

AWS IoT Core and LoRaWAN Based Sensory Network system for Military Search and Rescue Missions

Lieutenant Colonel Ramendra Mathur, Dr.Ashok Kumar Tiwari, Prashant Dutta

Abstract

The Indian armed forces is the second largest Military force in the world comprising of 1.4 million+ personnel's (Active force) and 1.1 million+ reserved personnel's. Apart from their duty of safeguarding its borders, the Military personnel's are also engaged in Search and Rescue operations (SAR) across the globe. The SAR is conducted both for Military personnel's and Civilians. The search and rescue operations are carried on varied demographic regions ranging from Snow areas (-50° C) to Deserts ($+55^{\circ}$ C), also SAR is conducted during natural calamities like Earthquake, Floods, Tsunami, Avalanche, Landslides etc. But the main challenge for any Military personnel in SAR is to identify whom to rescue first. By saying this we mean that if the personnel has the knowledge about the Vitals of the individual he/she is rescuing then he/she will be able to assess the gravity of the situation through those vitals. Vitals like SP02, Blood Pressure, Pulse rate can predict the survivability factor of the individual. So it means that if out of three individuals the Vitals of 1st person are serious and Vitals of the 2nd Person are Normal and the Vitals of the 3rd Person are Null, then it can be predicted that the 1st person should be rescued first and the 2nd person be rescued Second and the 3rd person whose Vitals are Null can be presumed dead and shall be rescued at the last. Now the main challenge comes, and that is how to gather those Vitals. The answer to this problem has been suggested through the experiments conducted in this paper. This paper suggests for a wireless sensor wrist band supporting LoRaWAN protocol (Long Range Wide Area Network). The LoRaWAN uses LoRa Radio to transmit in low power at long-range wide area network. The aforementioned wristband shall transmit data to the AWS Cloud using AWS IoT Core. The AWS Cloud will aggregate the data received from all the people wearing the wrist band. From the aforesaid data those persons can be easily identified whose Vitals are critical. This will prove to be a Life Saver for Armed forces.

Introduction

LoRaWAN

LoRaWAN is placed in the physical 'LoRa layer' and has a raw-max datarate of 27kbps. Given the low power long range capabilities of LoRa WAN wireless sensors, a single-sensor or 'network of wireless IoT sensors' can remain ON for about 10 years before they need any overhaul.

Some Characteristics of LoRaWAN are :-

- 1) Bidirectional
- 2) Long battery-life (Ten year)
- 3) Low Price
- 4) Low data-rate ('0.27bps – 50kbps')
- 5) Long range (at Line of Sight – 70 Kms)
- 6) Works in 'unlicensed spectrum'

7)Secure (AES-encryption)

How the LoRaWAN works

Devices<->LoRa radio <->Gateway<->Network Server <->End Application

Upstream messages (data sent from the device) are encrypted and sent over LoRa radio through transmissions. The message is received by one or more gateways, who then transfer the encrypted data over a network (typically IP cellular/Ethernet) to a network server. The network server is the software that authenticates the device and decrypts the LoRaWAN payload. The server then delivers the data packet to the appropriate end application.

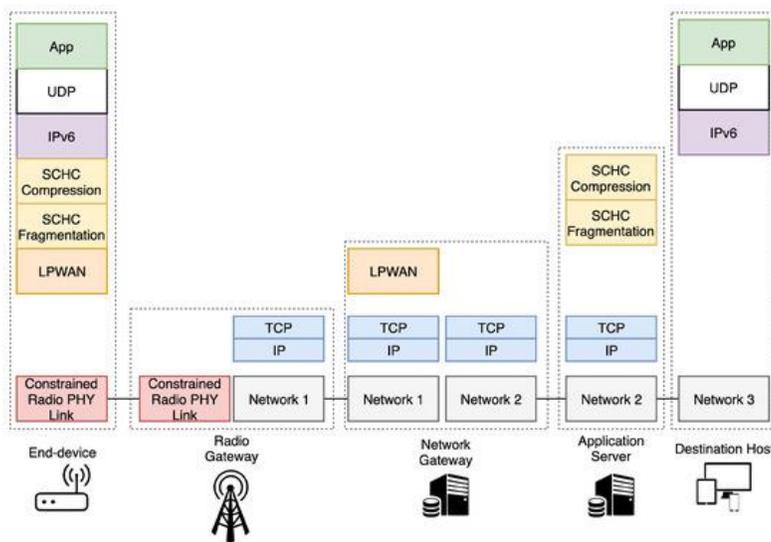


fig 1.1 Flow Diagram

LORAWAN WIRELESS SENSOR DEVICES

A LoRaWAN sensor device is any device that receives or transmits informational-data and is mainly done by out by devices like wireless-sensors, detectors, or actuators. These dvices are categorized in 3 categories :

Class A – These devices only receives data for small interval after transmitting data.

Class B- These devices receives data in previously set intervals.

Class C- These devices can constantly receive data, but it uses high power.

LORA RADIO

LoRa uses radio frequency (RF) signals to transmit and receive in the “unlicensed ISM spectrum”. By using this anyone can easily use this band without getting charged or taking a license

LORA GATEWAY

A gateway is an access point or modem(modulator/demodulator). The Gateway receives all LoRa radio data sent by sensors within range. If the gateway has a network-server built in it will process the data payload locally, and if the network-server is located in the ‘cloud’ the gateway will simply relay the encrypted packet to the server.

Use of Network Server

As soon as the LoRa-gateway has obtained a transmitted data-packet(message), it will then continue up-stream to a network-server. The network servers are the most smart devices of LoRaWAN. Network servers perform the following tasks:

- 1)Decrypt LoRa payload-data
- 2)Enables the sensors to connect to the network.
- 3)Grouping all messages from all LoRaWAN-gateways within its network and sensory range.
- 4)Direct/advance incoming-data to the destined applications.
- 5)Regulate LoRa-radio settings to the gateways.
- 6)Assess/monitor gateways and devices.

END APPLICATION

The data from the sensors via LoRaWAN and LoRa Gateway and through Cloud finally reaches the End Application where it is processed and insights are drawn.



fig 1.2 LoRaWAN Stack Architecture

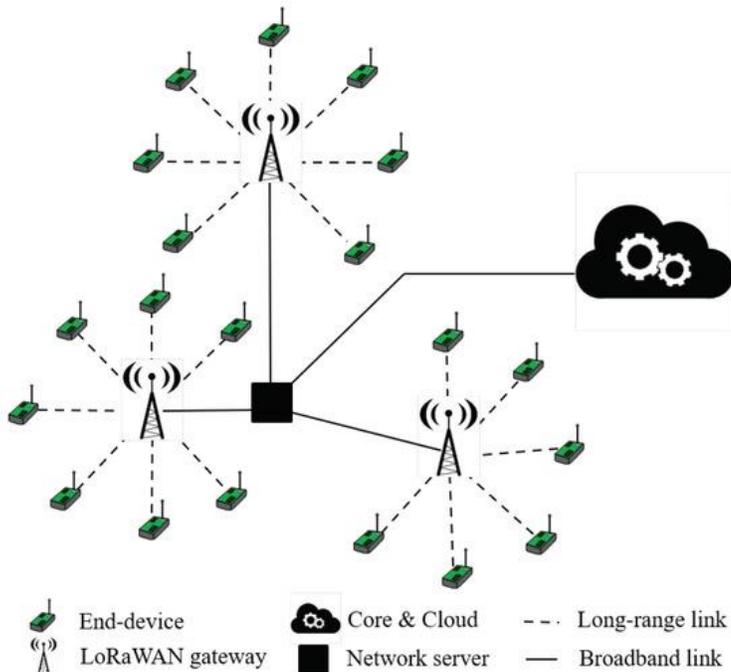


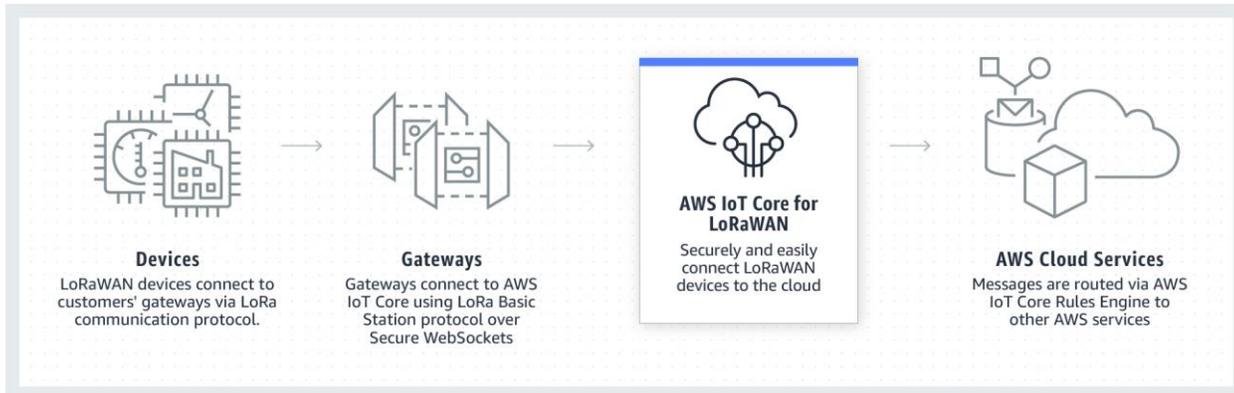
fig 1.3 LoRa Architecture

AWS IoT Core

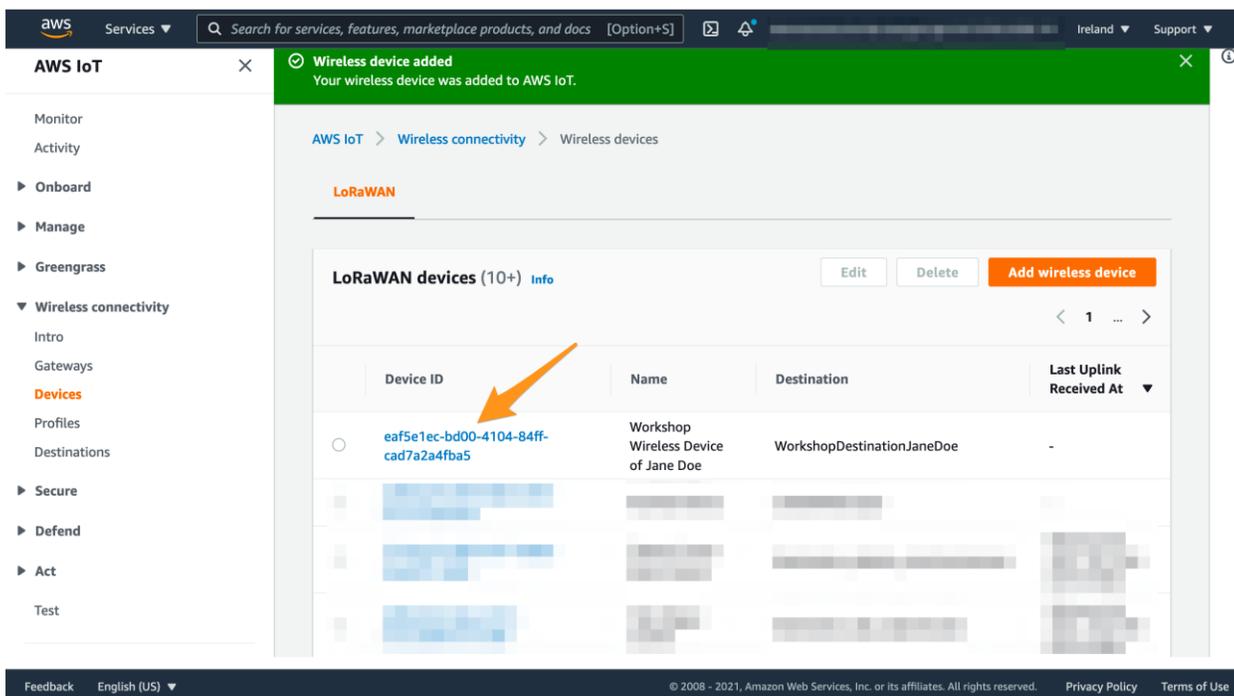
AWS IoT Core is a managed cloud service that lets connected devices easily and securely interact with cloud applications and other devices. AWS IoT Core can support billions of devices and trillions of messages, and can process and route those messages to AWS endpoints and to other devices reliably and securely. With AWS IoT Core, your applications can keep track of and communicate with all your devices, all the time, even when they aren't connected.

AWS IoT Core makes it easy to use AWS services such as AWS Lambda, Amazon Kinesis, Amazon S3, Amazon SageMaker, Amazon DynamoDB, Amazon CloudWatch, AWS CloudTrail, and Amazon QuickSight to build Internet of IoT applications that gather, process, analyze and act on data generated by connected devices, without having to manage any infrastructure.

The following figure demonstrates the parts of a LoRaWAN based solution using AWS-IoT-Core :



AWS IOT Core Dashboard to connect devices



The screenshot shows the AWS IoT Core console interface. A green notification banner at the top states "Wireless device added" with the subtext "Your wireless device was added to AWS IoT." Below this, the breadcrumb navigation is "AWS IoT > Wireless connectivity > Wireless devices". The "LoRaWAN" section is active, showing a table of "LoRaWAN devices (10+)". An orange arrow points to the first device entry in the table.

Device ID	Name	Destination	Last Uplink Received At
eaf5e1ec-bd00-4104-84ff-cad7a2a4fba5	Workshop Wireless Device of Jane Doe	WorkshopDestinationJaneDoe	-
[blurred]	[blurred]	[blurred]	[blurred]
[blurred]	[blurred]	[blurred]	[blurred]
[blurred]	[blurred]	[blurred]	[blurred]

Verify device connectivity

a. Trigger your device to send telemetry

Please consult your device's user manual to learn how initiate a sending of a message from your LoRaWAN device.

b. Open AWS IoT Core for LoRaWAN Management console

Please log in using the link <https://console.aws.amazon.com/iotwireless> and ensure that you choose the same region you previously used.

c. Select device

Click on "Devices" in the "Wireless connectivity" section and click on the link at left the left of the name of the device you created in the previous step

d. Verify last uplink received

Ensure that the field "Last uplink received" is populated as illustrated below:

Verify data ingestion

In this step we will use an MQTT Test client in the AWS management console to simulate an invocation of AWS IoT Rule, and see the outcome of binary transformation published to the MQTT topic `lorawanworkshop/transformed`:

a. Open MQTT Test Client

Please use this link or navigate to AWS IoT, Test and MQTT Client.

b. Subscribe to topic

Please input the topic name `dt/workshop_lorawanmessages` in "Subscription topic" field and click on "Subscribe to topic". Please ensure to use exactly the same topic name you created in "Create IoT Rule".

c. Trigger your device to send telemetry

Please consult your device's user manual to learn how initiate a sending of a message from your LoRaWAN device.

d. Review the incoming message from your LoRaWAN device

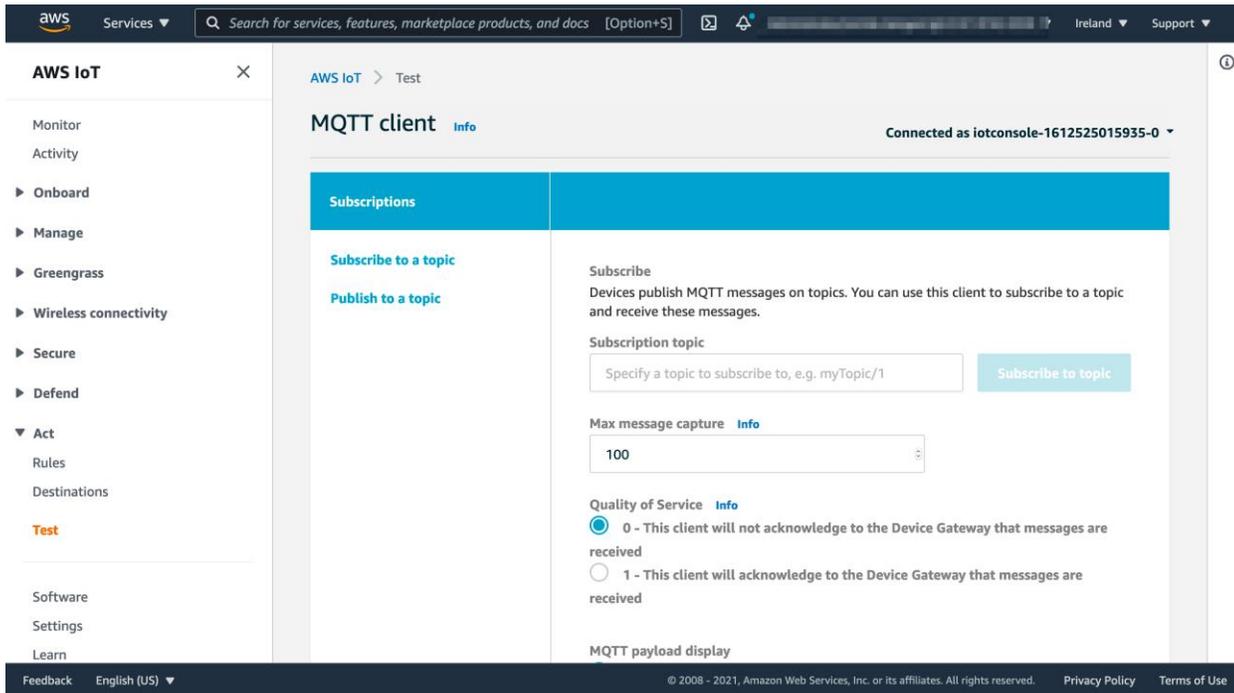
Please find below an explanation for some of the attributes:

PayloadData: Base64-encoded payload from the device

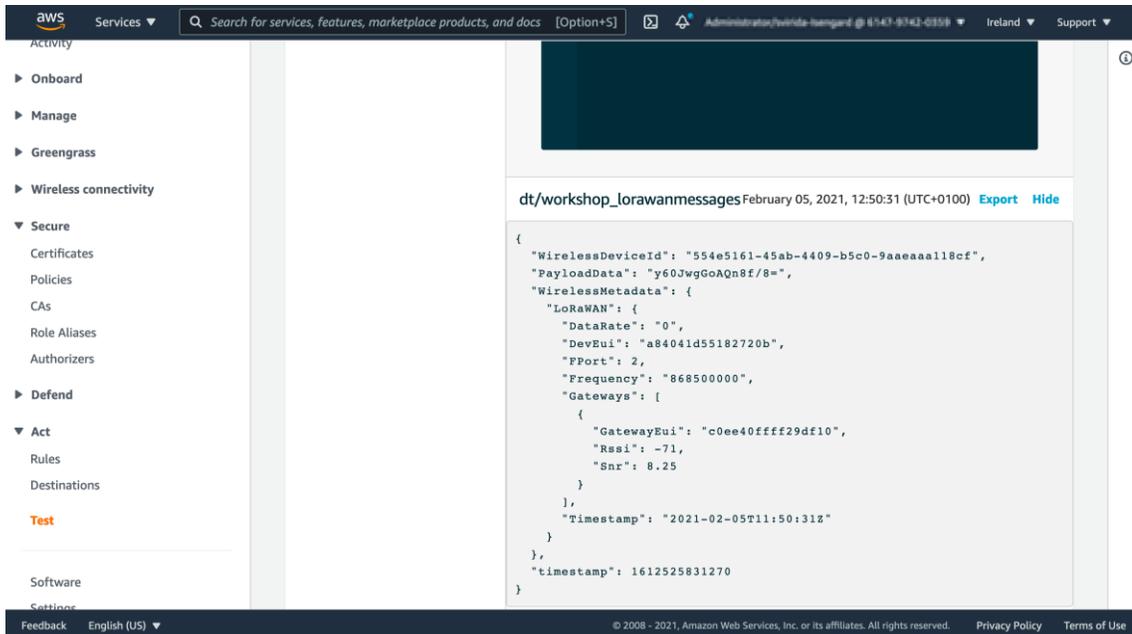
WirelessMetadata.FPort: FPort used by the device

WirelessMetadata.LoRaWAN.DevEui: EUI of the device sending the data

WirelessMetadata.LoRaWAN.Gateways: Information on RSSI and SNR per gateway



The screenshot shows the AWS IoT console interface. On the left is a navigation menu with options like Monitor, Activity, Onboard, Manage, Greengrass, Wireless connectivity, Secure, Defend, Act, Rules, Destinations, Test, Software, Settings, and Learn. The main content area is titled 'MQTT client' and shows configuration options for a client. It includes a 'Subscriptions' section with 'Subscribe to a topic' and 'Publish to a topic' buttons. Below that is a 'Subscription topic' input field with a 'Subscribe to topic' button. There is also a 'Max message capture' dropdown set to 100. The 'Quality of Service' section has two radio buttons: '0 - This client will not acknowledge to the Device Gateway that messages are received' (selected) and '1 - This client will acknowledge to the Device Gateway that messages are received'. At the bottom, there is a footer with 'Feedback', 'English (US)', and copyright information.



The screenshot shows the AWS IoT console displaying a received MQTT message. The message is titled 'dt/workshop_lorawanmessages' and is dated 'February 05, 2021, 12:50:31 (UTC+0100)'. The message payload is shown in a code block with the following JSON structure:

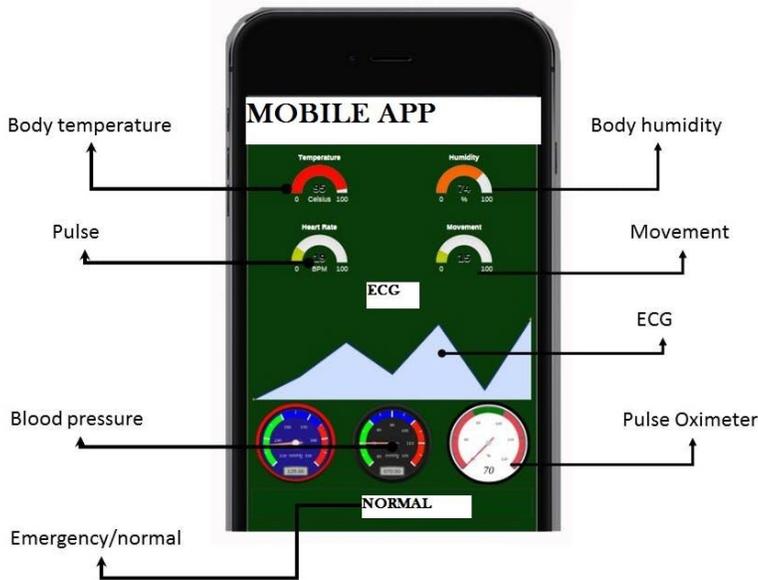
```
{
  "WirelessDeviceId": "554e5161-45ab-4409-b5c0-9aaeaa118cf",
  "PayloadData": "y60JwgGoAQn8f/8=",
  "WirelessMetadata": {
    "LoRaWAN": {
      "DataRate": "0",
      "DevEui": "a84041d55182720b",
      "FPort": 2,
      "Frequency": "868500000",
      "Gateways": [
        {
          "GatewayEui": "c0ee40ffff29df10",
          "Rssi": -71,
          "Snr": 8.25
        }
      ]
    },
    "Timestamp": "2021-02-05T11:50:31Z"
  }
},
"timestamp": 1612525831270
}
```

The footer of the console shows 'Feedback', 'English (US)', and copyright information.

The above steps will now configure the private-LoRaWAN also connects to the Gateway and to the wireless-sensor-device.

Results and Discussions

We have used a Lora development board as a main-controller of this Rescue-system. The Controller board gathers the data of the vitals of the persons through the wrist band and then transmitsto the LoRa Gateway which is further connected to the AWS Cloud through AWS. The mobile application helps in visualizing the sensory data about the patient's vitals and his health parameters. This application can be installed in the doctors mobile phone and through this application he can monitor the patient remotely.



Softwares Used

- a. AWS IoT Core
- b. Android Studio
- c. Arduino Web Editor
- d. Microsoft Visual Studio Code Extension for Arduino

Source Code

```

#include <SPI.h>
#include <MAX30100_PulseOximeter.h>
#include < Sensor.h>
#include < BME280.h>

// Set your AppEUI and AppKey
const char *appEui = "0000000000000000";
const char *appKey = "00000000000000000000000000000000";
#define loraSerial Serial1
#define debugSerial Serial
// Replace REPLACE_ME with TTN_FP_EU868 or TTN_FP_US915
#define freqPlan REPLACE_ME
//VARIABLE TO HOLD THE SENSORS DATA
int bpm;
int spo2;
float temp;
//the sea level pressure in your region (****)
BME280 bme; // BME280 Sensor declaration
unsigned long currentMillis; //hold the current time
//pulse oximeter time period (measurement time period)
#define REPORTING_PERIOD_MS 1000
PulseOximeter pox;
uint32_t tsLastReport = 0;

// Callback (registered below) fired when a pulse is detected
void onBeatDetected()
{
  // Serial.println("Beat!");
}

void measured_pulse(){
  pox.update();
  if (currentMillis - tsLastReport > REPORTING_PERIOD_MS) {
    bpm=pox.getHeartRate();
    tsLastReport = currentMillis;
  }
  ttn[loraSerial, debugSerial, freqPlan];
void setup()
{
  loraSerial.begin(57600);
  debugSerial.begin(9600);

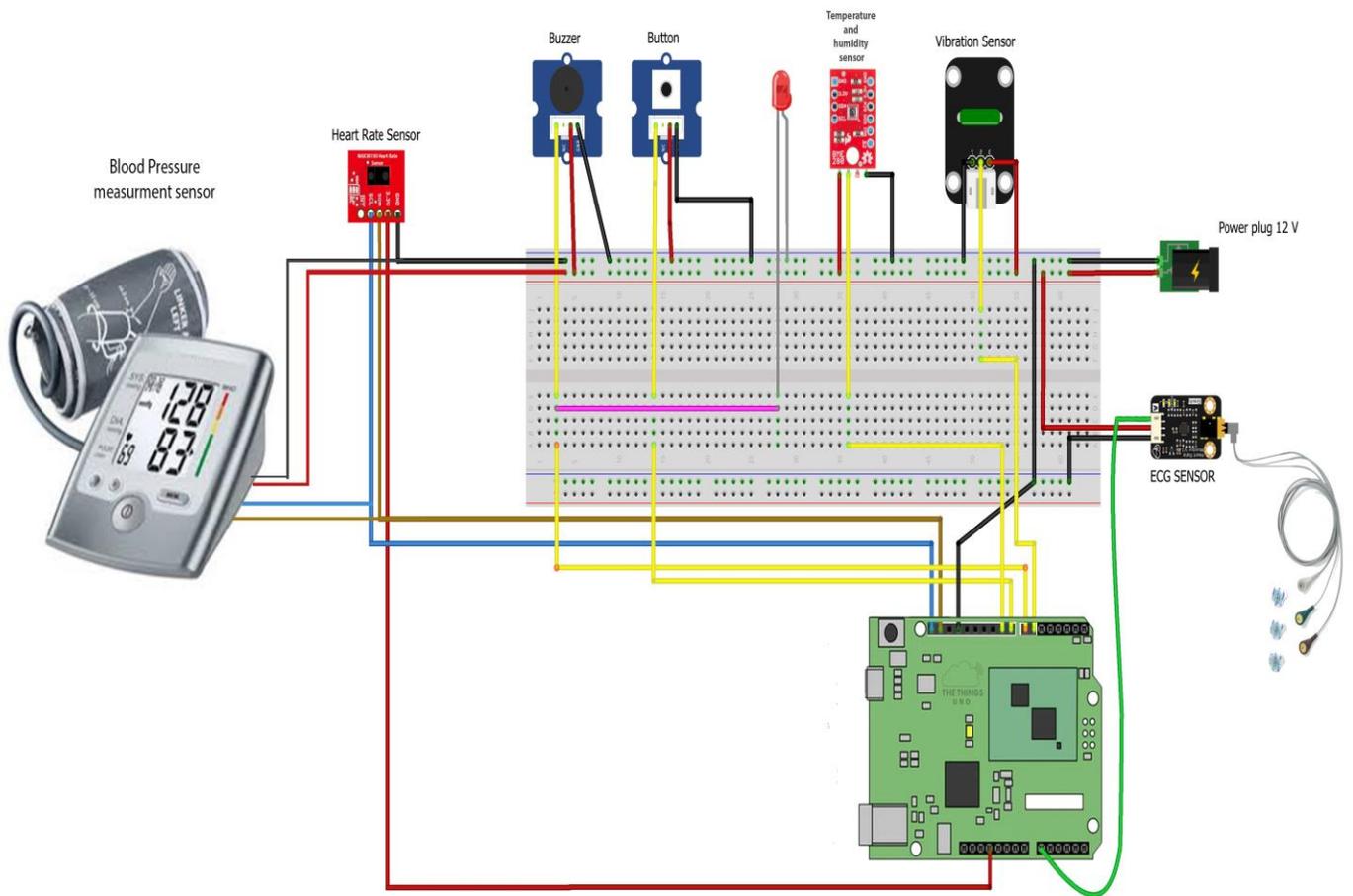
  // Wait a maximum of 10s for Serial Monitor
  while (!debugSerial && millis() < 10000)
  {
    debugSerial.println("-- STATUS");
    ttn.showStatus();
    debugSerial.println("-- JOIN");
    ttn.join(appEui, appKey);
    Serial.println(F("BME280 test"));
    Serial.println("initializing MAX30100");
    pox.begin();

    pox.setOnBeatDetectedCallback(onBeatDetected);
    bool status;
    status = bme.begin();
    if (!status) {
      Serial.println("Could not find a valid BME280 sensor, check wiring!");
      while (1);
    }
    pinMode(7, OUTPUT);
    pinMode(A0,INPUT);
    pinMode(8,INPUT);
    pinMode(6,INPUT);
  }
  void loop()
  {
    debugSerial.println("-- LOOP");
    h_rate = analogRead(A0);
    button = digitalRead(8);
    temperature = pox.getTemperature();
    spo2 = pox.getSpO2();
    bpm = bpm;
    humidity = bme.readHumidity();
    movement = digitalRead(6);
    byte payload[6];
    payload[0] = highByte(bpm);
    payload[1] = lowByte(temperature);
    payload[2] = highByte(humidity);
    payload[3] = lowByte(movement);
    payload[4] = lowByte(spo2);
    payload[5] = lowByte(button);
    payload[6] = lowByte(h_rate);
    debugSerial.print("Temperature: ");
    debugSerial.println(temperature);
    debugSerial.print("Humidity: ");
  }
}

```

```
debugSerial.println(humidity);  
debugSerial.print("BPM: ");  
debugSerial.println(bpm);  
debugSerial.print("SPO2: ");  
debugSerial.println(spo2);  
debugSerial.print("H_rate: ");  
debugSerial.println(h_rate);  
debugSerial.print("Button: ");  
debugSerial.println(button);  
debugSerial.print("Movement: ");  
debugSerial.println(movement);  
ttn.sendBytes(payload, sizeof(payload));  
delay(20000);  
}
```

Circuit Diagram



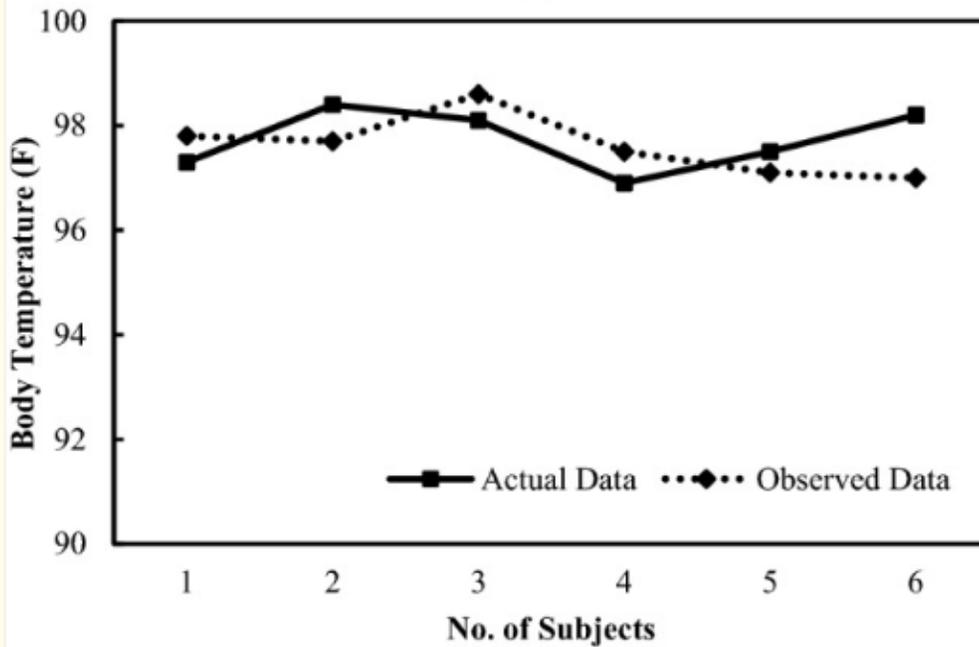
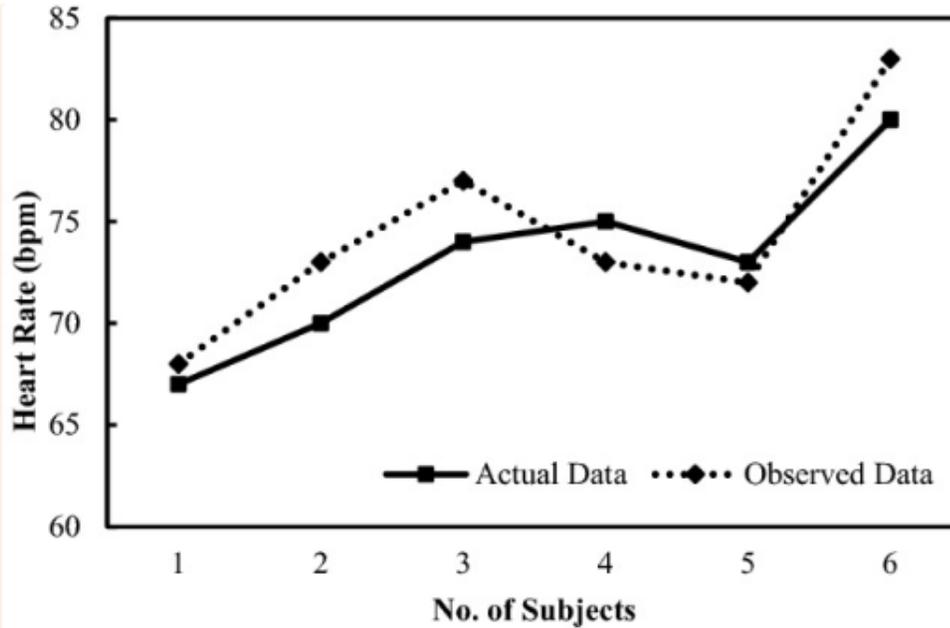
Heart rate data collected by analog machine (actual) and developed system (observed)

Subjects	Actual data (bpm)	Observed data (bpm)	Error (%)
S ₁	67	68	1.49
S ₂	70	73	4.28
S ₃	74	77	4.05
S ₄	75	73	2.66
S ₅	73	72	1.36
S ₆	80	83	3.75

Results

Body temperature data collected by analog machine (actual) and developed system (observed)

Subjects	Actual data (°F)	Observed data (°F)	Error (%)
S ₁	97.3	97.8	0.51
S ₂	98.4	97.7	0.71
S ₃	98.1	98.6	0.50
S ₄	96.9	97.5	0.62
S ₅	97.5	97.1	0.41
S ₆	98.2	97.0	0.81



4. Conclusion and Future Work

The future holds to these sensory networks and their long range capabilities. The present model uses low range low power sensors, but going ahead we need high range high power sensors. Further the Gateway and Cloud needs to be integrated in one system. The analysis is done in the cloud and the data is visualized using a mobile phone app. But in the near future we need an application which is not internet dependent. The present system largely depends on the internet and cloud computing, whereas there are regions like Kargil and Siachen in India which have zero or very low access to Internet. Hence we need an evacuation sensory system which is independent of the internet.

