CS-684-2018 Final Report

Smart Garbage Collection
Team Smart Clean
Team Code - 03

Manas Das(163236001)
Sambit Senapati(163230013)
Emmey Teja(163230003)

Under Guidance of
Prof. Kavi Arya
TA
Saurav Shandilya

Computer Science & Engineering, IIT Bombay
April 29, 2018

Contents

Chapter 1

Introduction

Deploying smart garbage bins for real time waste management system is one of the key applications of a smart city system. Municipal authorities need an efficient way to clear the trash from all public places before it becomes a mess. And this needs to be achieved with the minimum overhead of cost and impact to the city dwellers. LoRaWAN is one of the earlier LPWAN technologies that envisages a city-wide network for keeping track of public infrastructure assets. Once fitted with a BLS(Bin Level Sensing) device on a trash can, LoRaWAN allows the city authorities to keep a tab on the bins via wireless connectivity. LoRaWAN[5] does suffer from some limitations, such as extremely low data rates, higher collision and deteriorated network performance with increasing number of nodes and inability to perform over the air updates to the device. Like every emerging technology, it is also going through an evolutionary phase and is also getting a healthy competition from some of the other emerging standards such as NB-IoT and Weightless. However, at the current state, this technology is well suited for non-mission critical[2] applications, and tracking garbage bins is one such use case.The overall architecture of implementation will be as shown in fig 1.1

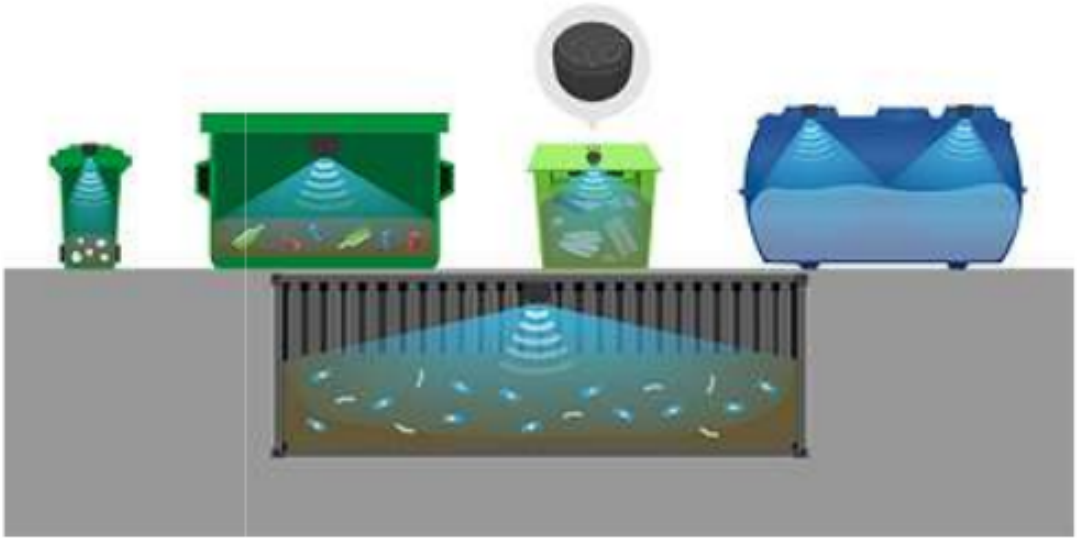

Figure 1.1: Overall Implementation

Chapter 2

Problem Statement

2.1 Goal to achieve

The aim is to implement smart waste management by means of LoRaWAN protocol. LoRaWAN[4] provides long range wireless communication with ultra-low power consumption that enables very long battery life. Our task is to build a smart garbage bin[1] with level indicator to be implemented on Atmega328p i.e. compatible with Arduino IDE as sensor node and a javascript based web application with Google map integrated into it which will indicate location of the bin.This solution is ideally suited for waste pickup and management companies, who can optimize their logistics resources while reducing collection costs. The overall implementation looks as in fig 2.1

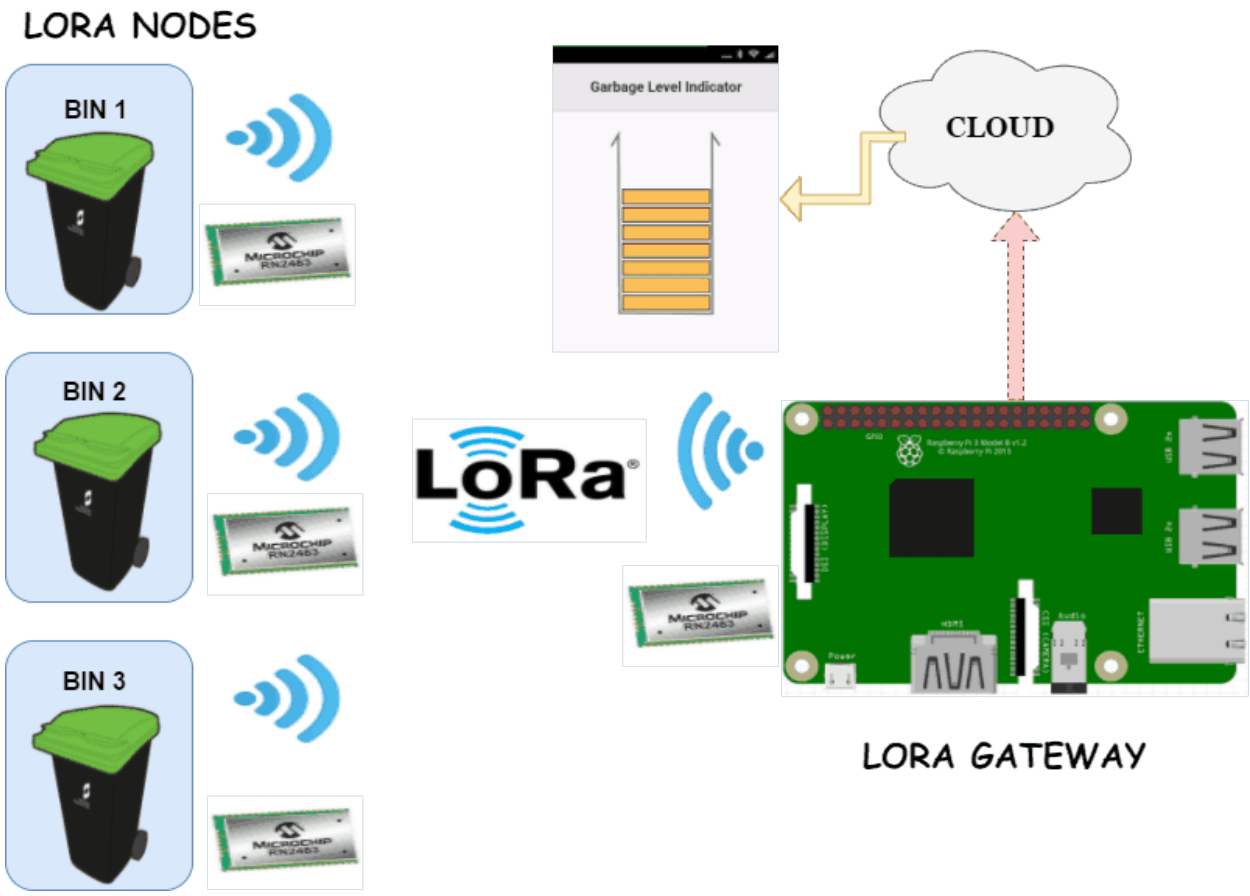

Figure 2.1: Overall Implementation

Chapter 3

Requirements

3.1 Functional Requirements

1. Node – This is the BLS device fitted in the garbage bin. It has an ultrasonic sensor to sense the garbage level inside the bin and a LoRa module (Microchip RN2483) as the communication interface. The node hardware is based on Atmega328p.
2. Ultrasonic sensor to detect the solid wastes in the bin
3. Level indicator detection on the node setup and displaying the same on the dashboard.
4. Gateway – This is used as an external gateway, which aggregates the data from the nodes and pushes it to cloud. The gateway connects to the nodes using LoRa. The gateway is built upon Raspberry Pi/User Laptop and it is also interfaced with RN2483 for receiving the data from nodes over LoRa physical layer.
5. WEB / Mobile UI App – Web app shows the all the bins in the city on google map and current level of the bins live on the dashboard with user authentication.

3.2 Non-functional Requirements

1. Database integration with dashboard as the all the realtime data can be stored on cloud platforms like PubNub.com.

2. Real-time Google map tracker

3.3 Hardware Requirements

1. Compact circuit design of Sensor Node with Atmega328p, Lora Module[3], Ultrasonic sensor, sensor switching circuit and DC to DC booster module.

2. Coming up with a compact PCB design so that all the sensors and communicating modules can be fitted and setup can be used as robust standalone module.

3.4 Software Requirements

1. Developing an Arduino code for proper functioning of sensor node with Lora module with the facility of level indicator for waste level.

2. Python code to initiate lora environment and taking data from sensor node to dashboard.

3. Developing a web application for sensor data visualization and indicating the location of the bin on Google map.

Chapter 4

System Design

4.1 Overall System Architecture

The overall implementation is described as in fig.2.1

4.2 Hardware Architecture

Basic architecture of sensor node is as described in fig. 4.1

Each sensor node is made to be built on a compact PCB with following components mounted on it, such as :

1. Armega328p
2. Ultrasonic sensor
3. Lora Module i.e. RN2483
4. Low-dropout or LDO, booster module and all the necessary components mounted on it

The circuit design looks as in fig 4.2
PCB design of the sensor is as described in fig 4.3
The packaged sensor node looks as shown in fig 4.4

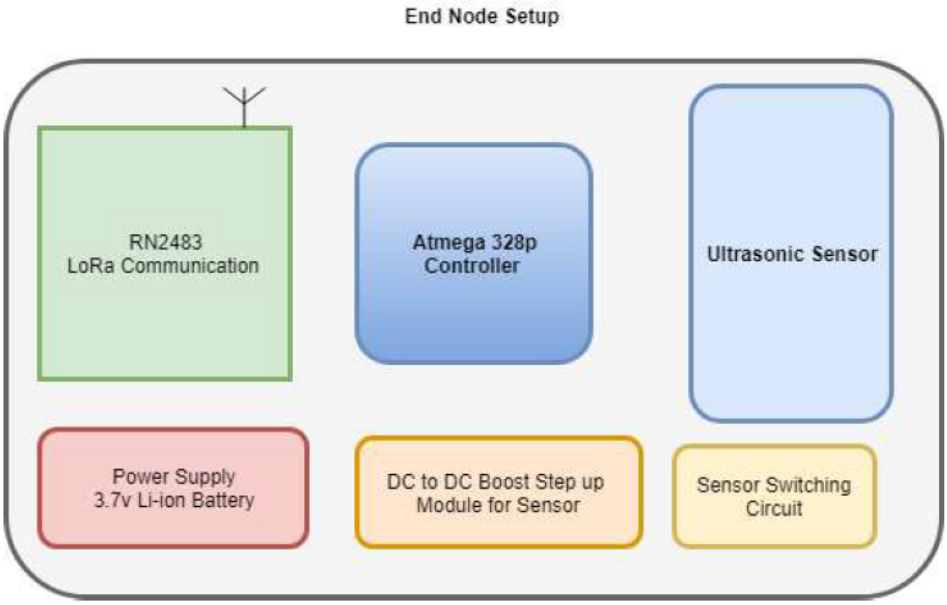

Figure 4.1: Sensor node architecture

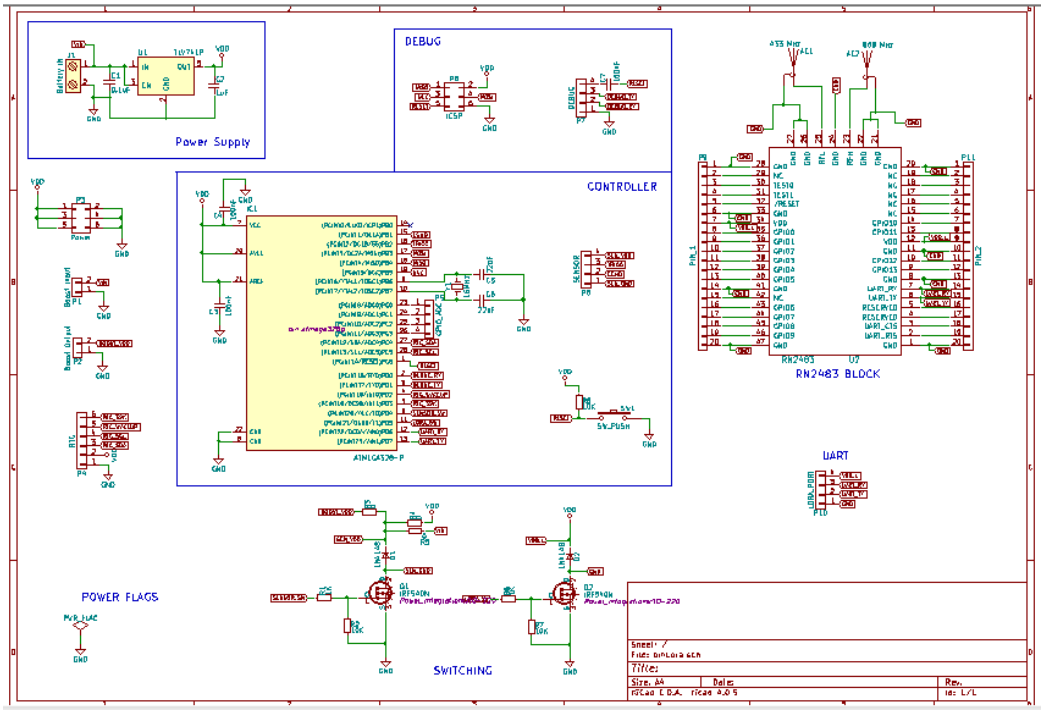

Figure 4.2: Schematic of sensor node

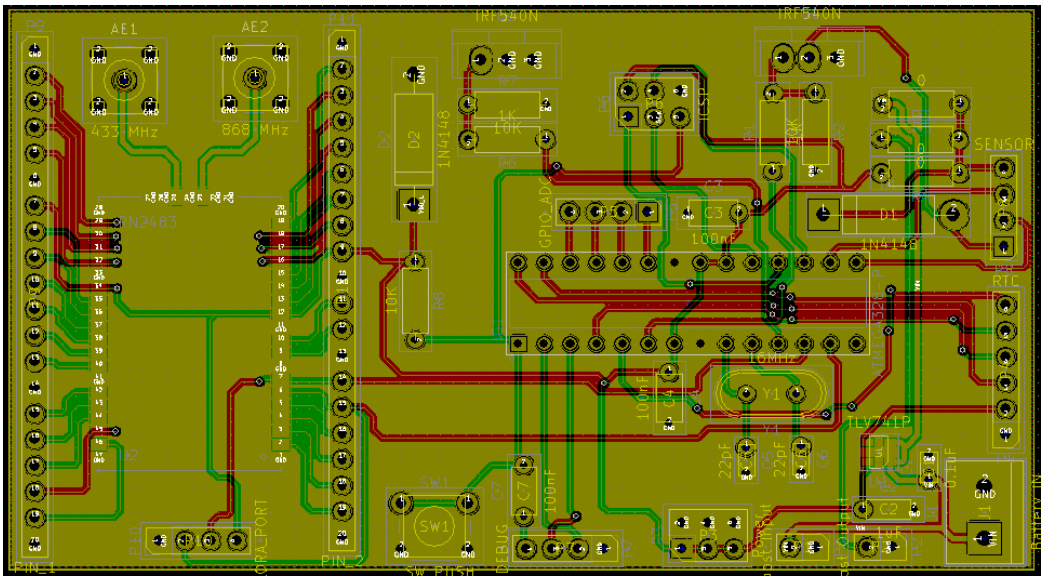

Figure 4.3: PCB design of sensor node

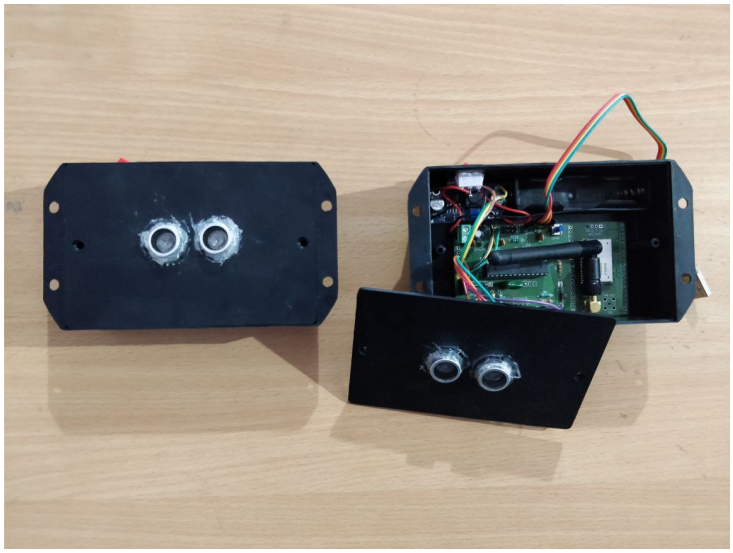

Figure 4.4: Overall node setup

Chapter 5

Working of the system & Test Results

The whole setup was built with two sensor nodes and a gateway.

The gateway on linux machine with lora module attached is as shown in fig 5.1

Sensor node setup as packaged is shown in fig 5.2

Testing setup of two sensor nodes with waste bins are done as shown in fig 5.3

Sensor data from the nodes and its visualization on dashboard is shown as verification in fig 5.4

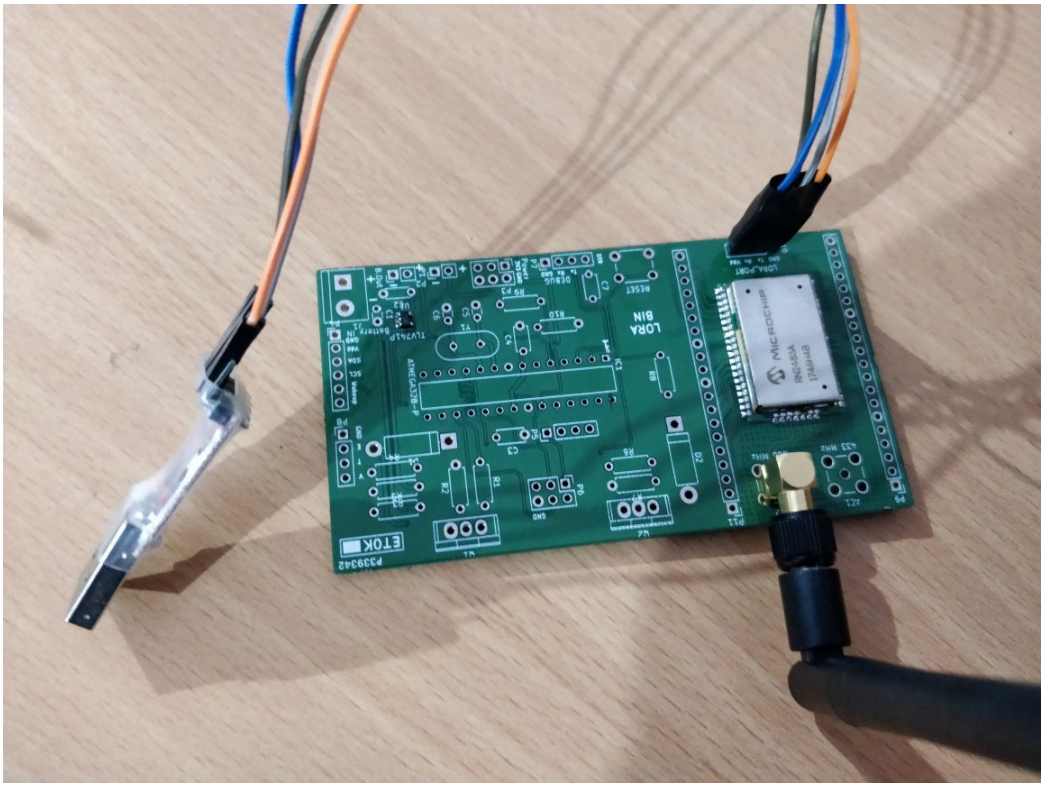

Figure 5.1: Gateway Lora module seup

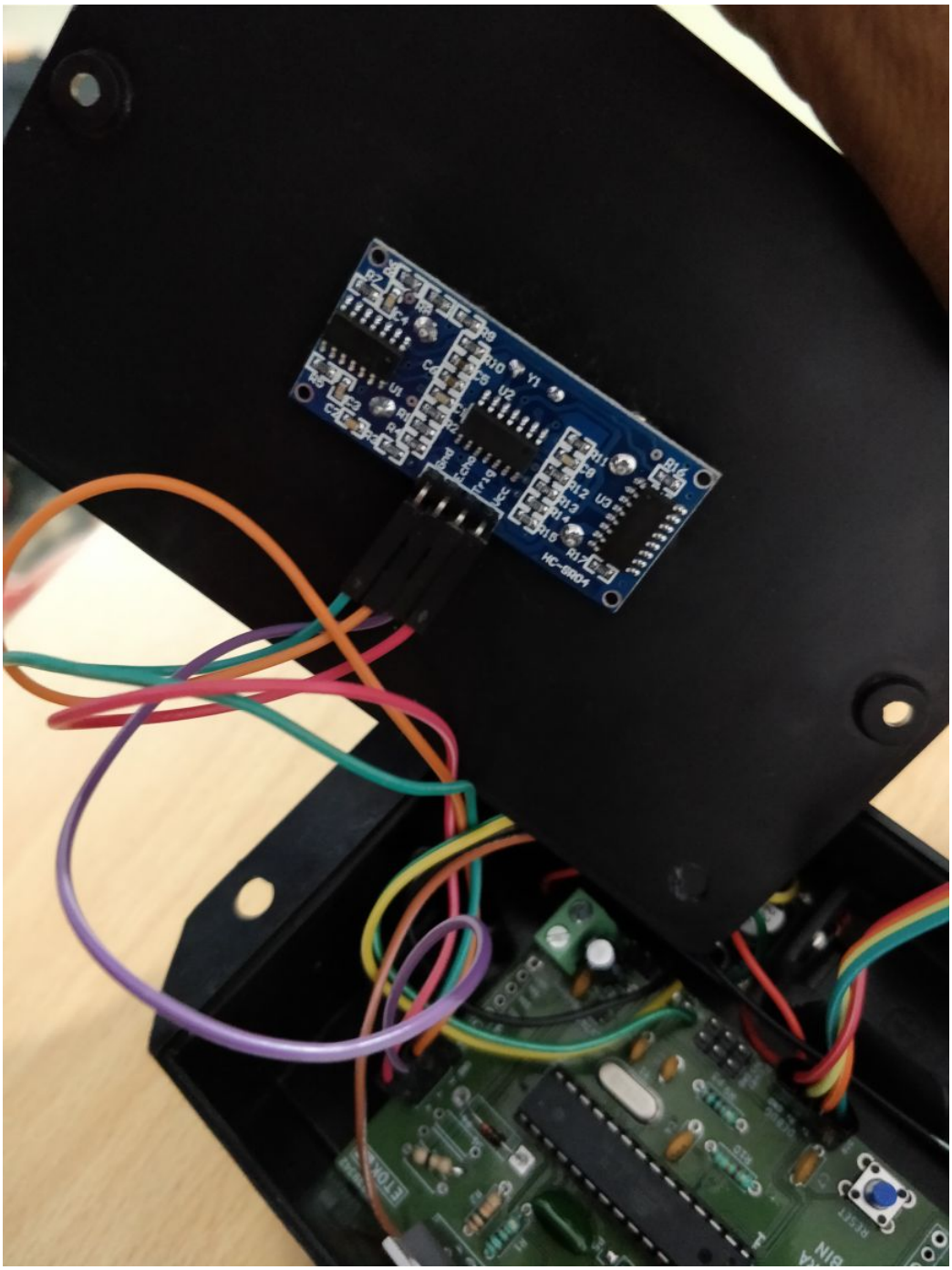

Figure 5.2: Sensor node

Figure 5.3: Realtime Testing of sensor nodes

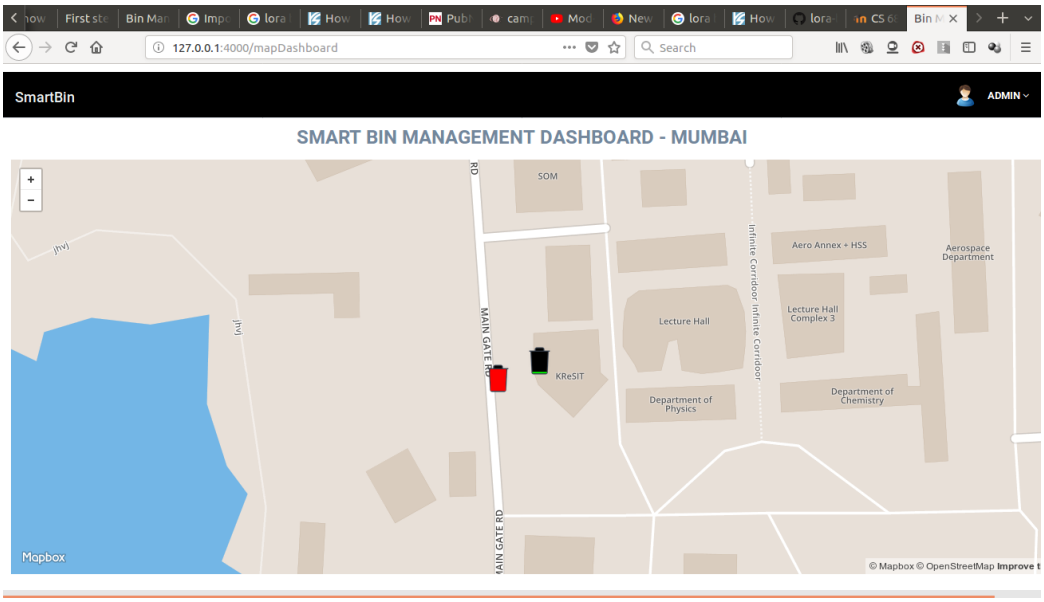

Figure 5.4: Dashboard with Google map & level indicator

Chapter 6

Discussion of the system

- The setup is working accordingly as expected i.e. showing the waste levels and specifying the location on dashboard.But the levels indicated by the ultrasonic sensor is not quite adaptive to different sizes of the bins i.e. bins of varying length.
- The sensor node built is quite compact in form and robust with proper power management facilities.

Chapter 7

Future work

- The major issue with this setup is the ultrasonic sensor isn't adaptive i.e. it doesn't detect the level of waste in the bin properly for bins of different sizes.
- On next stage of implementation, the product can suffice environmental sensors (Capturing Moisture, Bin movement using GPS / Motion Sensors) and sensor to detect the tilt of the BIN Top Door.
- Maps can be improved showing the route of the filled bins.

Chapter 8

Conclusion

The overall system with LoRaWAN protocol for smart garbage collection was implemented successfully and initially the PoC was tested for its working.But as the PoC was bulky so came up with a much robust, compact model which is more portable and easy to install in variuos environment conditions.And the whole system was tested with two nodes fitted to different waste bins and dashboard hosted on linux machine.The whole of the setup is worked pretty well and the previous data instances being stored on PubNub.com cloud pltaform.

Appendices

A Python code of lora environment initialization

Python Code .1 `mclora.py`

```
1 #LoRa Serial Handle - Send Commands and Handle the
     response
2 #Import the Modules Required
3 import serial
4
5 '''
      **********************************************************************

6 Class Name            :    MCLoRa
7 Description           :    Handle LoRa Events
```

```
8  ********************************************************************
       '''
9  class MCLoRa:
10     def __init__(self, port):
11         """Conctructor - needs serial port string."""
12         self.ser = serial.Serial(port, 57600)
13
14     '''
           ************************************************************

15     Function Name        :   testOK
16     Description          :   Check module is working
17     Parameters           :   none
18     ************************************************************
         '''
19     def testOK(self):
20         """Tests communication with Microchip Lora
              Module."""
21         # send:
22         # sys get ver
23         # expect:
24         # RN2483 0.9.5 Mar 24 2015 14:15:33
25         try:
26             self.ser.write("sys reset\r\n".encode())
27             s = self.ser.readline().decode().split()
28             if s[0] == 'RN2483':
29                 return (s[0], s[1], " ".join(s[2:]))
30             else:
31                 return False
32         except Exception as error:
33             print error
34             self.ser.write("sys get ver\r\n".encode())
35             s = self.ser.readline().decode().split()
36             if s[0] == 'RN2483':
37                 return (s[0], s[1], " ".join(s[2:]))
38             else:
39                 return False
40
```

```
41      '''
           ************************************************************

42      Function Name        :   pause
43      Description          :   Pause MAC Operation and
           continue with radio
44      Parameters           :   none
45      ************************************************************
           '''
46      def pause(self):
47          """Pauses LoRaWAN stack."""
48          self.ser.write('mac resume\r\n'.encode())
49          val = self.ser.readline().decode()
50          self.ser.write('mac pause\r\n'.encode())
51          val = self.ser.readline().decode()
52          return val
53      '''
           ************************************************************

54      Function Name        :   recv
55      Description          :   Receive Data over LoRa
56      Parameters           :   none
57      ************************************************************
           '''
58      def recv(self):
59          """Waits for data. This call will block.
60          """
61          # start receive - will block
62          self.ser.write('radio rx 0\r\n'.encode())
63          # get response
64          val = self.ser.readline().decode().strip()
65          data = None
66          if val == 'ok':
67              data = self.ser.readline().split()
68              print(data)
69          # expected:
70          # radio_rx <data>
71          if data[0] == 'radio_rx':
```

```
72              data = data[1]
73          return data
74
75      '''
            ************************************************************
76      Function Name       :   getUniqueID
77      Description         :   Obtain the Unique ID
78      Parameters          :   none
79      ************************************************************
            '''
80      def getUniqueID(self):
81          """Get globally unique number provided by
                Microchip.
            """
82
83          # example:
84          # sys get hweui
85          # 0004A30B001AF09E
86          self.ser.write('sys get hweui\r\n'.encode())
87          id = self.ser.readline().decode().strip()
88          return id
89
90      '''
            ************************************************************
91      Function Name       :   send
92      Description         :   Send the data over LoRa
93      Parameters          :   none
94      ************************************************************
            '''
95      def send(self):
96          """Waits for data. This call will block.
97          """
98          # start receive - will block
99          self.ser.write('radio tx 01\r\n'.encode())
100         # get response
101         val = self.ser.readline().decode().strip()
102         print val
```

```
103             if val == 'ok':
104                 data = self.ser.readline().split()
105                 print(data)
106             if data[0] == 'radio_tx_ok':
107                 data = data[0]
108             return data
109 
110 #End of the Script
111 ##*************************************************************
```

B Python code for data receive on gateway

Python Code .2 `binServer.py`

```
1  #Import the Modules Required
2  import sys
3  import datetime
4  import pytz
5  import json
6  import argparse
7  from mclora import MCLoRa
8  from threading import Thread
9  from pubnub.callbacks import SubscribeCallback
10 from pubnub.enums import PNStatusCategory
11 from pubnub.pnconfiguration import PNConfiguration
12 from pubnub.pubnub import PubNub
13 from numpy import interp, clip
14 import logging
15 
16 logging.basicConfig(level=logging.INFO)
17 
18 TIME_ZONE = "Asia/Kolkata"
19 
20 binMapping = {
21     "25046B" : 1,
22     "25046A" : 2
23 }
```

```
24
25 pnconfig = PNConfiguration()
26
27 pnconfig.subscribe_key = 'sub-c-62fed8bc-2d10-11e8-
   a27a-a2b5bab5b996'
28 pnconfig.publish_key = 'pub-c-002fbba1-6de1-410b-8c9e-
   ff75720aaa49'
29
30 pubnub = PubNub(pnconfig)
31
32 #System Variables
33 port = " "
34 loraM = " "
35
36 def my_publish_callback(envelope, status):
37     # Check whether request successfully completed or
       not
38     if not status.is_error():
39         print "Successfully Sent"
40         # Message successfully published to specified
           channel.
41     else:
42         pass  # Handle message publish error. Check '
           category' property to find out possible
           issue
43         # because of which request did fail.
44         # Request can be resent using: [status retry];
45
46 class MySubscribeCallback(SubscribeCallback):
47     def presence(self, pubnub, presence):
48         pass  # handle incoming presence data
49
50     def status(self, pubnub, status):
51         if status.category == PNStatusCategory.
           PNUnexpectedDisconnectCategory:
52             pass  # This event happens when radio /
               connectivity is lost
53
```

```
54          elif status.category == PNStatusCategory.
               PNConnectedCategory:
55              pass
56              # Connect event. You can do stuff like
                   publish, and know you'll get it.
57              # Or just use the connected event to
                   confirm you are subscribed for
58              # UI / internal notifications, etc
59              # pubnub.publish().channel("awesomeChannel
                   ").message("hello!!").async(
                   my_publish_callback)
60          elif status.category == PNStatusCategory.
               PNReconnectedCategory:
61              pass
62              # Happens as part of our regular operation
                   . This event happens when
63              # radio / connectivity is lost, then
                   regained.
64          elif status.category == PNStatusCategory.
               PNDecryptionErrorCategory:
65              pass
66              # Handle message decryption error.
                   Probably client configured to
67              # encrypt messages and on live data feed
                   it received plain text.
68
69      def message(self, pubnub, message):
70          print message
71
72
73  '''
      **********************************************************************

74  Function Name        :    obtain_port
75  Description          :    Obtain the Serial Port from
       argument
76  Parameters           :    none
77  **********************************************************************
```

```
    '''
78  def obtain_port():
79      global port
80      #obtain the Port from the Argument
81      descStr = "Traffic Controller: Enter the LoRa Port
            "
82      parser = argparse.ArgumentParser(description=
            descStr)
83      parser.add_argument('--port', dest='serial_port',
            required=True)
84      args = parser.parse_args()
85      if args.serial_port:
86          port = args.serial_port
87
88  def scale(value, src_min, src_max, dst_min, dst_max,
        round_=False):
89      """
90      Scale a value from one range to another.
91
92      :param value: Input value
93      :param src_min: Min value of input range
94      :param src_max: Max value of input range
95      :param dst_min: Min value of output range
96      :param dst_max: Max value of output range
97      :param round_: True if the scale value should be
            rounded to an integer
98
99      :return: The scaled value
100     """
101     scaled = interp(clip(value, src_min, src_max), [
            src_min, src_max], [dst_min, dst_max])
102     if round_:
103         scaled = int(round(scaled))
104
105     return scaled
106
107 '''
    ************************************************************
```

```
108 Function Name        :   loraReceive
109 Description          :   Receives the LoRa Data
110 Parameters           :   none
111 ************************************************************************
    '''
112 def loraReceive():
113     global loraM, pubnub
114     count  = 0
115     while True:
116         print "LoRa Packet Receive Start"
117         try:
118             loraData = str(loraM.recv())
119             if loraData != "['radio_err']":
120                 loraList = loraData.split("2C")
121                 distanceReceived = scale(int(loraList
                        [1],16), 0, 250, 100, 0)
122                 loraPacket = {
123                     "binId" : binMapping[loraList[0]],
124                     "binData" : {
125                         "fillLevel" : int(
                                distanceReceived),
126                         "batteryLevel" : int(loraList
                                [2],16),
127                         "timeStamp": str(datetime.
                                datetime.now(pytz.timezone(
                                TIME_ZONE)).strftime('%m-%d
                                 %H:%M'))
128                     }
129                 }
130                 print loraPacket
131                 print pubnub.publish().channel("
                        binData").message(loraPacket).async
                        (my_publish_callback)
132                 # 25046B2C00B42C62
133             else:
134                 print "No Data Received"
135
```

```
136             except Exception as error:
137                 print error
138 
139 '''
    ***********************************************************************

140 Function Name        :   systemInit
141 Description          :   Initiazie the LoRa and Twilio
    Client
142 Parameters           :   None
143 ***********************************************************************
    '''
144 def systemInit():
145     global port, loraM, pubnub
146     obtain_port()
147 
148     #loraM handles all the loraEvents
149     loraM = MCLoRa(port)
150     success = loraM.testOK()
151     if success:
152         print "Bin Management Init Success"
153         print (success)
154     else:
155         print("Bin Management Init Failure")
156     loraM.pause()
157 
158     pubnub.add_listener(MySubscribeCallback())
159     pubnub.subscribe().channels('binData').execute()
160 
161 if __name__ == "__main__":
162     systemInit()
163     loraThread = Thread(target = loraReceive)
164     loraThread.setDaemon(True)
165     loraThread.start()
166 
167     print("Bin Management Started ...\n")
168 
169     while True:
```

```
170         try:
171             pass
172         except KeyboardInterrupt:
173             sys.exit(0)
174
175
176 #End of the Script
177 ##*************************************************************************
```

C Arduino code for Lora communication & Waste detection

Arduino Code .1 `smartBin_withSleep.ino`

```
1  // **** INCLUDES *****
2  #include "LowPower.h"
3  #include <SoftwareSerial.h>
4
5  //LoRa Serial
6  SoftwareSerial Serial1(7, 6); // RX, TX
7
8  const char deviceID[10] = "25046A";
9  #define Seconds 30
10 #define LoRa_VDD 15
11 #define Sensor_VDD 4
12
13 // sensor pins numbers
14 const int trigPin = 10;
15 const int echoPin = 9;
16
17 // sensor variables
18 uint16_t distance_to_send = 0;
19
20 //batteryLevel
21 int batteryLevel = 0;
22
23 void systemInit() {
24   //Sensor Init
```

```
25    pinMode(Sensor_VDD, OUTPUT);
26    pinMode(trigPin, OUTPUT); // Sets the trigPin as an
          Output
27    pinMode(echoPin, INPUT); // Sets the echoPin as an
          Input
28
29    //LoRa Init
30    pinMode(LoRa_VDD, OUTPUT);
31    Serial1.begin(57600);
32
33  }
34  void setup()
35  {
36    // No setup is required for this library
37    Serial.begin(9600);
38
39    systemInit();
40    loraReset();
41    //Calibation of the distance
42    distance_to_send = distance_calibrated();
43  }
44
45  int sleepCount = 0;
46
47  void loop()
48  {
49    sleepCount++;
50    if (sleepCount >= (Seconds / 8)) {
51      sleepCount = 0;
52      distance_to_send = distance_calibrated();
53      lora_send();
54    }
55    // Enter power down state for 8 s with ADC and BOD
          module disabled
56    LowPower.powerDown(SLEEP_8S, ADC_OFF, BOD_OFF);
57  }
58
59  void loraReset()
```

```
60 {
61   digitalWrite(LoRa_VDD, LOW);
62   delay(250);
63   digitalWrite(LoRa_VDD, HIGH);
64   delay(250);
65 }
66
67 uint8_t obtain_distance() {
68   long duration = 0;
69   int distance = 0;
70   // Clears the trigPin
71   digitalWrite(Sensor_VDD, HIGH);
72   digitalWrite(trigPin, LOW);
73   delayMicroseconds(2);
74
75   // Sets the trigPin on HIGH state for 10 micro
       seconds
76   digitalWrite(trigPin, HIGH);
77   delayMicroseconds(10);
78   digitalWrite(trigPin, LOW);
79
80   // Reads the echoPin, returns the sound wave travel
       time in microseconds
81   duration = pulseIn(echoPin, HIGH);
82
83   // Calculating the distance
84   distance = duration * 0.034 / 2;
85
86   // Prints the distance on the Serial Monitor
87   //   Serial.print("Distance: ");
88   //   Serial.println(distance);
89   return distance;
90 }
91
92 uint8_t distance_calibrated() {
93   int i = 0;
94   int newDistance = 0;
95   int oldDistance = 0;
```

```
96    int oldFlag = 0;
97    int newFlag = 0;
98    for (i = 0; i <= 20 ; i++) {
99      newDistance = obtain_distance();
100     if ((oldDistance - 5) > newDistance > (oldDistance
          + 5)) {
101       Serial.println("Old Distance");
102       oldFlag++;
103       newFlag = 0;
104       if (oldFlag >= 2) {
105         oldDistance = newDistance;
106       }
107     }
108     else if ((oldDistance - 5) < newDistance < (
          oldDistance + 5)) {
109       newFlag++;
110       oldFlag = 0;
111       if (newFlag >= 3) {
112         Serial.println(" ");
113         Serial.print("Parsed Distance: ");
114         Serial.println(newDistance);
115         Serial.print("Battery Level: ");
116         batteryLevel = map(analogRead(A0), 0, 925, 0,
              100);
117         Serial.println(batteryLevel);
118         Serial.println(" ");
119         delay(250);
120         return newDistance;
121       }
122     }
123   }
124 }
125
126 void lora_send() {
127   char loraBuffer[100];
128   loraReset();
129   Serial1.println("mac pause");
130   while (!Serial1.available() ) {
```

```
131       delay(100);
132     }
133     while (Serial1.available())
134       Serial.write(Serial1.read());
135
136     delay(500);
137
138     sprintf(loraBuffer, "radio tx %s2C%04X2C%02X",
            deviceID, distance_to_send, batteryLevel);
139     Serial1.println(loraBuffer);
140     while (!Serial1.available() ) {
141       delay(100);
142     }
143     while (Serial1.available())
144       Serial.write(Serial1.read());
145
146     delay(3000);
147   }
```

Bibliography

[1] Abhay Shankar Bharadwaj; Rainer Rego; Anirban Chowdhury. *IoT based solid waste management system: A conceptual approach with an architectural solution as a smart city application*. IEEE Annual India Conference (INDICON), 2016.

[2] Mehrdad Babazadeh; Sokratis Kartakis; Julie A McCann. *Highly-distributed sensor processing using IoT for critical infrastructure monitoring* . Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC), 2017.

[3] Mirochip. *http://www.microchip.com/wwwproducts/en/RN2483*. 2016.

[4] Alexandru Lavric; Valentin Popa. *A LoRaWAN: Long range wide area networks study*. International Conference on Electromechanical and Power Systems (SIELMEN), 2017.

[5] Steffen Thielemans; Maite Bezunartea; Kris Steenhaut. *Establishing transparent IPv6 communication on LoRa based low power wide area networks (LPWANS)*. Wireless Telecommunications Symposium (WTS), 2017.