be addressed to Paulo Régis C. de Araújo; pauloregi@gmail.com

Received 4 August 2014; Accepted 4 November 2014; Published 16 December 2014

Academic Editor: Alessandro Bogliolo

Currently, electricity distributors make use of various types of equipment divided into levels of automation. This automation enables the integration of elements such as Intelligent Electronic Devices (IEDs) to the supervision of the distribution electrical system, but there is not an appropriate environment to increase the scale of these elements. In this context, the smart grid comes with specifications that allow adding new elements to the intelligence of the power grid operation. However, the cost of communication is still an impediment to the scalability of the integration of these elements into the current structure. In this paper, we propose a middleware that optimizes the communication of this integration using wireless sensor networks (WSN). The goal is to ensure a gradual integration of new elements taking advantage of the increase in the number of sensor nodes in the network due to the scalability of the system itself. The conversion solutions have been used to allow easy communication between the WSN and the smart grid system, and we also have used data aggregation and compression techniques to increase the lifetime of the wireless sensor network.

1. Introduction

As presented in Volkmann and Bretas [1], the quantity and quality of the information processed by the Substation Automation Systems (SAS) have an important feature, because it is part of a critical application, and it has greater reliability than the other industry sectors. All the devices within the substation must be protected and constantly monitored by the SAS that collects information from power system equipment and manages their actions. In addition, there is a trend toward monitoring the actions of the equipment which is installed outside the substations, more precisely at the electricity distribution or transmission lines, or even at the point of customer consumption. Thus, due to the distance from the substation, there is difficulty and a great cost in the communication between the monitoring and measuring equipment and the substation when the system is using a wired network (Ethernet). In this situation, the best option is to use wireless communication, for example, RF (radio frequency), GSM (global system for mobile communications), and GPRS (general packet radio service), among others. Due to the low cost and lower energy consumption, RF communication is more attractive, and this characteristic makes WSN (wireless sensor networks) an excellent application for monitoring actions over large areas. There is also a growing trend in the control and automation of electrical system equipment and components, toward devices which now have "intelligence" and are called IEDs (intelligent electronic devices). The interconnection, communication, and control of these devices has originated the concept of the smart grid which has been explained in [2–4].

The control center of a smart grid requires information acquisition from different systems and the control of a large number of IEDs (intelligent electronic devices) installed on the electricity distribution network. Currently, a wide variety of protocols, interfaces, and proprietary systems offered by different manufacturers make the management of this

communication and automation infrastructure the major barrier to effective deployment of smart grids. As long as there are no systems and structures that standardize the acquisition, storage, and access to information, the costs of implementation and maintenance of smart grids will be prohibitive for electricity companies. For this reason, the DNP3 (distributed network protocol) was developed by Harris Distributed Automation Products. In November 1993, the responsibility for defining further DNP3 specifications and for ownership of the DNP3 specifications was turned over to the DNP3 Users Group, a group composed of utilities and vendors who are utilizing the protocol. In the same way, the IEC (International Electrotechnical Commission), which is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees), through its technical committee 57 for standardization of the power system control and associated communications, has created the IEC 61850 protocol.

Due to the scalability of new components outside the substations, including the measurement of electricity consumption, the WSNs can be widely used. With the increasing use of the electrical system components standardization and their communications, the WSNs that will be installed on the electrical system must accept and understand the rules that are imposed by the communication protocols of the smart grid, such as the DNP3 and IEC-61850 protocols. So it becomes necessary for a translation or conversion between the type of communication used by the WSN and the type based on the smart grid (DNP3 or IEC-61850) protocols, as well as for an architecture and the management of the WSN, to meet the reliability and real-time constraints imposed by an electrical system monitoring application. The reliability concept is important because a high voltage power system environment poses serious challenges to the reliability of WSN communications in future smart grids [5]. Furthermore, electric power systems may be subject to a wide range of high power disturbances that may affect wireless communication systems that operate inside the 2.4 GHz ISM band as well as their electronic circuits [6]. In this paper, the authors present a middleware, which consists of the modules "wSn converter," "Core," and "DNP3 converter," to enable the communication and integration of a WSN with a smart grid, and also to enable the implementation of techniques for query management in the sensor network, in order to increase the responses' reliability and the WSN lifetime. Due to the application scalability, a dynamic addressing system for the network sensor nodes is also proposed.

This work is structured in four sections. Section 2 presents the materials and methods which were used for developing this work. In Section 3, the results are demonstrated and the discussion about the results is presented. We use Section 4 to describe the conclusions. Additionally, the acknowledgments and references are included.

2. Materials and Methods

Due to the evolution of new substation electrical and distribution side equipment, which already provides wireless communication, or for the use of the legacy equipment with wired communication, the WSN concept will always be present. Thus, strategies and techniques for integrating these devices through the WSN are required for smart grids. An example is for a WSN to integrate itself and communicate with an electrical system which is using the DNP3 protocol for its communication. Therefore it is necessary that the WSN understands the requests made by the SAS, usually through SCADA (supervisory control and data acquisition) [7] and also that the WSN's sink node converts the network answers into a message structure used by the smart grid protocol. In some works such as [8, 9], the sink node is called the gateway node.

This work tries to contribute to the integration of new equipment or sensors with smart grids, based on the WSNs, by executing the query manager (running on the sink node) and also by executing the middleware (running on the Concentrator). The concentrator, which consists of some C and Python modules, runs on the substation RTU (remote terminal unit). In addition, the WSN query techniques have been implemented to allow a more reliable operation and also to extend the WSN lifetime. To validate this work, we have used XBEE radios to perform the WSN communication. These radios are based on the IEEE 802.15.4 standard and implement ZigBee which is a protocol widely used in smart grids, as presented in [9]. Due to the characteristics of the ZigBee protocol such as self-forming, self-healing, and secure wireless communication protocol, some authors [2, 8–11] have used this protocol in their works. Each sensor node is based on the Arduino UNO architecture, which has the ATmega328 microcontroller. The sink node consists of the Raspberry PI architecture, due to the higher processing power and better storage and IO capability. All components of this network are integrated with the smart grid through a server (RTU (remote terminal unit)), called the concentrator, which is interconnected with the SCADA software. Below we present a description of all the material used to perform this work.

(i) The sensor node consists of the Arduino UNO Kit, based on the ATmega328 microcontroller, with 32 K of flash memory, 2 K of SRAM, and running at 16 MHz. This kit is programmed using the Arduino software which is based on the C language.

(ii) The communication between the sensor nodes and the sink node is performed by the XBEE S2 radio, with 2.4 GHz of frequency, 1.25 mW of power, 120 m of distance, ZigBee/Mesh topology, and 250 Kbps of data rate.

(iii) The sink node is represented by the Raspberry PI model B, with 512 MB of RAM, two USB ports, and a 100 Mb Ethernet port. The Raspberry PI runs the Raspbian operating system which is derived from the Debian OS. The application into the Raspberry is made using the Python language.

(iv) A server is represented by a VM of 1 GB of memory, 1 CPU, and 256 GB of storage. This server runs the

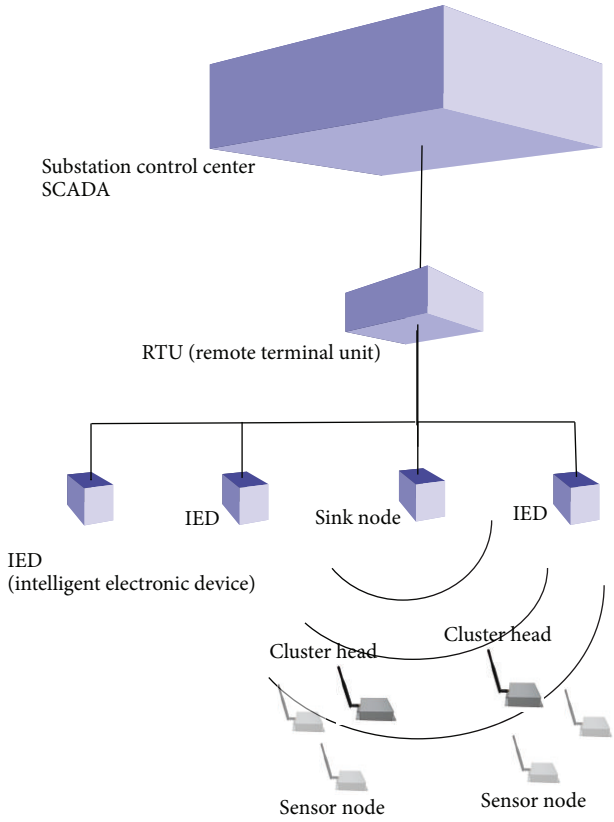

Figure 1: System topology including the WSN and the substation side environment.

Linux operating system, and the application into the server is made using C and Python languages.

To explain the functionality of the application, we will present in the next subsection the system topology as well as a detailed explanation of its operation.

2.1. Substation Side Environment—RTU and Concentrator. To explain the functionality of the application, we present in Figure 1 the system topology which consists of the substation side equipment and controls and the WSN. In the substation control center, all the substation control and supervision management is present. Part of this control and supervision is carried out by a supervisory software called SCADA (supervisory control and data acquisition) which performs reading requests of the instantaneous values of the power system analogical and digital points, such as voltage, current, circuit breaker status, and switch positions. Furthermore, the SCADA can act on the IED to control it.

The SCADA communicates with the RTU (remote terminal unit) using a standard protocol, such as the DNP3 or IEC-61850 protocols, to perform read requests on the measurements (and to operate over systems, if necessary). Based on Figure 2, communication between the SCADA and RTU happens by invoking the method "GWDNP3()" (GateWay DNP3) that belongs to the module "Core," in the concentrator.

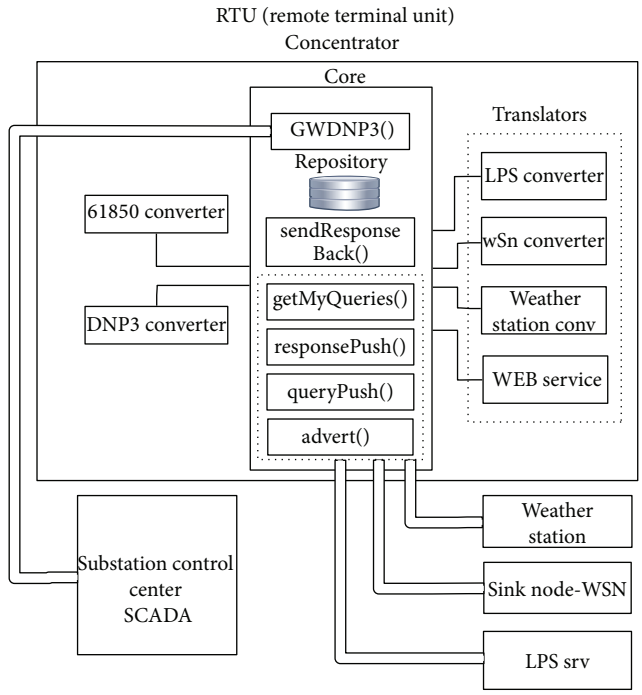

Figure 2: Block diagram of the concentrator.

When the request comes to the RTU, it selects, through the modules "DNP3 converter" and "61850 converter", which system should be used and, through the translators, translates the requests into its own protocol, allowing communication. Likewise, if there occurs a read request for a WSN node, a translator module named "wSn converter" should run in the concentrator to translate every data, allowing communication within the RTU. The concentrator is software developed by the authors which constitutes a set of protocol translating modules. There are modules developed to translate each proprietary protocol of the legacy equipment and also DNP3 and IEC-61850, allowing full communication within the RTU and to SCADA. Specifically, the middleware, which is the focus of this work, consists of the modules "wSn convertr," "Core," and "DNP3 converter," and has, as its main purpose, the communication between a WSN and the smart grid.

In Figure 2, the block diagram of the Concentrator which is responsible for allowing communication between the legacy equipment or sensors and the smart grid is presented.

The concentrator communicates internally with its modules through RPC (remote procedure call) due to the impossibility of having all modules running on the same server. As explained in [12], remote procedure call (RPC) is an interprocess communication that allows a computer program to call or execute the procedure or routine in another address space (generally on a remote machine). RPC protocol makes the remote procedure look like a local one. In our work, RPC was chosen to make a transparent data transport and enable the development of the application in its own protocol.

Due to features such as the recognition of new devices, request for protocol translations, and request for Smart Grid pattern recognition, among others, the most important module of the concentrator (and also of the middleware)

```
def advert(self,id):
  try:
    if persistencia[id]
      pass
  except:
    persistencia[id]={
      'tensaoFaseA':0,
      'tensaoFaseB':0,
      'tensaoFaseC':0,
      'correnteFaseA':0,
      'correnteFaseB':0,
      'correnteFaseC':0,
      'correnteDeFuga':0,
      'status':0
      }
  ...
```

Algorithm 1: Method advert(id).

```
def sendResponseBack(self,id):
  _c=lambda v, c: (int(((v*2.35/
  (8388608*3)*701*c/10000)*65535)/
  2000))
  _k=[
    'tensaoFaseA',
    'tensaoFaseB',
    'tensaoFaseC',
    'correnteFaseA',
    'correnteFaseB',
    'correnteFaseC',
    'correnteDeFuga'
    ]
  for i in range(len(_k)):
    self.rpcclient.srvDNP3(
    id,2,1,i,_c(self.religadores[id]
    [_k[i]],self.constantes[id][_k[i]])
    )
...
```

Algorithm 2: Method sendResponseBack(id).

```
def getMyQueries(self,id):
  try:
    _tmp=self._queries[id][0]
    ['query']
    self._queries[id][0]['status']=0
    return _tmp
  except:
    return -1
```

Algorithm 3: Method getMyQueries(id).

```
def responsePush(self, id, command):
  command = command.decode('hex')
  _tmp=self.protocol.detect(command)
  tmp.decode(command)
  ...
  try:
    self.response[id].append(command)
  except:
    self.response[id] = []
    self.response[id].append(command)
  return True
```

Algorithm 4: Method responsePush(id, command).

```
def queryPush(self, id, command):
  try:
    self._queries[id][0]['status']=1
    self._queries[id][0]['query']=
    command.encode('hex')
  except:
    self._queries[id]=[{'status':0,
    'query':''},
    {'status':0,'query':''},
    None,None,None,None,None,None,
    {'status':0,'query':''}]
    self._queries[id][0]['status']=1
    self._queries[id][0]['query']=
    command.encode('hex')
  return True
```

Algorithm 5: Method queryPush(id, command).

is the "Core." In the "Core," there are 6 common methods: advert(id); sendResponseBack(id); getMyQueries(id); responsePush(id); queryPush(id, command); GWDNP3(). These methods and the repository have been implemented to manage the requests and responses. "Core" is the only module that communicates with the other modules and devices. Below, we will present these methods and their functionality.

The first one is the announcement method which is called "advert(id)." This method is invoked to instantiate a new device when it is connected to a smart grid, and its main function is to create the basic structure to store sensor measurement values internally. See Algorithm 1.

The second one is the SCADA return method "sendResponseBack(id)" which collects the query response from the repository and regulates and normalizes it. After that, it calls the method "GWDNP3()" to send the data to SCADA, using the DNP3 protocol connected via RPC. See Algorithm 2.

The third method is "getMyQueries(id)," a method that is used by the layer directly connected to the device in order to get all requests generated by SCADA for its ID (IDentifier). Thus, the device can process the queries and answer them. See Algorithm 3.

The fourth method, called "responsePush(id,command)," is used by the layer directly connected to the device in order to receive all query responses and store them into a repository. See Algorithm 4.

The method called "queryPush(id, command)" is created to translate the packet generated by the method

```
def GwDNP3(rbt_id, action, dataType,
pointIndex, pointValue=0):
  try:
    self.pontos[id].append(
    {'action':action,'datatype':
    dataType,'pointindex':pointIndex})
  except:
    self.pontos[id] = []
    self.pontos[id].append(
    {'action':action,'datatype':
    dataType,'pointindex':pointIndex})
    if action == 0: # comando leitura
      if dataType == 2: # binary output
        self.rbt.payload(0x02)
      ...
      self.queryPush(rbt_id,
      self.rbt.encode())
    else:
      pass
  return True
```

Algorithm 6: Method GWDNP3().

"GWDNP3()" to the internal structure and then to store it in the repository.

And, finally, the method "GWDNP3()" is the gateway for SCADA commands, and its parameters for the default values of the SCADA points, in order to perform requests, are action, data type, and point index. Another parameter, called point value, has been used to act on the IED. These variables are mapped internally in SCADA, in a nonstandard way, and the operator is responsible for selecting them. When this module is asked, it checks to see if there is a return structure ready for the device. If there is not a return structure, it creates one, because all the devices must have a monitoring framework for various cases. Then, it checks what kind of "action" should be performed (if a reading or acting operation) and what kind of command has been used. Finally, an order is sent to the layer directly connected to the device, through the "queryPush()," which is ready to be sent to the device to be processed. See Algorithms 5 and 6.

The concentrator consists of other administrative functions, but the methods presented above are the most important to provide data flux between any kind of device and SCADA.

All other information is stored internally in the concentrator through global variables, and any other module can use them, allowing intercommunication without SCADA interference.

2.2. The WSN and the Sink Node Modules: The Query Processor and Data Aggregation Modules. As discussed by Karl and Willig [13], in many classes of applications, the physical environment is the focus of attention. Computation is used to exert control over physical processes, for example, when controlling chemical processes in a factory for correct temperature and pressure. At the substation scenario, the sensing of the events and variables is performed by the sensors. These sensors are represented by the IEDs and other legacy equipment. Some IEDs are connected using wires, which offers some obstacles to success: wiring is more expensive, wires constitute a maintenance problem, wires prevent entities from being mobile, and wires can prevent sensors and actuators from being close to the phenomenon that they are supposed to control. Hence, *wireless communication* between such devices is, in many application scenarios, an inevitable requirement. One of the most effective and inexpensive ways of wireless communication is the use of radio frequency (RF). Thus, the monitoring of a large area can be realized by a sensor network with RF communication. This type of network is called wireless sensors network (WSN). But there is a disadvantage in this type of sensor network, which is the power consumption. In many applications, such as high voltage transmission line monitoring, there is no power supply to the sensor or WSN. Thus, a battery should be used to feed the monitoring system. So, we have to observe the most critical point in WSNs, which is energy consumption [14]. The largest part of energy consumption in a sensor node occurs during the data transmission or reception. For this reason, the main goal of most algorithms designed for WSN applications is the communication cost reduction in terms of energy consumption [15–17]. For example, the use of in-network aggregation operators [15–17] is an efficient strategy to reduce the volume of data transmitted in a given WSN and consequently the energy consumption.

Two other important concepts in the electrical systems monitoring applications are reliability and scalability. Thus, techniques or strategies to enable reliable communication of the wireless sensor network are very important, as is the possibility of adding new sensors, in a transparent and automatic way, in the sensor network. To meet these and other requirements of the application, the following modules have been created: the query processor module and the data aggregation module (DatAggrMod). In the next subsection, we will present the query processor module.

2.2.1. The Query Processor Module. Before presenting the query processor module, we must show how to represent the sensor network topology to a smart grid application. The electrical system is typically hierarchical. For example, a low voltage transformer supplies power to the various consumers, the tower of a high voltage transmission line concentrates multiple insulators, and a substation feeder can connect to a group of transformers, among other examples. A hierarchical WSN, which also has been used in [11, 18], can be represented by the clusters which represent groups of sensor nodes. Each cluster has a leader called the cluster head (CH) which coordinates communications with the sensor nodes (end nodes (EN)) in its cluster, among other tasks.

To present the applicability of the query processor module and the data compression and aggregation techniques in a WSN, we use, as an example, a WSN installed on a high-voltage transmission line to monitor the leakage current of electrical insulators. The use of sensors and smart controls for the monitoring and operation of high-voltage transmission lines also includes the concept of smart grid. The example

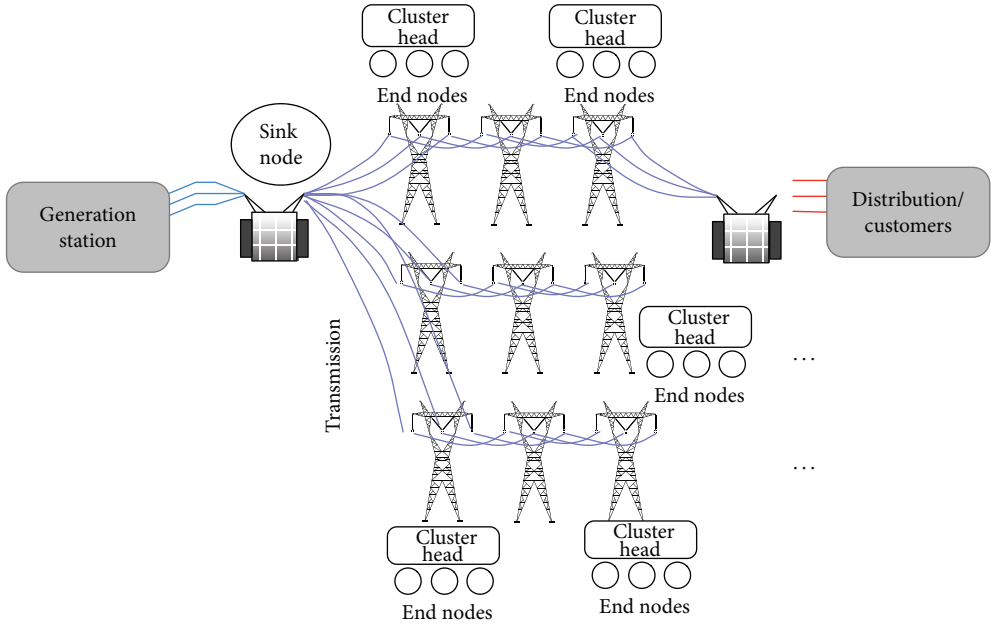

Figure 3: WSN topology installed on the high voltage transmission system.

presented in this paper for the electrical insulator monitoring of high-voltage transmission lines can also be applied to the insulator monitoring of a distribution network, which is more related to the concept of smart grid. In this example, a WSN is composed of three sensor node types: (i) end node (EN, sensor node), which is responsible for sensing and transmitting sensed data to a cluster head. Each insulator chain has only one end node; (ii) cluster head node (CH), which compresses and aggregates sensed data, forwards the (sub)query for the end nodes, and returns the query results to the sink node (base station). Each high-voltage transmission tower has only one cluster head node and (iii) sink node (base station), which contains the query processor, data aggregation module (DatAggrMod), and the data prediction module (DatPredMod). The sink node is installed in the electricity substation. Figure 3 presents an example of a hierarchical WSN installed on the high voltage transmission system. In the figure, a cluster is the group of ENs (sensor nodes) at each transmission tower. Each cluster has a CH which coordinates its cluster and communicates with other CHs. And, finally, the sink node that is located at the electricity substation is responsible for coordinating the communication of the WSN, as well as to establish the rules and policies for sending queries and receiving responses from the network.

The query processor module runs on the sink node which is connected to the RTU via Ethernet. Through RPC (remote procedure call), the wSn converter module, at the RTU, sends a query request to the query processor module, at the sink node. In Figure 4, the architecture of the sink node is shown. Based on the figure, the sink node receives a query request from the RTU with a message structure compatible with the message structure used by the WSN. Subsequently, the query processor assembles the query packet and sends it to the sensor network. As an example, the WSN of this work is installed on the high voltage transmission line. The answer to this query is a simple or aggregate response packet which depends on the sensor node identifier included in the query. If the query is directed to an individual sensor node, the response is a simple message, and if it is directed to all the sensors of the network, the response is an aggregated message.

The data packet shown in Figure 5 is the message structure used by the WSN. It is observed that the message structure designed for the WSN is similar to the message structure used by the DNP3 protocol. This similarity simplifies the conversion between the protocols, making it faster and more reliable.

As shown in Figure 5, the message consists of some fields, and each field has its functionality as presented below.

(i) Start (start byte): this field is the message start identifier. In this field, the value 0x7E (or 126 decimal) is used to indicate the start of the frame (packet/message).

(ii) Length: this field represents the message(frame) size. The value of this field indicates the number of bytes contained between the fields "length" and "CRC."

(iii) Func code (function code): this field contains the function code which is the code used to perform an operation on the WSN. As an example, the function code "0x01" is the read operation (READ) only on the digital values of the sensor node.

(iv) aux (auxiliary) is an auxiliary field to complement the field "function code." For example, the function code "0x04" represents the reading (READ) operation only on the analog values of the sensor node that exceeded

Table 1: The sensor header field.

Sensor header—1 Octet (8 bits)							
1°	2°	3°	4°	5°	6°	7°	8°
Signal type	Length	Length	Sensor type	Sensor type	Sensor type	Sensor type	Sensor type

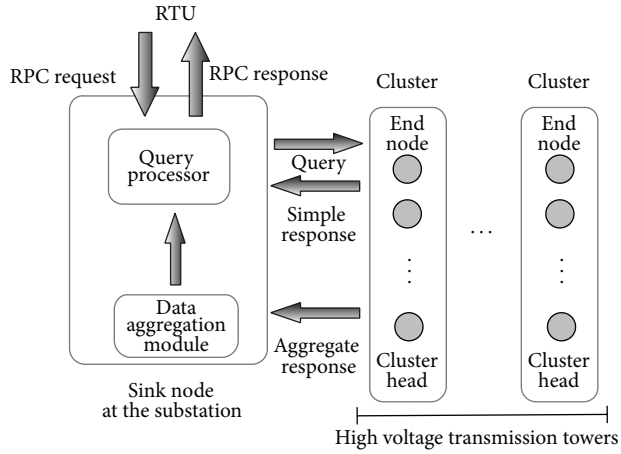

Figure 4: The block diagram of the sink node architecture.

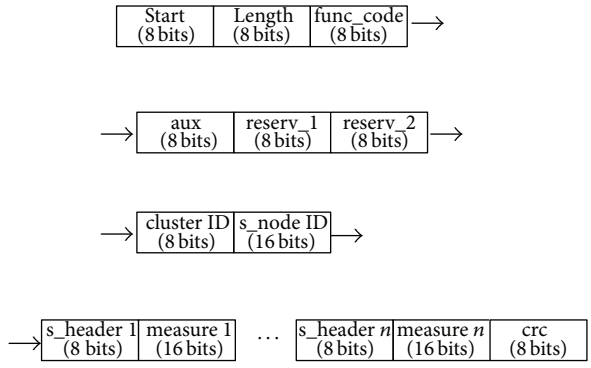

Figure 5: The message structure of the WSN.

threshold values. The field "aux" contains this threshold value. As an example, the code "aux=0x14" is a threshold value of 2.

(v) reserv_1 and reserv_2 (reserved 1, reserved 2): these fields are reserved for future implementations of other communication functions.

(vi) Cluster ID: this field is used to identify the cluster in a hierarchical network. As an example, for cluster 0 this field must contain the value "0x00." In the case of a nonhierarchical WSN or a query for all network nodes, this field should present the value "0xFF."

(vii) s_node ID (sensor node ID): this field is used to identify the sensor node (EN) in the cluster, in the case of the hierarchical network, or the sensor node in a nonhierarchical WSN. In the case of a query for all network nodes, this field should be changed to "0xFFFF."

```
def serial_setup():
  ser = serial.Serial('/dev/ttyAMA0',
  9600, timeout = 0)
  return
```

Algorithm 7: Fuction serial_setup().

(viii) s_header 1 (sensor header 1) is a header for the first sensor to be read. In the same way, sensor_header *n* is a header for the *n*th sensor to be read. In this header, the information about the sensor signal type (analog or digital), the measured value size (8, 16, or 32 bits), and the sensor type (temperature, pressure, humidity, presence, level, gas, voltage, current, and other) are presented. Thus, this field can be presented as shown in Table 1.

(ix) Measure 1 (measurement 1) is the measurement value of the first sensor which can contain the 8, 16, or 32 bits value. In the same way, the measure *n* is the measurement value of the *n*th sensor.

(x) crc is the cyclic redundancy check (CRC) code that is an error-detecting code commonly used in digital networks, storage devices, or communication networks to detect accidental changes to raw data.

Upon receiving a query request from the RTU, the query processor will create a query message (packet) with the structure shown in Figure 5. Subsequently, the query message is sent to the WSN. The algorithm used by the query processor to perform this query is given below.

Query Processor Algorithm. The query processor, which runs on the sink node, was developed in the Python language and can be explained by the following algorithm.

(1) In the first query, the Raspberry serial interface is configured to communicate with the XBEE radio which is coupled to it, we have created the function in Algorithm 7, the radio has been configured to communicate with a baud rate of 9600 bps. For this configuration, we have created the function in Algorithm 7.

(2) Due to the first query, the Raspberry (sink node) must communicate with the XBEE radio in the AT command mode and sends the configuration words to the radio. These words configure the radio to transmit messages in the broadcasting mode (the "ATDL 0000FFFF" command configures this mode). It is important to note that the WSN addressing is done by the fields "sensor node ID" and "cluster ID" of the

```
def radio_config():
  ser.write('+++')
  time.sleep(1)
  ser.write('ATDL 0000FFFF\r')
  ser.write('ATWR\r')
  ser.write('ATCN\r')
  time.sleep(1)
  return
```

Algorithm 8: Fuction radio_config().

```
from wSn import *
a = wSn()
def query(snID=0):
  a.funcao = 0
  a.aux = 0xff
  a.sensorNodeID=int(snID)
  a.clusterID=0x00
  a.addMeasure('a',16,'humidity',0)
  for k in bytearray(a.encode()):
      ser.write(pack('B',k))
  time.sleep(1)
  x = wait_ack(snID)
  return x
```

Algorithm 9: Function query().

query message and not by the address configured on the radio. The function in Algorithm 8 has been developed for the radio configuration.

(3) After the radio configuration, the query message is packaged according to the message structure shown in Figure 5. A new class called wSn has been created in order to facilitate the message encapsulation. As shown below, the code performs a query to the sensor node 0 of the cluster 0, with the function code (a.funcao) set to 0, which is a read operation (READ) of the sensor values indicated by the header sensor. The sensor header is configured by the method "a.addMeasure()," which sets the read signal type as analog ("a"), the measured value size of 16 bits, and the sensor type of humidity. See Algorithm 9.

To increase the reliability, the communication performed by the query processor uses the remote invocation method. As an example, after the query sending process, the query processor has to wait for an acknowledgment message to confirm the message reception successfully, which is achieved by checking the CRC code in the sensor node. The function "wait_ack()" presented in the code above performs this confirmation.

(4) In the case of a successful acknowledgment message, the query processor will wait for the query response. First, it waits for the byte start to arrive and then begins receiving the frame fields which are stored in a buffer called_buff. Subsequently, a method called "a.decode()" is invoked to perform the decoding of the WSN response frame. In other words, it converts the response frame to the message structure presented in Figure 5. The code presented below performs this procedure.

In the case of the acknowledgment message being unsuccessful, due to the mismatch of the CRC code, a new query should be performed by the query processor. See Algorithm 10.

The next algorithm details the operation of the sensor node (EN). This algorithm allows the reception of the query, the reading of the sensors, and the subsequent submission of the responses to the CH.

Sensor Node Algorithm. The Arduino language, which is based on the C language, has been used for the development of the sensor node program. The algorithm that the sensor node has used to answer the query is simpler. The following steps present the query response strategy.

(1) First, the sensor node checks for the arrival of a complete message at its buffer.

(2) Subsequently, the sensor node compares the first byte with the start byte. If it is true, the next step is executed. See Algorithm 11.

(3) The sensor node calls the function "unpack_message()" to extract the message fields and compares the cluster ID and sensor node ID fields which identify the destination cluster and node. If the message is not directed to this node, the message is discarded. Otherwise, the next step is performed.

(4) Then, the function "func_code_check()" is called to extract and check the function code. This function returns a code which has been used to identify what types of sensors and signals must be read.

(5) Finally, the response message (with the read measurement values) is sent back to the sink node through its CH.

Sensor Node Algorithm for Scalability. We have developed an algorithm to enable scalability at the sensor network without having to perform manual configuration of the sensor nodes. The code portion in Algorithm 12 performs this procedure.

The whole process begins when a new sensor node is installed on the electrical system. The new sensor node waits for messages from some CHs which are closer. For each received message, the new sensor node stores the radio signal strength (RSSI). Subsequently, it compares the powers of the received signals and selects the cluster head that has sent the message with the higher radio signal power. The following steps explain the algorithm.

(1) First, the sensor node checks for the arrival of a complete message at its buffer and compares the first byte with the start byte. If it is true, the message is unpackaged. The cluster ID field value is stored. Subsequently, the function "ATCommand_resp()" is called. This function configures the radio to the AT command mode and sends the *ATDB* AT command to the sensor node radio. This command returns the radio signal strength of the last received message.

(2) Then, the sensor node waits for other messages. It checks for messages of different CHs and stores their IDs in its array. For each received message, the radio signal strength is stored.

```
...
while ser.inWaiting() > 0:
  _buff = ''
  x = ser.read()
  y = unpack('B',x)[0]
  if (int(y) == 126):
    _buff += x                  # startByte
    _buff += ser.read()         # size
    _qtdeM = ser.read()         # number_measures
    _buff += _qtdeM             # assign
    _buff += ser.read()         # func_code
    _buff += ser.read()         # aux
    _buff += ser.read()         # reserv_1
    _buff += ser.read()         # reserv_2
    _buff += ser.read()         # clusterID
    _buff += ser.read()         # sNodeID-MSB
    _buff += ser.read()         # sNodeID-LSB
    for _i in range(int(unpack('B', _qtdeM)[0])):
      _buff += ser.read()         # s_Header
      _buff += ser.read()         # MeasureMSB
      _buff += ser.read()         # MeasureLSB
      _buff += ser.read()         # CRC
      a.decode(_buff)
...
```

Algorithm 10: Waiting response.

```
...
void loop() {
 delay(500);
 if (Serial.available() >= 18) {
  pac.start_byte = Serial.read();
  if (pac.start_byte == 126) {
   unpack_message();
   if(pac.cluster_id == 0 &&
   pac.node_id_lsb == 0){
    for(int i = 0;
    i < pac.num_measure; i++){
     pac.sens_header[i] = Serial.
     read();
     pac.measurement[i] = Serial.
     read();
    }
    func = func_code_check();
    read_sensors(func);
    send_package(126, pac.length + 8,
    4, 7, 255, 255, 255, 0, 0, 0,
    pac.sens_header,
    pac.measurement);
   }
  }
 }
}
...
```

Algorithm 11: Sensor node code.

```
...
while (Serial.available() <= 18){
   delay(1000)
}
pac.start_byte = Serial.read();
if (pac.start_byte == 126){
  unpack_message();
  clusterID[0] = pac.cluster_id;
  strength_r[0] = ATCommand_resp();
}
for (i = 1;i <= 3;i++){
  while (Serial.available() <= 18){
   delay(1000)
  }
  pac.start_byte = Serial.read();
  if (pac.start_byte == 126){
   unpack_message();
   if (!cluster_thereis(clusterID)){
    clusterID[i] = pac.cluster_id;
    strength_r[i] = ATCommand_resp();
   }
  }
}
cID = Compare_strength(strength_r);
send_id(cID);
wait_sID();
...
```

Algorithm 12: Code for scalability.

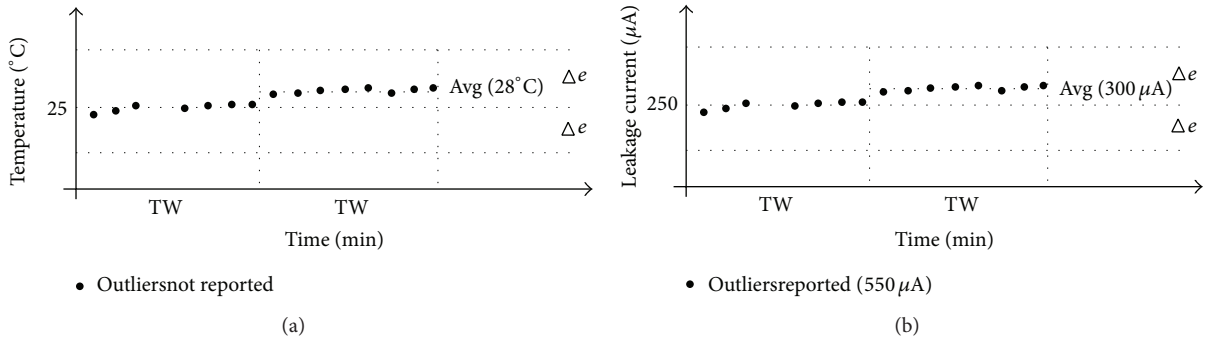

Figure 6: Graphics of variables measurements.

(3) After this, the new sensor node compares the radio signal strength of the received messages and selects as its leader (cluster head) the cluster head that has sent the message with the highest radio signal strength.

(4) And finally, a message is sent to the selected CH to inform it of the presence of a new sensor node. The CH answers this message and sends back the sensor ID value (identifier) in its cluster. This value is based on the sensor ID table which is stored in each CH. Then the cluster head has to send the information of the new sensor node to the sink node.

2.2.2. Data Aggregation and Compression in the WSN Query. We presented in Section 2.2.1 the possibility of performing a query for all the network nodes. To do this, we have to change the cluster ID and sensor node ID fields to 0xFF to 0xFFFF, respectively. Thus, all nodes have to make a response and return it to the sink node. In this case, as all the sensor nodes answer a query, the number of radio transmissions and receptions increases considerably, while increasing the energy consumption of the sensor network. Therefore, the use of techniques or strategies to reduce the amount of data in the network is required. In this section, the DatAggrMod (data aggregation module) module is described, as well as its strategies to aggregate and compress data.

2.2.3. DatAggrMod (Data Aggregation Module). We will use a real example (with a real data set collected in [19]) of a WSN installed on the high voltage transmission line to illustrate the module that performs the data aggregation and compression. This real example is based on the application that aims to generate an alert if the leakage electrical current exceeds a threshold value. The leakage current depends on the insulator structural state besides other variables such as temperature and humidity. In this case, the cluster is represented by a group of sensor nodes installed on each transmission tower, and each sensor node is installed on the insulator chain to detect leakage current and other variables, such as temperature, humidity, and noise signal. Figures 3 and 4 demonstrate this topology.

The data aggregation module is implemented in each cluster head and in the sink node. In the cluster head, its goal is to aggregate and compress sensed data received from the end nodes and from the nearest (or the closest) cluster head. In the sink node, it aims to realize a data aggregation and compression reverse operation.

Due to continuous and linear variation of some variables, such as temperature and humidity, we have used the average value of the measurements as an aggregation strategy. If the query is received in a time window, the average value is produced by the sum of the measured values divided by the number of measurements which occurred in this time.

To explain the strategy, we suppose a query sent by the cluster head to the end nodes of its cluster. To answer the query, the end nodes return the query results to the cluster head. The data aggregation module, in the cluster head, produces an average value of the received measurements. Equation (1) represents the average value of the sensed variables:

$$Aq_{\mathrm{sd}} = \frac{\sum_{j=1}^{6} \left(\sum_{\mathrm{ts}=1}^{\mathrm{te}} \mathrm{sd}\left(\mathrm{ts}, j\right) / \mathrm{te} \right)}{6}. \quad (1)$$

In (1), the Aq_{sd} is the average value of variable measurements in the time interval (te–ts) of the six insulator chains. In this way, the average value equation of the leakage current is produced by using index lc instead of index sd (*sensed data*). The same analysis is used for the average value of the temperature, humidity, and noise signal variables. Regarding measurements of temperature and humidity, of which variations are continuous and slow, the outliers are considered only as errors and are not reported. On the other hand, variations of leakage current and noise signal measurements may not be continuous and slow, and their changing may be reported as a valid measurement. In Figure 6, the top graph presents the temperature measurements in the time window TW, with the average value of measurements represented by Avg. It is important to observe that the outliers are not reported. In the same figure, the bottom graph presents the leakage current measurements in the time window TW, with the average value of the measurements represented by Avg. In this case, the outliers may be reported to the sink node. In Figure 6, the outliers are measurements whose values exceed Avg+Δe. Based on [19], the temperature database collected in the tests does not present any instantaneous difference greater than 5°C. Then, Δe for temperature is 5. The same analysis is made for other variables. The leakage current measurement

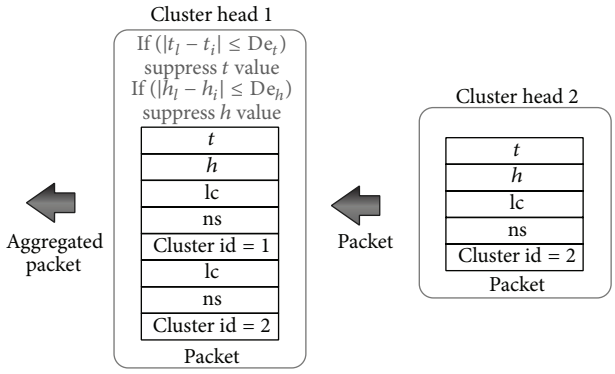

Figure 7: Packet flow between clusters.

can change instantly due to the atmospheric discharge. Thus outliers may represent correct and valid values.

The aggregation strategy is also used when incoming packets are arriving at a cluster head. The temperature and humidity variables have a tightly spatial and temporal correlation. These variables can be aggregated based on their similarity. As an example, in Figure 7, cluster head 2 sends to cluster head 1 its nonaggregated packet. Cluster head 1 extracts the temperature and humidity fields from the packet and compares them with its temperature and humidity average values. If the difference between the values is less than the factor De (De_t for temperature and De_h for humidity), the local value is suppressed and an aggregated packet is created. Based on the database collected in [19], there is no substantial variation of leakage current when temperature variation is less than 1°C. Then the temperature factor is $De_t = 1$. The same approach is used for humidity whose factor is $De_h = 5$.

After the action of data aggregation in the cluster head, the data compression is implemented. In the next subsection, this strategy is presented.

2.2.4. Data Compression in the DatAggrMod. After sensed data aggregation takes place, a data compression strategy is further used to reduce the sensed data packet size, which will be sent back to the sink node. In our experiments, based on [19], we have observed that the leakage current variation sensibility due to temperature variation and humidity variation presented important values, with temperature variation between 20°C and 50°C and humidity variation between 50% and 89%. In this way, we can represent the average value of the measurements with a lower number of bits, instead of using one byte to represent each value. With these temperature and humidity variations, we can use 2 bits to represent the second decimal digit (e.g., ten **2**5) of the average values of the temperature and humidity and 4 bits to represent the unity (e.g., unity 2**5**) of the average values of the temperature and humidity. Then we can use only 6 bits instead of 8 bits for each variable. The noise signal ns, which is related to the leakage current, presented a variation between 0 dB and 15 dB and can be represented by 4 bits. And, finally, the leakage current has a variation between 1560 μA and 2960 μA, with the variation of its second decimal digit representing the sensibility due to other variables. So we have 140 (296–156) units to represent the leakage current variation, which can use 8 bits to report it.

It is important to observe that the initial packet of 5 octets (bytes) used to represent the 4 measurements was compressed to 3 octets (bytes). This strategy represents a reduction of 40% in the packet size, and, consequently, a reduction in the network energy consumption. After the compression action, the aggregated and compressed packet is forwarded to the next cluster head or to the sink node, if it is closer.

When the aggregated packet reaches the sink node, the DatAggrMod module performs the inverse operation of the data aggregation and compression. The compressed data of size 4 and 6 bits are converted to their original size of 8 bits. In the same way, the compressed data of 8 bits are converted to the size of 16 bits. The inverse operation of the data aggregation is simpler, since the measurements that have been received from their respective clusters are replicated to all other clusters.

3. Results and Discussion

3.1. Results of Running the Middleware in the RTU. To validate the middleware operation in the RTU, we have performed a remote access to the concentrator through SCADA, and we have produced a sequence of 50 queries to all nodes of the WSN. Each query should get just the analog value of the temperature (with a 16 bit size) of each sensor node. For validation, we have used a WSN with 1 cluster head and 4 end nodes, as shown in Figure 8. Two parameters have been analyzed in these tests: the correctness and the response time of queries. These parameters which have been obtained by the queries in the WSN were compared with the same parameters which have been obtained by the queries performed earlier in the low voltage (power) switchgears. Currently, the low voltage (power) switchgears are already monitored by SCADA, in the substation, and this monitoring has been validated and it has proved to be quite a satisfactory operation.

We have created a script to perform the queries automatically, and we have stored the query answers in the log files. A timestamp field has been added in each query packet to record the response time of the queries, and the correctness is analyzed by comparing the correct value of the measurements. In Figure 9, a piece of the log file content is presented. In the piece of the code, we can observe the time stamp which is represented by the date and time, the identifier of the sensor node, represented by the field "ID," and the sent and received data packets.

Based on comparison of the LPS (low power switchgear) and WSN query log files, the response time and correctness graphics are presented in Figures 10 and 11, respectively.

As shown in Figure 10, the horizontal axis represents the query which is performed on each sensor node and LPSs (low power switchgears). The vertical axis represents the response time, in seconds. In other words, it is the time difference between the query packet output of the concentrator and the arrival of the response packet on the concentrator. The

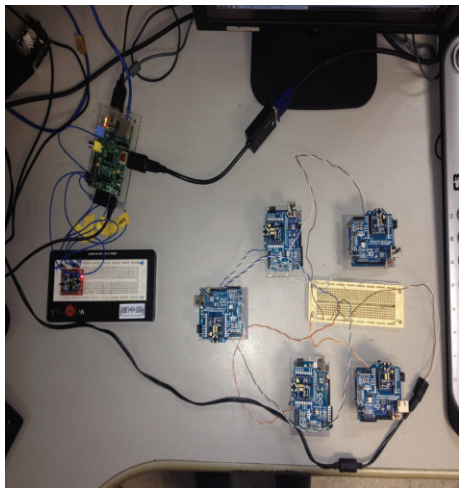

Figure 8: The WSN picture.

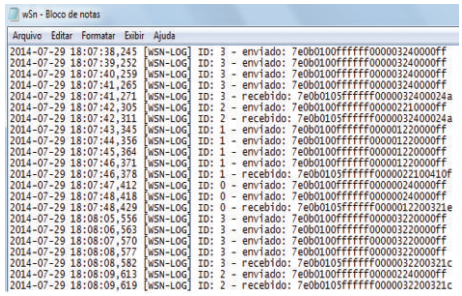

Figure 9: The piece of the WSN query log file.

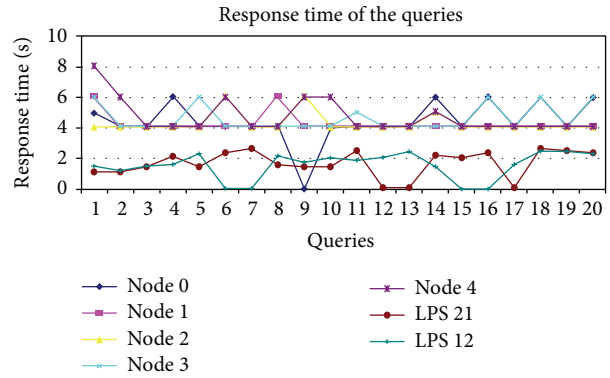

Figure 10: Comparison of the query response time.

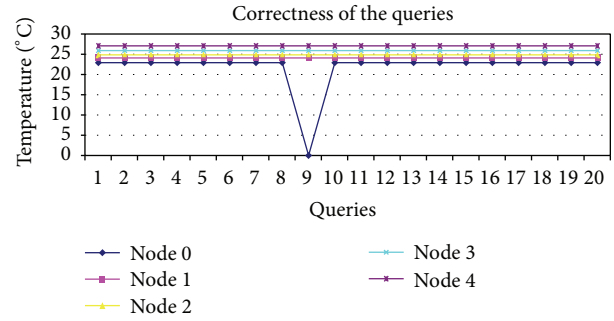

Figure 11: Correctness of the query responses.

colored lines represent the response time variations of five sensor nodes (nodes 0 to 4) and two LPSs (LPS 21 and LPS 12). For each sensor node, five query attempts are performed, with a timeout of one second between them. This query strategy is called the remote invocation method and is used to provide greater reliability in the communication. We have used a time interval of 10 seconds between queries (which are represented by the horizontal axis) on the devices. In this graphic, it is important to note that the WSN response time has little difference in the LPS monitoring response time. The average time difference which has been observed is only 2.8 seconds. This time difference can be explained by the following. Due to noise in the environment, radio frequency signals may experience interference, and errors can be added to the message packets received or sent by the sensor nodes. Thus, the remote invocation method was added to increase the reliability of the WSN communication. This method consists of sending the message to the destination node and waiting for an acknowledgment message of the received packet success. This method increases the reliability in the communication but also causes an increase in the communication time of the WSN. Furthermore, synchronization between sending and receiving packets in the WSN, presented in this work, occurs by periodically checking an indicator byte of packet start, called start byte. Periodically checking this byte also causes an increase of the communication time in the WSN. In recent experiments it was observed that a decrease in the period of start byte checking also decreases the sending and receiving time of message packets. Then this difference can be reduced by reducing the communication timing (period of start byte checking) between the sink node and the end nodes (sensor nodes). Another important fact to be noted is that the communication failures which are represented by the values of response time equal zero. The communication between the concentrator and the LPSs has shown more failures. These failures occur due to the fact that communication between the RTU and LPSs is performed by a GPRS (general packet radio service) MODEM which is susceptible to connection faults. As an example, the communication between the RTU and LPS 12 has presented 4 response failures among 20 queries performed. And nodes 1 to 4 have not shown any failure for 20 queries performed. So, this is an advantage of the monitoring by the WSN. The communication procedure between the RTU and LPS is similar to the procedure between the RTU and the WSN but with a difference: server LPS srv communicates with the LPS via GPRS. The srv.py, which is a method of the LPS srv, has a multithreaded socket that waits for a connection request from an LPS. After connection, the id (identifier) of the LPS is passed to the LPS srv that initiates the invocation of the "advert()" method via RPC to instantiate the new device. All remaining procedures follow as presented in Section 2.1, until the server LPS srv receives the query parameters of the LPS. Subsequently, the server LPS srv sends the query to the destination LPS via GPRS.

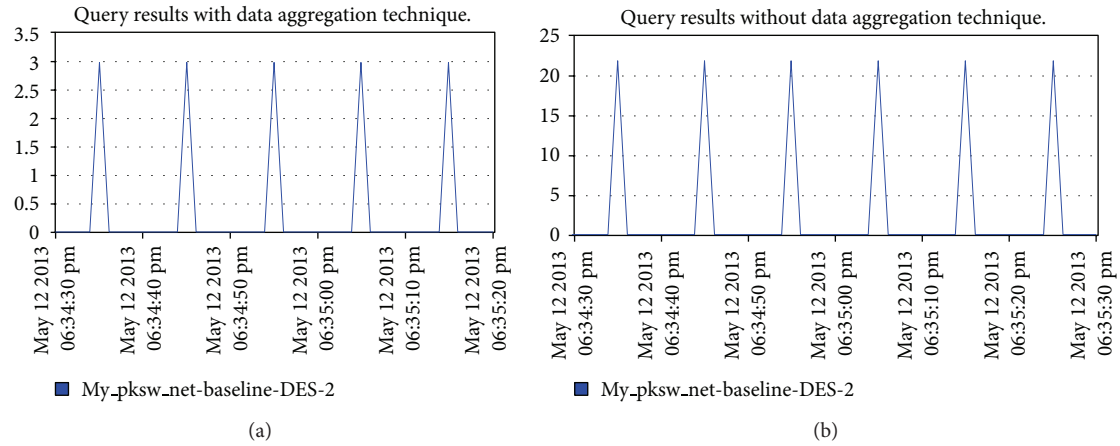

Figure 12: Number of packets received in cluster head 1 due to query results.

As shown in Figure 11, the horizontal axis represents the query which is performed on each sensor node. The vertical axis represents the temperature, in ˚C. The colored lines represent the temperature measurement variations of five sensor nodes (nodes 0 to 4). To validate the correctness of the communication and protocol conversion, we have configured the temperature measurement field of each sensor node to the following values: node 0 with temperature of 23˚C, node 1 with temperature of 24˚C, and so on until node 4. Subsequently, we have performed queries for temperature reading of all sensor nodes, and we have compared the received measurements with the temperature values configured in the sensor nodes. As seen in Figure 11, 100% of the temperature measurements received by the concentrator have coincided with the values configured in the sensor nodes. We have observed only one exception in node "0" which has presented a receiving error at the ninth query. It is important to note that the SCADA recognizes the WSN as a smart grid device and can visualize its measured values in the real-time mode.

3.2. Simulation Results of the Protocol in the WSN. We have used the OPNET modeler 14.5 as a simulator to validate the WSN protocol, specifically related to data aggregation. Our experiments are based on a campus type scenario. This scenario is represented by a group of clusters and 1 sink node, and each cluster has 1 cluster head and 2 end nodes. Each cluster is installed on the transmission tower, and the sink node is installed into the electrical substation. We have analyzed two scenarios as depicted below.

(i) A scenario with 24 sensor nodes and 1 sink node: it is equivalent to 8 transmission towers. In this scenario, we have fixed 25˚C for the temperature and 70% for the humidity for clusters 2, 3, and 4; 27˚C for the temperature and 60% for the humidity for clusters 5 and 6; and 29˚C for the temperature and 50% for the humidity for clusters 7 and 8.

(ii) A scenario with 99 sensor nodes and 1 sink node: it is equivalent to 33 transmission towers. In this scenario, we have fixed 25˚C for the temperature and 70% for the humidity for clusters 2 to 25; 27˚C for the temperature and 60% for the humidity for clusters 26 to 60; and 29˚C for the temperature and 50% for the humidity for clusters 61 to 99.

For each scenario, we have generated ten queries (every 10 seconds) for the temperature and the humidity for all sensor nodes in the WSN. Two situations have been implemented for each scenario: in one situation, we have used a data aggregation technique to reduce the number of packets traveling on the network, and in the other situation, we have not used the data aggregation technique. Figure 12 represents these two situations for scenario (ii) (with data aggregation technique) and for scenario (i) (without data aggregation technique).

Figure 12 shows the data flow in the radio receiver of cluster head 1 for two situations: Figure 12(a) represents scenario (ii) (with data aggregation technique) and Figure 12(b) represents scenario (i) (without data aggregation technique). This cluster head was chosen due to its closeness to the sink node and relays all queries and results to the WSN. So it is the node with the largest number of retransmissions in the network and with the greatest criticality as to the energy consumption. In Figure 12(a), we have noted that, due to scenario (ii), with 3 groups of parameters (temperature and humidity) and using data aggregation technique, only 3 data packets have been received by the radio receiver of this node as a result of each query in the network. With data aggregation technique disabled for scenario (ii), we have observed the reception of 90 data packets.

Figure 12(b) represents the data flow in the radio receiver of cluster head 1, but now scenario (i) is addressed, with 3 groups of parameters (temperature and humidity) configured for all nodes, as shown in scenario (i). In this configuration, we have disabled the data aggregation module for all cluster heads. We have observed that 22 data packets have been received for each query in the WSN. In this case, there was packet loss, because 23 data packets were expected. When we have simulated with data aggregation technique enabled for this scenario, we have observed a reception of only 3 data packets. This is due to the scenario 1 has three cluster groups

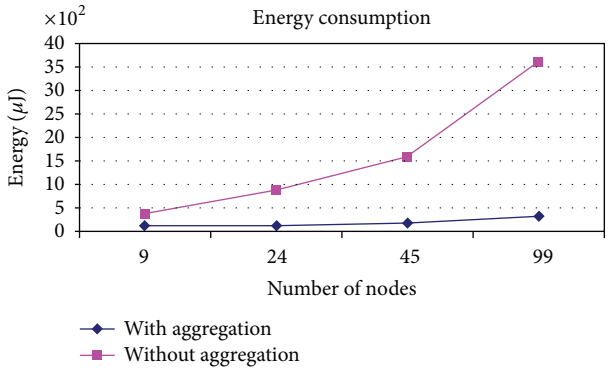

Figure 13: Energy expended by the receiver of cluster head 1.

with sensors subjected to the same temperature and the same humidity. So with the data aggregation technique enabled for this scenario, only 3 data packets must reach the sink node. For this scenario, a comparison between the application of the data aggregation technique (reception of 3 packets) and its absence (reception of 22 packets) shows a reduction of 86.36% in the energy consumption of the CH1 (Cluster Head 1) radio receiver.

Figure 13 presents a graph that relates the energy consumption of the CH1 radio receiver with the number of network sensor nodes. We have used, for energy consumption of the radio receiver, the value of 0.5 μJ/bit [20] and 10 bytes (10 octets) for the packet size which corresponds to a basic package of a query made by the sink node or the response of a sensor sode (EN) of the WSN. This basic package consists of the following fields: start, length, func code, aux, cluster ID, s_node ID, s_header1, and measure 1. The blue and red lines represent, respectively, the energy consumption of the CH1 radio receiver for scenarios (i) and (ii) (24 and 99 nodes) with and without data aggregation technique enabled in all cluster heads. We have also used scenarios with 9 and 45 nodes to validate this test. The best result of the simulations was achieved with scenario (ii), with the reduction of energy consumption of 3,600 μJ (without aggregation) to 320 μJ (with aggregation), representing a reduction greater than 10 times. The energy consumption of 320 μJ is related to the energy expended by the CH1 transceiver to relay the query packet made by the sink node and receive the answer packets (related to this query) from all sensor nodes (EN) of the WSN using the data aggregation technique. The energy consumption of 3,600 μJ is seen in the same way, but the WSN does not use the data aggregation technique. To give an idea about these values of energy consumption, suppose a sensor node with a battery capacity of 1000 mAh that needs to send and receive packets of data every minute. The communication energy consumption of 320 μJ allows the battery to have a lifetime of approximately 21 years. If the sensor node had a communication energy consumption of 1500 μJ, the battery lifetime would be about 4 and a half years which would be a critical lifetime.

We have also observed that the reduction in the energy consumption is scalable. That is, if the number of sensors increases, the gain in the reduction of energy consumption (using an aggregation technique) also increases.

4. Conclusions

In this paper, we present a middleware for integrating a WSN in a smart grid, as well as the techniques and strategies to enable better WSN reliability and lifetime. This integration allows the sensor nodes to understand the requests made by the SCADA software and to respond to these requests using the default protocol of the Smart Grids.

Some challenges were overcome in the development of this work, including the conversion of message structures, the correctness and response time to requests made by SCADA software in the sensor network, the reliability of the WSN communication, and the WSN growth with dynamic addressing, among others.

The data aggregation and compression strategies which were used to reduce the WSN energy consumption, in turn, are simple to implement, and they do not require much processing in the cluster head. The data compression technique used in this study gave a reduction of 40% in the size of the data packet, and, consequently, a reduction in the network energy consumption. It was also observed that the data aggregation technique has allowed a reduction in the WSN energy consumption of 10 times, when it has been applied to a WSN with a scenario represented by 99 sensor nodes. In this scenario, the sensors are divided into 3 groups, and each group of sensors is subjected to the same temperature and humidity.

The simulation experiments have demonstrated that the middleware enables the integration of a WSN with a smart grid attending the constraints of response time and correctness from a smart grid application. Furthermore the results have proved that the combination of the aggregation and compression mechanisms is effective and has increased significantly the WSN lifetime.

As future work, we will focus more on the WSN query strategies. We should implement a context-aware query, based on event prediction, using artificial intelligence or pattern recognition techniques. We also intend to implement a voting (election) of cluster head based on communication reliability due to radio frequency and corona interference in the transmission line.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] R. E. Volkmann and A. S. Bretas, *Integracao de subestacoes atraves do protocolo IEC-61850*, Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil, 2012.

[2] M. E. Brak and M. Essaaidi, "Wireless sensor network in home automation network and smart grid," in *Proceedings of the*

International Conference on Complex Systems (ICCS '12), pp. 1–6, IEEE, Agadir, Morocco, November 2012.

[3] C. Qian, Z. Luo, X. Tian, X. Wang, and M. Guizani, "Cognitive transmission based on data priority classification in WSNs for Smart Grid," in *Proceedings of the IEEE Global Communications Conference (GLOBECOM '12)*, pp. 5166–5171, Anaheim, Calif, USA, December 2012.

[4] P. Zhang, F. Li, and N. Bhatt, "Next-generation monitoring, analysis, and control for the future smart control center," *IEEE Transactions on Smart Grid*, vol. 1, no. 2, pp. 186–192, 2010.

[5] V. C. Gungor and G. P. Hancke, "Industrial wireless sensor networks: challenges, design principles, and technical approaches," *IEEE Transactions on Industrial Electronics*, vol. 56, no. 10, pp. 4258–4265, 2009.

[6] R. Braunlich, M. Hassig, J. Fuhr, and T. Aschwanden, "Assessment of insulation condition of large power transformers by on-site electrical diagnostic methods," in *Proceedings of the Conference Record of the IEEE International Symposium on Electrical Insulation*, pp. 368–372, Anaheim, Calif, USA, April 2000.

[7] M. Qiu, W. Gao, M. Chen, J.-W. Niu, and L. Zhang, "Energy efficient security algorithm for power grid wide area monitoring system," *IEEE Transactions on Smart Grid*, vol. 2, no. 4, pp. 715–723, 2011.

[8] M. T. Vo, M. T. Nguyen, T. D. Nguyen, C. T. Le, and H. T. Huynh, "Towards residential smart grid: a practical design of wireless sensor network and Mini-Web server based low cost home energy monitoring system," in *Proceedings of the International Conference on Advanced Technologies for Communications (ATC '13)*, pp. 540–545, October 2013.

[9] C.-W. Lu, S.-C. Li, and Q. Wu, "Interconnecting ZigBee and 6LoWPAN wireless sensor networks for smart grid applications," in *Proceedings of the 5th International Conference on Sensing Technology (ICST '11)*, pp. 267–272, Palmerston North, New Zealand, December 2011.

[10] A. M. Gaouda, "Adaptive Partial Discharge monitoring system for future smart grids," in *Proceedings of the 39th Annual Conference of the IEEE Industrial Electronics Society (IECON '13)*, pp. 4982–4987, November 2013.

[11] I. Al-Anbagi, M. Erol-Kantarci, and H. T. Mouftah, "A delay mitigation scheme for WSN-based smart grid substation monitoring," in *Proceedings of the 9th International Wireless Communications and Mobile Computing Conference (IWCMC '13)*, pp. 1470–1475, July 2013.

[12] S. Srivastava and P. Srivastava, "Performance analysis of sun rpc," in *Proceedings of the National Conference on Parallel Computing Technologies (PARCOMPTECH '13)*, pp. 1–9, February 2013.

[13] H. Karl and A. Willig, *Protocols and Architectures for Wireless Sensor Networks*, John Wiley & Sons, West Sussex, UK, 2005.

[14] I. Dietrich and F. Dressler, "On the lifetime of wireless sensor networks," *ACM Transactions on Sensor Networks*, vol. 5, no. 1, article 5, 2009.

[15] A. Brayner, A. Lopes, D. Meira, R. Vasconcelos, and R. Menezes, "Toward adaptive query processing in wireless sensor networks," *Signal Processing*, vol. 87, no. 12, pp. 2911–2933, 2007.

[16] A. Brayner, A. Lopes, D. Meira, R. Vasconcelos, and R. Menezes, "An adaptive in-network aggregation operator for query processing in wireless sensor networks," *Journal of Systems and Software*, vol. 81, no. 3, pp. 328–342, 2008.

[17] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos, "Processing approximate aggregate queries in wireless sensor networks," *Information Systems*, vol. 31, no. 8, pp. 770–792, 2006.

[18] I. Al-Anbagi, M. Erol-Kantarci, and H. T. Mouftah, "An adaptive QoS scheme for WSN-based smart grid monitoring," in *Proceedings of the IEEE International Conference on Communications Workshops (ICC '13)*, pp. 1046–1051, Budapest, Hungary, June 2013.

[19] P. R. Araujo and R. Ramos, *Indoor Tests of Insulator Chain Electric Characteristics Using High Voltage—p&d Chesf*, CHESF, Campina Grande, Brazil, 2003.

[20] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System architecture directions for networked sensors," in *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 93–104, 2000.