"SISTEMA DE COMUNICACIÓN DE LARGO ALCANCE (LoRa), PARA LA GESTION Y EL MONITOREO DEL AGUA EN COMUNIDADES RURALES ALTOANDINAS DE HUANCAVELICA - PERÚ"

Report 01, del proyecto "Comunicación dedicada para la gestión y monitoreo del agua en zona altoandina de Perú (Fase 1)", realizada entre la UPC y UNH.

Pol Auladell, Pau Font, Daniel Navarrete y Anna Sánchez

Junio de 2020

Código del proyecto: 2019-B012

Responsable: Filimon Alejandro Quispe Coica

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Centre de Cooperació per al Desenvolupament

ESCGD
EngineeringSciences&GlobalDev

UNIVERSIDAD NACIONAL DE HUANCAVELICA
"Universidad Alto Andina Mejorando Calidad de Vida"
LICENCIADA

Proyecto: “Comunicación dedicada para la gestión y monitoreo del agua en zona altoandina de Perú (Fase 1)”

Código: 2019-B012

Responsable del proyecto: Filimon Alejandro Quispe Coica

Financiación:

Este proyecto se llevó a cabo gracias a la financiación del Centro de Cooperación para el Desarrollo (CCD) de la Universitat Politècnica de Catalunya (UPC), y el apoyo de la Universidad Nacional de Huancavelica (UNH).

Un agradecimiento especial, a la Colección Española de Cultivos Tipo (CECT) de la Universitat de València, por la donación de recursos económicos que permitieron la adquisición de materiales e insumos adicionales, para el monitoreo de la calidad del agua. Asimismo, a investigadores y colaboradores de CETAQUA, Centro Nacional de Microelectrónica (CNM) de Barcelona, Unidad de Salud Ambiental de la Red de Salud Huancavelica, DIRESA-Huancavelica.

Cita sugerida:

Auladell, P., Font, P., Navarrete, D., & Sánchez, A. (2020). *“Sistema de comunicación de largo alcance (LoRa), para la gestión y el monitoreo del agua en comunidades rurales altoandinas de Huancavelica - PERÚ.”* Barcelona.

Barcelona

Junio del 2020

“SISTEMA DE COMUNICACIÓN DE LARGO ALCANCE (LoRa), PARA LA GESTION Y EL MONITOREO DEL AGUA EN COMUNIDADES RURALES ALTOANDINAS DE HUANCAVELICA - PERÚ”

Report 01, del proyecto “Comunicación dedicada para la gestión y monitoreo del agua en zona altoandina de Perú (Fase 1)”, realizada entre la UPC y UNH.

Pol Auladell, Pau Font, Daniel Navarrete y Anna Sánchez

Junio de 2020

Código del proyecto: 2019-B012

Responsable: Filimon Alejandro Quispe Coica

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Centre de Cooperació per al Desenvolupament

ESCGD
EngineeringSciences&GlobalDev

UNIVERSIDAD NACIONAL DE HUANCAVELICA
"Universidad Alto Andina Mejorando Calidad de Vida"
LICENCIADA

PREFACIO

El presente documento es el report 01 del proyecto de “Comunicación dedicada para la gestión y monitoreo del agua en zona altoandina de Perú (Fase 1)”. Surge como iniciativa por la falta de medición remota-insitu, deslocalización, baja cobertura de internet o de medios de comunicación limitados en algunas comunidades rurales de Huancavelica. En consecuencia, la información no llega oportunamente a los gestores locales de agua y saneamiento (como las municipalidades, JASS, ATM). Es en ese sentido, que en esta primera fase se hacen pruebas con sistemas de comunicación de largo alcance (LoRa) y de bajo consumo energético: Inicialmente en el campus de la Universitat Politècnica de Catalunya (UPC) y la comunidad de Fórnols de Matarranya, posteriormente han sido replicados en el campus de la Universidad Nacional de Huancavelica y las comunidades rurales de Huancavelica.

Por otra parte, en este documento encontrará información desglosada de los procedimientos realizados para lograr alcanzar la conectividad entre los sistemas de comunicación LoRa con Raspberry Pi y Gateway. Los scripts de programación también son mostrados, para que cualquier estudiante o técnico pueda replicarlo sin muchos inconvenientes.

Finalmente, si bien la Fase I del proyecto ha culminado con muchos aprendizajes, se espera que en la Fase II del proyecto se consolide la integración de los sistemas de comunicación con los sistemas de monitoreo de la calidad del agua.

A todos los involucrados en el proyecto, mi sincero agradecimiento.

Filimon Alejandro Quispe Coica

Responsable del proyecto de cooperación

Índice

1. Enviar Hello World desde un LoRa HAT para Raspberry Pi a un Gateway mediante LoRa

A continuación, se explicarán los pasos a seguir para conseguir enviar un mensaje de forma continua mediante Lora.

Para seguir el tutorial se dan por supuesto los siguientes puntos:

- Se ha registrado un Lora gateway a The Things Network, se ha configurado a las frecuencias adecuadas y este se encuentra conectado en Internet y activo. En nuestro caso se ha utilizado el modelo OLG02 de Dragino. El manual de usuario es el siguiente: https://www.dragino.com/downloads/index.php?dir=LoRa_Gateway/LG02-OLG02/
- Se ha configurado una Raspberry Pi activando la comunicación SSH y conectándola en la red Wi-Fi.

Una vez hecha esta configuración inicial pasamos ya a la transmisión de un mensaje LoRa. En nuestro caso la transmisión se hará desde un hat de Lora por la Rasperry Pi hasta el Gateway. El hat utilizado es el Lora/GPS hat de Dragino que como se verá más adelante, consta de un módulo GPS a partir del cual podemos enviar la ubicación del dispositivo.

Figura 1. LORA Hat. Fuente: https://wiki.dragino.com/index.php?title=Lora/GPS_HAT

Entonces añadimos una aplicación a nuestra consola de The Things Network (https://wiki.dragino.com/index.php?title=Connect_to_TTN#Use_LoRa_GPS_HAT_and_RPi_3_as_LoRa_End_Device) y programamos el código para hacer la transmisión. El código baso utilizado es el siguiente https://github.com/ernstdevreede/lmic_pi/archive/master.zip.

Una vez lo tenemos descargado, para su correcto funcionamiento nos hay que cambiar las direcciones del fichero: lmic_pi-master/examples/thethingsnetwork-send-v1/**thethingsnetwork-send-v1.cpp**

Las direcciones que tenemos que cambiar son las siguientes y se encuentran en el apartado del nuevo dispositivo de la aplicación creada a TTN.

Importante: Hay que poner las direcciones en hexadecimal y con el bit más significante delante.

```
// LoRaWAN Application identifier (AppEUI)
static const u1_t APPEUI[8]  = { }; //MSB

// LoRaWAN DevEUI, unique device ID
static const u1_t DEVEUI[8]  = { }; //MSB

// LoRaWAN NwkSKey, network session key
// Use this key for The Things Network
static const u1_t DEVKEY[16] ={ }; //MSB

// LoRaWAN AppSKey, application session key
// Use this key to get your data decrypted by The Things Network
static const u1_t ARTKEY[16] = { }; //MSB

// LoRaWAN end-device address (DevAddr)
// See http://thethingsnetwork.org/wiki/AddressSpace
static const u4_t DEVADDR =  ; // <-- Change this address for every node!
```

Figura 2. Líneas donde hay que añadir la dirección HEX

Hechos estos cambios, ejecutamos el código y hemos visto que daba varios errores. Para solucionarlos hemos hecho varias modificaciones:

- Del fichero lmic_pi-master/lmic/**oslmic.h** hemos modificado la siguiente línea para que el programa no se quede bloqueado al fallar y siga enviando mensajes:

```
#define ASSERT(cond) if(!(cond)) // hal_failed(__FILE__,__LINE__)
```

- Del fichero que contiene el programa principal: lmic_pi-master/examples/thethingsnetwork-send/**thethingsnetwork-send-v1.cpp** hemos modificado la función `do_send()` de la siguiente manera:

```
static void do_send(osjob_t* j){
      int numero = 0;
      time_t t=time(NULL);
      fprintf(stdout, "[%x] (%ld) %s\n", hal_ticks(), t, ctime(&t));
      // Prepare upstream data transmission at the next possible time.
      char buf[100];
      sprintf(buf, "Hello world! [%d]", cntr++);
      int i=0;
      while(buf[i]) {
        mydata[i]=buf[i];
        i++;
      }
      mydata[i]='\0';
      LMIC_setTxData2(1, mydata, strlen(buf), 0);
    // Schedule a timed job to run at the given timestamp (absolute system tim
e)
    os_setTimedCallback(j, os_getTime()+sec2osticks(20), do_send);

}
```

De este modo, al ejecutar el programa vemos como se empiezan a enviar mensajes indefinidamente. Desde TTN podemos comprobar que estos mensajes llegan al Gateway tanto como ver todos los mensajes enviados por el dispositivo (Applications –> Nombre dispositivo –> Fecha). Observamos que

el contenido de los mensajes está en forma hexadecimal, para cambiarlo y ver los datos en ASCII modificamos la función decoder de Payload Format (en TTN) de la siguiente manera:

```
function Decoder(bytes, port) {
  return {
      Value: String.fromCharCode.apply(null, bytes)
  };
}
```

De este modo observamos que se transmiten “Hello world” continuamente, con una frecuencia de 1 mensaje cada 20 según aproximadamente

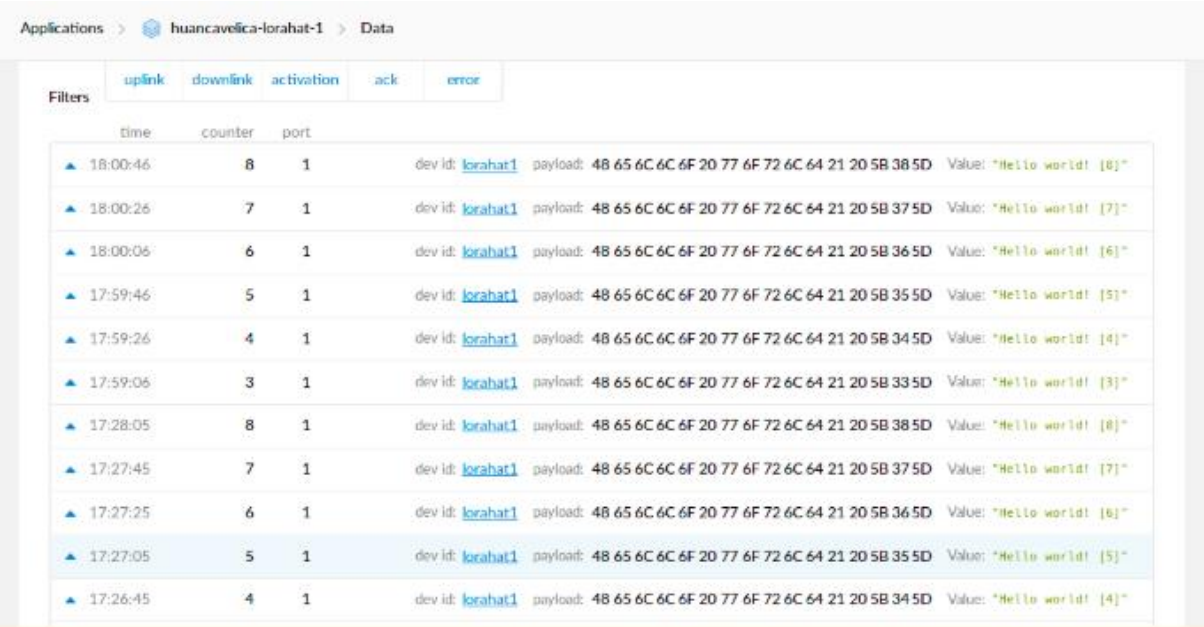


Figura 3. Fuente: Captura de los datos recibidos en TTN

Una vez ya hemos conseguido transmitir datos entre el hat y lo gateway solo nos falta procesar los datos recibidos para obtener la información deseada.

2. Enviar posición GPS con el LoRa HAT mediante LoRa

El hat LoRa de Dragino para RPi puerta incorporado un módulo GPS con el cual podemos trabajar y obtener la información de la localización a partir de este y la RPi. Para nuestro proyecto usamos estos hats para transmitir información del estado y calidad del agua, así como la localización de las diferentes comunidades con las que trabajamos. Así pues, a partir de una Raspberry Pi y de un hat de LoRa con GPS hemos hecho llegar esta información a uno gateway central situado, en nuestro caso, en el edificio de la Universidad Nacional de Huancavelica.

El material que utilizamos es el siguiente:

- OLG01 LoRa Gateway de Dragino
- RPi 3B+
- LoRa Hat GPS

Para poder transmitir información a TTN con el Hat, nos basamos lo el ejemplo desarrollado por Dragino y por IBM (https://github.com/dragino/lmic_pi/) (examples/*thethingsnetwork-*send-*v1). Por lo tanto, primero tenemos que importar la librería a la RPi (con un git clone), y poner las credenciales que hemos encontrado al TTN, en modo bit más significativo (MSB). Tenemos que hacer un make para compilar el archivo con el Makefile que el ejemplo da, y ejecutamos, ahora tendría que poder transmitir un Hello World.

Configuración módulo GPS y RPI

Pero el objetivo que tenemos, es enviar las coordenadas GPS, por lo tanto, el primer paso es habilitar la ubicación a la Raspberry:

1) En /boot/config.txt añadimos al final los siguientes parámetros:

```
dtparam=spi=on
dtoverlay=pi3-disable-bt-overlay
core_freq=250
enable_uart=1
force_turbo=1
```

2) En /boot/cmdline.txt sustituimos todo el archivo por:

```
dwc_otg.lpm_enable=0 console=tty1 root=/dev/mmcblk0p2 rootfstype=ext4 elevator
=deadline fsck.repair=yes rootwait
```

3) Deshabilitamos la configuración bluetooth que viene por defecto, escribiendo en la terminal:

```
sudo systemctl disable hciuart
sudo nano /lib/systemd/system/hciuart.service
```

Ahora sustituir

```
After=dev-serial1.device
```

por

```
After=dev-ttyS0.device
```

4) Para guardar la configuración reiniciamos y actualizamos la raspberry:

```
sudo apt-get update
sudo apt-get upgrade
sudo reboot
```

5) Los datos GPS se están guardando continuamente a **/dev/ttyS0**. Por lo tanto, para provocar que se dejen de guardar en aquel fichero, lo deshabilitamos y hacemos un reboot:

```
sudo systemctl stop serial-getty@ttyS0.service
sudo systemctl disable serial-getty@ttyS0.service
sudo shutdown -r now
```

6) La librería de RPi que se utiliza para trabajar con los datos es gpio, por lo tanto, la instalamos:

```
sudo apt-get install gpsd gpsd-clients python-gps libgps-dev
```

Para comprobar que está bien instalada escribimos:

```
cgps -s
```

Ahora podemos ver que estamos recibiendo las coordenadas GPS en nuestra RPi.

En caso de que no leamos datos GPS (el HAT tiene que tener visibilidad al cielo), tenemos que activar manualmente el paquete gpsd:

```
sudo systemctl stop gpsd.socket
sudo systemctl disable gpsd.socket
sudo killall gpsd
sudo gpsd /dev/ttyS0 -F /var/run/gpsd.sock
```

Si queremos que el paquete gpsd funcione automáticamente, escribimos de nuevo:

```
sudo systemctl enable gpsd.socket
sudo systemctl start gpsd.socket
```

Transmitir los datos GPS

Ahora el que toca hacer es modificar el código proporcionado por Dragino para poder transmitir los datos GPS al TTN.

Primeramente, importaremos las librerías necesarias que son:

```
#include <iostream>
#include <iomanip>
#include <ctime>
#include <sstream>
#include <libgpsmm.h> // Esta es la que acabamos de instalar con el paquete gpsd
```

Después añadimos un método para captar la señal GPS con el gpsd:

```
static void getGPS(){
    gpsmm gps_rec("localhost", DEFAULT_GPSD_PORT);
    gps_rec.stream(WATCH_ENABLE | WATCH_JSON);
    struct gps_data_t *gpsd_data;
        while (((gpsd_data = gps_rec.read()) == NULL) || (gpsd_data->fix.mode < MODE_2D)){}
    auto latitude  { gpsd_data->fix.latitude };
    auto longitude { gpsd_data->fix.longitude };
        fprintf(stdout,"%f, %f\n", latitude, longitude);
    int32_t lat = latitude * 1000000;
    int32_t lon = longitude * 1000000;
        coords[0] = lat >> 24;
        coords[1] = lat >> 16;
        coords[2] = lat >> 8;
        coords[3] = lat;
        coords[4] = lon >> 24;
        coords[5] = lon >> 16;
        coords[6] = lon >> 8;
        coords[7] = lon;
}
```

Lo enviamos así para poderlo descifrar al TTN, puesto que sólo podemos enviar como mucho 8 bits a la vez y la localización tiene 32, por lo tanto lo dividimos en un bus de 8 (lo tenemos que declarar como **`uint8_t coords[8];`** al inicio del script) para que realmente sea un buzo de 8 bits con 8 datos.

Finalmente, al método do_send, gritamos al `getGPS()` y enviamos el bus que hemos creado:

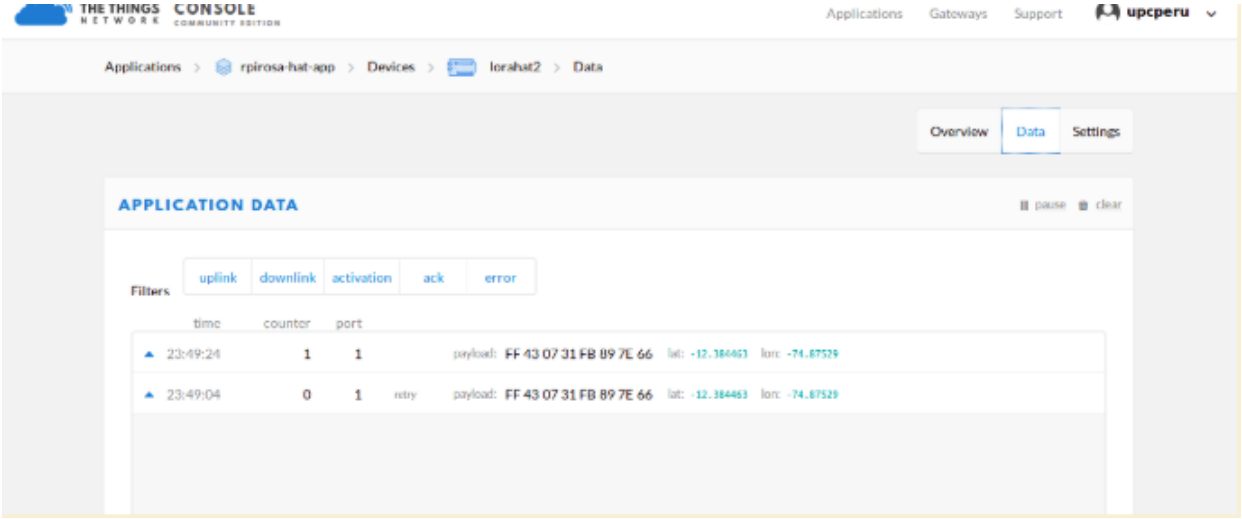
```
LMIC_setTxData2(1, coords, sizeof(coords), 0);
```

Para ejecutar un fichero c++, hay qye hacer el Makefile, que queda de la siguiente manera:

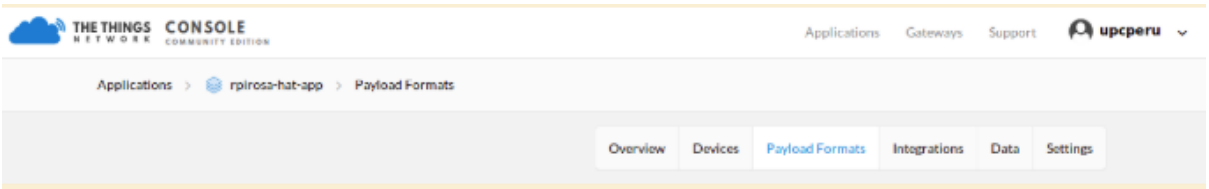
```
CFLAGS=-I../../lmic
LDFLAGS=-lwiringPi
LDLIBS=-lgps
SRCS = prova.cpp ../../lmic/*.o
lecturagps: prova.cpp
      cd ../../lmic && $(MAKE)
      g++ -Wall -pedantic $(LDFLAGS) $(CFLAGS) -o lecturagps $(SRCS) $(LDLIBS)

all: lecturagps

.PHONY: clean clean: rm -f *.o lecturagps
```

Conversión mensaje recibido a TTN

A The Things Network, en el apartado de Data del Dispositivo que previamente habremos creado sobre su respectiva aplicación, encontraremos el mensaje recibido.



Posiblemente el que se muestre por vuestra pantalla no coincida exactamente con la pantalla que os mostramos. Si solo conseguís ver el Payload en formato hexadecimal, tendréis que modificado el formato de este para que se muestre tal cual vemos en la figura anterior. Por esto accedemos a `Applications > 'Tu aplicación' > Payload Formats`



y ponemos el siguiente código, donde se hace el procedimiento inverso al algoritmo seguido para transmitir la posición GPS:

```
function Decoder(bytes, port) {
    var lat = ((bytes[0] << 24) | (bytes[1] << 16) | (bytes[2] << 8) | bytes[3]) /
 1000000;
    var lon = ((bytes[4] << 24) | (bytes[5] << 16) | (bytes[6] << 8) | bytes[7]) /
 1000000;
    return {
      lat: lat,
      lon: lon,
    }
}
```

Una vez hecha este paso lo guardas y aplicas. Los mensajes grabados por el dispositivo a tu aplicación los encontrarás en el formato mostrado.

3. Enviar Hello World desde un LoRa HAT a otro LoRa HAT

Tras conseguir transmitir datos de un punto a otro, nos disponemos a hacer conexión multinodo con el fin de conseguir más flexibilidad en las comunicaciones.
El primer paso es enviar información entre hats. Hemos seguido un tutorial dado por dragino, el cual explicamos aquí también.

```
wget https://codeload.github.com/dragino/rpi-lora-
tranceiver/zip/master

unzip master

// Accedemos a la carpeta que hemos descomprimido (cd CARPETA)

make
```

Ahora ya está listo para funcionar. Aunque es necesario cambiar la frecuencia de trabajo en función de la región o país que estemos trabajando. Como hemos mencionado en otros tutoriales, la frecuencia que se debería usar en América Latina es 915MHz, mientras la europea es 868MHz. Por lo tanto, si estamos en Latino América, hay que cambiar la frecuencia que aparece en el main.c por 915000000. No olvidar que hay que cambiar la frecuencia en los dos hats.

Una vez tenemos los programas preparados, ejecutamos.

```
main sender //para enviar
main rec //para escuchar
```

4. Packet Relay: Conexión entre Hat - Gateway (bridge) - Gateway

Simplemente tenemos que usar el programa del primer punto, donde el LoRa HAT transmite un Hello World a un Gateway. Posteriormente, para hacer que un Gateway actúe como repetidor, únicamente debemos cambiar la configuración del Gateway que actúa como repetidor. Esta disposición de dispositivos será necesaria cuando la señal del LoRa HAT no llegue al Gateway porque no existe una línea de vista directa.

Simplemente debemos cambiar el siguiente parámetro de la configuración del Gateway, desde la interfaz de configuración:

En este parámetro deberemos poner la opción de Packet Relay. Esto hará que todos los paquetes que le lleguen a este Gateway los repita para que pueda llegar a otro Gateway. Como se puede deducir, este Gateway al funcionar como repetidor no es necesario que esté conectado a Internet ya que no subirá los mensajes al servidor TTN. Sí que es necesario tener el último Gateway conectado a Internet si deseamos poder visualizar los mensajes desde TTN.

5. Enviar Hello world desde un LoRa HAT a otro LoRa HAT y que este lo repita al Gateway principal

Una vez lograda la conexión entre los hats, lo siguiente es enviar la información recibida al gateway. Para ello utilizaremos el programa ya usado anteriormente en el envío de información de hat a gateway. A partir de este nuevo programa llamaremos mediante la función SYSYEM(); al script que enviará al GW.

```
void receivepacket() {
    long int SNR; int rssicorr;
    if(digitalRead(dio0) == 1) {
    if(receive(message)) {
        byte value = readReg(REG_PKT_SNR_VALUE);
        if( value & 0x80 ) // The SNR sign bit is 1 {
        // Invert and divide by 4 value = ( ( ~value + 1 ) & 0xFF ) >> 2; SNR = -
value; } else
        {
            // Divide by 4 SNR = ( value & 0xFF ) >> 2; }
            if (sx1272) {
            rssicorr = 139; } else {
            rssicorr = 157; }
            printf("Packet RSSI: %d, ", readReg(0x1A)-
rssicorr); printf("RSSI: %d, ", readReg(0x1B)-
rssicorr); printf("SNR: %li, ", SNR); printf("Length: %i", (int)receivedbytes); pr
intf("\n"); printf("Payload: %s\n", message);
            FILE * missatge; missatge = fopen("missatge.txt","w+"); if (missatge!=
NULL) { fputs(message,missatge);
            fclose(missatge); system("~/lmic_pi-master/examples/thethingsnetwork-
send- v1/thethingsnetwork-send-
v1.cpp");//o su respectivo directorio para //vuestro caso
            fprintf(stdout,"hem pausat el SENDTOGW i seguim"); } } // received a m
essage
        }
    }
}
```

Como podéis ver, primeramente, guardamos en un fichero externo el mensaje recibido. Posteriormente se ejecuta mediante la función SYSTEM el script thethingsnetwork- send-v1.cpp para enviar al gateway. Para usar correctamente el script debemos editarlo también de la siguiente manera:

```
static void do_send(osjob_t* j){
    time_t t=time(NULL); fprintf(stdout, "[%x] (%ld) %s\n", hal_ticks(), t, ctime(
&t));
    int c; char* cstr; FILE * missatge;
    missatge = fopen("missatge.txt","r");
    char mystring [100];
        if (missatge == NULL) perror ("Error opening file");
        else { if ( fgets (mystring , 100 , missatge) != NULL ){ puts (mystring);}
 }
    fclose(missatge);
    char buf[100];
    int i=0;
    sprintf(buf,mystring, cntr++);
    while(buf[i]) {
        mydata[i]=buf[i];
        i++;
    }
    mydata[i]='\0';
    LMIC_setTxData2(1, mydata, strlen(buf), 0);
    remove("missatge.txt");
    os_setTimedCallback(j, os_getTime()+sec2osticks(20), do_send);
}
```

De esta manera obtenemos del fichero externo el mensaje a enviar. Una vez enviado, como podemos ver, borramos el fichero "missatge.txt" para asegurar cada vez un único mensaje. Para que funcione, tenemos que cerrar el programa cada vez que recibimos datos del otro hat. Para ello modificamos el main de esta forma:

```
int main() {
    setup();
    while (1) { do_send(&sendjob); os_runloop(); }
    return 0;
}
```

Ahora, lo que debemos hacer un bucle para iterar de nuevo el programa y recibir datos todo el rato para luego enviarlos contínuamente. Esto lo vamos a hacer mediante el `/etc/init.d` donde automatizaremos el programa.

6. Base de datos

El procesamiento de datos desde The Things Network es muy limitado. Es por esto que es conveniente descargarnos toda la información importante a un servidor externo (como podría ser nuestra máquina) para así poder analizar los dato de manera mucho más extensa y personalizada. En este documento se exponen los diferentes métodos utilizados para extraer toda la información importante de The Things Network y guardarla en una base de datos propia.

Método 1: Utilizar la integración HTTP de TTN

The Things Network dispone de diferentes integraciones para guardar la información y visualizarla en una página web externa. La principal ventaja de este método es la simplicidad y sencillez. Además, se enviará toda la información a nuestro servidor web sin procesar, por lo que luego la podremos manipular de manera personalizada. Aún así, este método obliga a tener un servidor web, por lo que no soluciona el problema de poder manipular los datos desde nuestro propio espacio.

Entre las diferentes opciones disponibles, en nuestro caso se ha testeado la integración HTTP, que tal y como indica su nombre, envía la información a un servidor web externo mediante el protocolo HTTP. El procedimiento para implementar la integración se encuentra detallado en la web de TTN[1].

Método 2: Utilizar un script que descarga los datos de la integración Storage

Otra de las integraciones con las cuales podemos trabajar desde TTN es Storage. Esta alternativa es interesante ya que mediante un script básico se puede descargar a nuestro espacio todo lo que la integración guarda en la nube. El principal inconveniente de este método es que la integración sólo permite almacenar información de los últimos 7 días, por lo que cada 7 días habría que ejecutar el script.

Para descargar los datos y guardarlos en un fichero utilizamos el siguiente script de bash:

```
#!/bin/bash
echo "Welcome"
curl -X GET --header 'Accept: application/json' --header 'Authorization:
key ttn-account-v2.xxxxx' 'https://huancavelica-lorahat-
1.data.thethingsnetwork.org/api/v2/query?last=7d' >> info
echo "this is the whole list of dir"
```

Método 3: Utilizar Python SDK y un script que guarde los mensajes recibidos

Este método consiste en ejecutar en bucle un programa Python que permite descargar la información recibida en tiempo real. La implementación de este método se encuentra detallada en la web de TTN[2]. El programa base ha estado modificado para evitar errores de permisos:

```
import time
import ttn
app_id = "huancavelica-lorahat-1"
access_key = "ttn-account-**********************"
#insert here your access_key
```

[1] Implementación de la integración HTTP: https://www.thethingsnetwork.org/docs/applications/http/
[2] Aplicación Python SDK: https://www.thethingsnetwork.org/docs/applications/python/

```
def uplink_callback(msg, client):
  print("\n")
  print(msg)
handler = ttn.HandlerClient(app_id, access_key)
mqtt_client = handler.data()
mqtt_client.set_uplink_callback(uplink_callback)
mqtt_client.connect()
try:
    while True:
time.sleep(40)
finally:
    mqtt_client.close()
```

Ejecutando este programa obtenemos una información mucho más extensa (no sólo el mensaje como en los casos anteriores, sino que también el nivel de señal, la relación señal-ruido, etc.). La principal desventaja de este método es el hecho que el programa tiene que estar corriendo continuamente. El programa creado para que ejecute y guarde el resultado en un fichero es el siguiente:

```
#!/bin/bash
 echo "Welcome to the DataBase of Huancavelica Project"
 echo "Writing messages to the DataBase..."
 python pythonsdk.py >> DataBase
```

Una vez tenemos la información en nuestra máquina, hay que procesarla. Para ello se ha modificado el programa anterior de forma que guarde los datos en un fichero en formato JSON. Este formato permite trabajar muy cómodamente con los datos. Cuando tenemos los datos procesados y fáciles de analizar, parece interesante poder colgarlos en la nube para poder acceder a ellos desde cualquier sitio. Para ello utilizamos un servidor gratuito de Google llamado Firebase.

Finalmente, el código queda modificado de tal forma:

```
import time, ttn, json, firebase_admin, datetime
from firebase_admin import credentials
from firebase_admin import db
from collections import OrderedDict

# Fetch the service account key JSON file contents
cred = credentials.Certificate('firebase-adminsdk.json')

# Initialize the app with a service account, granting admin privileges
firebase_admin.initialize_app(cred, {
    'databaseURL': 'https://huancadb.firebaseio.com/'
})

# TTN keys
```

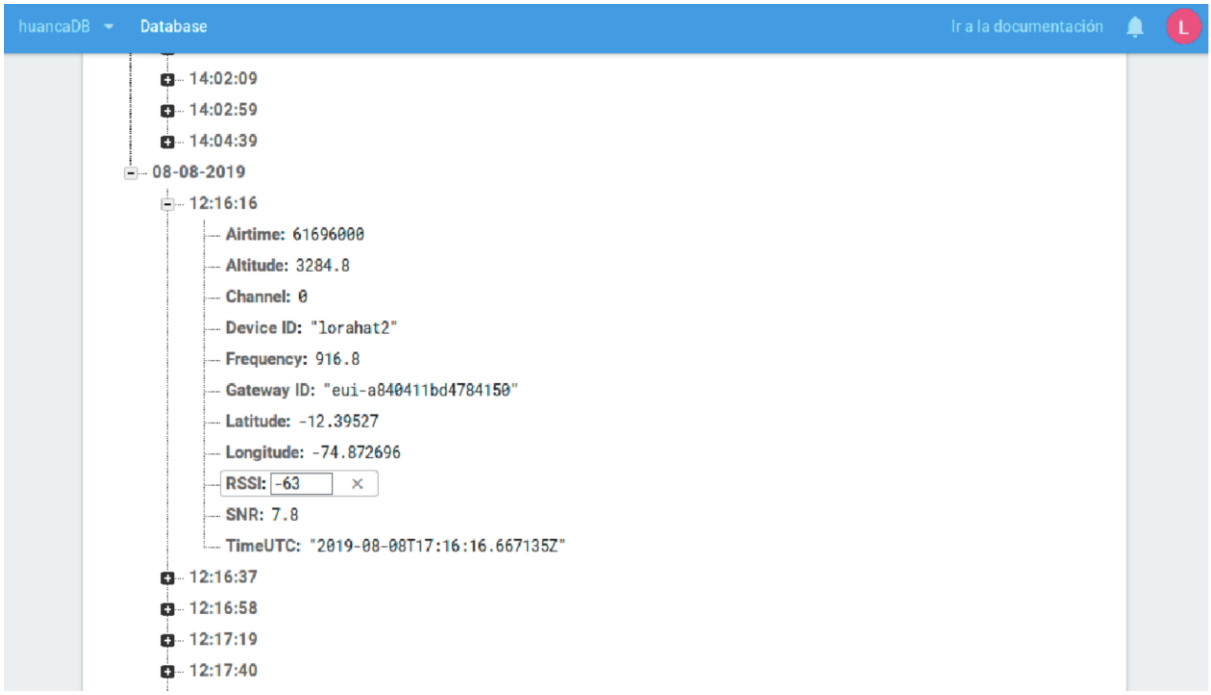
```
app_id = "huancavelica-lorahat-1"
access_key = "ttn-account-v2.xxxxxx"

# Open the JSON file
jsonFile= "./db.json"
f = open(jsonFile,'r')
usefuldata = f.read()
if usefuldata == "" or usefuldata == None:
      usefuldata = []
else:
      usefuldata=json.loads(usefuldata)
f.close()

# Get TTN data
def uplink_callback(msg, client):
      timeUTC=datetime.datetime.strptime(msg[7][5][0][5], '%Y-%d-
%mT%H:%M:%S.%fZ')
      horaperuana = datetime.timedelta(hours=05)
      time = timeUTC - horaperuana
      date = datetime.date.strftime(time, "%d-%m-%Y_%H:%M:%S")

      newData={"Device ID": msg[0],
      "Value": msg[3][3],
      "Altitude": msg[3][1],
      "Latitude": msg[3][0],
      "Longitude": msg[3][2],
      "Airtime": msg[7][2],
      "Frequency": msg[7][4],
      "SNR": msg[7][5][0][4],
      "TimeUTC": msg[7][5][0][5],
      "RSSI": msg[7][5][0][7],
      "Channel": msg[7][5][0][8],
      "Gateway ID": msg[7][5][0][0]}

      # Put together all the data
      usefuldata.append(newData)
      f = open(jsonFile,'w')
      f.write(json.dumps(usefuldata, sort_keys=True))
      f.close()
      upload2firebase(date,newData)

def upload2firebase(date,newData):
      db.reference('/' + date).set(newData)

# Access to TTN (main function)
handler = ttn.HandlerClient(app_id, access_key)
mqtt_client = handler.data()
mqtt_client.set_uplink_callback(uplink_callback)
```

```
mqtt_client.connect()

try:
      while True:
            time.sleep(40)
finally:
      mqtt_client.close()
```

La información almacenada en el fichero JSON queda organizada de la siguiente manera:


```
{
    "Airtime": 61696000,
    "Altitude": 3283.9,
    "Channel": 0,
    "Device ID": "lorahat2",
    "Frequency": 916.8,
    "Gateway ID": "eui-a840411bd4784150",
    "Latitude": -12.39527,
    "Longitude": -74.872696,
    "RSSI": -63,
    "SNR": 7.8,
    "Time": "2019-08-08T17:16:37.902948Z",
    "Value": null
},
{
    "Airtime": 61696000,
    "Altitude": 3283.6,
    "Channel": 0,
    "Device ID": "lorahat2",
    "Frequency": 916.8,
    "Gateway ID": "eui-a840411bd4784150",
    "Latitude": -12.39527,
    "Longitude": -74.872696,
    "RSSI": -62,
    "SNR": 7.8,
    "Time": "2019-08-08T17:16:58.633355Z",
    "Value": null
```


Y en el servidor de Firebase:



Script para convertir un fichero JSON a CSV:

El formato JSON nos permite tener datos muy fáciles de manejar, pero un poco más difíciles de leer para un usuario cualquiera. Es por esto que se ha desarrollado un código que convierte un fichero JSON a otro de tipo CSV (mucho más interpretable por parte del usuario). El programa es el siguiente:

```
import csv, json, sys
from collections import OrderedDict
#if you are not using utf-8 files, remove the next line
#sys.setdefaultencoding("UTF-8") #set the encode to utf8

inputFile = open('db.json') #open json file
outputFile = open('DataBase.csv', 'w') #load csv file
data = json.load(inputFile,object_pairs_hook=OrderedDict) #load json content
for element in data:
    datetime = element['Time'].split('T')
    element['Date'] = datetime[0]
    element['Time'] = datetime[1][0:8]
    aux = element['Time'].split(':')
    hora = str((int(aux[0])-5)%24)
    element['Time'] = hora.zfill(2)+':'+aux[1]+':'+aux[2]
    element['Message ID'] = element['Date'].replace("-","") +
element['Time'].replace(":","")

inputFile.close()
#Ordre de les columnes a la DB
fieldnames = ['Device ID', 'Gateway ID', 'Message ID', 'Frequency',
'Channel', 'Airtime','RSSI', 'SNR', 'Altitude', 'Latitude',
'Longitude','Value','Date','Time']
writer = csv.DictWriter(outputFile, fieldnames=fieldnames)
writer.writeheader()
```

```
for row in data:
    writer.writerow(row)
```

Finalmente, abriendo el fichero CSV con un programa como Excel se pueden observar los datos de la siguiente manera:

Device ID	Gateway ID	Message ID	Frequency	Channel	Airtime	RSSI	SNR	Altitude	Latitude	Longitude	Value	Date	Time
lorahat2	eui-a840411bd4784150	20190808121637	916.8	0	61696000	-63	7.8	3283.9	-12.39527	-74.872696		2019-08-08	12:16:37
lorahat2	eui-a840411bd4784150	20190808121658	916.8	0	61696000	-62	7.8	3283.6	-12.39527	-74.872696		2019-08-08	12:16:58
lorahat2	eui-a840411bd4784150	20190808121719	916.8	0	61696000	-64	7.8	3284	-12.39527	-74.872696		2019-08-08	12:17:19
lorahat2	eui-a840411bd4784150	20190808121740	916.8	0	61696000	-63	7.8	3284.6	-12.39527	-74.872696		2019-08-08	12:17:40
lorahat2	eui-a840411bd4784150	20190808121822	916.8	0	61696000	-63	7.8	3283.9	-12.39527	-74.872696		2019-08-08	12:18:22
lorahat2	eui-a840411bd4784150	20190808121843	916.8	0	61696000	-63	7.8	3283.1	-12.39527	-74.872696		2019-08-08	12:18:43
lorahat2	eui-a840411bd4784150	20190808121904	916.8	0	61696000	-62	7.8	3283.9	-12.39527	-74.872696		2019-08-08	12:19:04
lorahat2	eui-a840411bd4784150	20190808121925	916.8	0	61696000	-63	7.8	3285.5	-12.39527	-74.872696		2019-08-08	12:19:25
lorahat2	eui-a840411bd4784150	20190808121946	916.8	0	61696000	-63	7.8	3286.6	-12.39527	-74.872696		2019-08-08	12:19:46
lorahat2	eui-a840411bd4784150	20190808122007	916.8	0	61696000	-63	7.8	3287.1	-12.39527	-74.872696		2019-08-08	12:20:07
lorahat2	eui-a840411bd4784150	20190808122028	916.8	0	61696000	-62	7.8	3287.1	-12.39527	-74.872696		2019-08-08	12:20:28
lorahat2	eui-a840411bd4784150	20190808122049	916.8	0	61696000	-63	7.8	3286.7	-12.39527	-74.872696		2019-08-08	12:20:49
lorahat2	eui-a840411bd4784150	20190808122110	916.8	0	61696000	-63	7.8	3285.6	-12.39527	-74.872696		2019-08-08	12:21:10

7. Controlador

El controlador es otro ejemplo de un dispositivo que nos permite transmitir información mediante LoRa. Hemos utilizado el modelo LT-33222 de la marca Dragino para realizar las comunicaciones del controlador. La gran novedad de este dispositivo es que nos permite enviar mensajes de downlink, es decir, mensajes desde el Cloud al dispositivo. Esto nos permitirá que el controlador pueda reaccionar a estos mensajes, por ejemplo, activando una luz o cerrando una válvula.

Material

Para la configuración del controlador se necesita el siguiente material

- USB TTL Adapter
- Jumpers

Manual de uso

El primer paso debe ser conectar el controlador al ordenador para realizar la configuración de las frecuencias de LoRa. El USB nos permitirá la conexión del controlador al ordenador. Usaremos los jumpers para conectar los dos dispositivos.

GND→ Negro
RXD→Blanco
TXD→ Rojo

Para la configuración de las frecuencias, recomendamos descargar el programa SecureCTR.
Para la configuración de las frecuencias se debe seguir el siguiente proceso.

1. Poner password por defecto del controlador por tener acceso AT.
2. AT + FDR Nos permite hacer un reset de fábrica.
3. Poner password por defecto otra vez.
4. AT+CLASS=C Permite trabajar en clase C.
5. AT+ NJM = 0 Configura el modo ABP.
6. AT + ADR = 0 Configura en off el Adaptive Data Rate.
7. AT + DR = 5 Configura el Data Rate.
8. AT + TDF = 60000 Configura el intervalo a 60s.
9. AT + CHS = 868400000 Frecuencia de transmisión a 868.4 MHz
10. AT + RX2FQ = 868400000 Frecuencia del servidor a 868.4MHz

11. AT + RX2DR = 5 Data Rate (debe coincidir con la del servidor).
12. AT + DADDR = 26 01 1A F1 Configuración del ID del controlador
13. ATZ Reset MCU

El Data Rate y el SF deben estar acorde a la región en la que se produzca la comunicación. Por este motivo las siguientes tablas son esenciales para escoger los valores:

DataRate	Configuration	bit/sec	TXPower	EIRP
0	LoRa: SF10 / 125 kHz	980	0	30 dBm – 2*TXpower
1	LoRa: SF9 / 125 kHz	1760	1	28 dBm
2	LoRa: SF8 / 125 kHz	3125	2	26 dBm
3	LoRa: SF7 / 125 kHz	5470	3..13	….
4	LoRa: SF8 / 500 kHz	12500	14	2 dBm

Mensajes de Uplink (Controlador - The Things Network)
Una vez configurado todo el dispositivo, el controlador automáticamente empezará a enviar mensajes. Debemos registrar nuestro dispositivo a The Things Network y desde ahí podremos ver los mensajes.

Mensajes de Downlink (The Things Network - Controlador)
The Things Network tiene la opción de enviar mensajes de downlink. Según cómo sean estos mensajes, podemos llegar a activar/desactivar distintos relays (RO1 se refiere al Relay 1 y el RO2 se refiere al Relay 2) del controlador.

Downlink Code	RO1	RO2
03 00 11	Open	No Action
03 01 11	Close	No Action
03 11 00	No Action	Open
03 11 01	No Action	Close
03 00 00	Open	Open
03 01 01	Close	Close
03 01 00	Close	Open
03 00 01	Open	Close

Cayenne Integration

Una de las maneras de visualizar los datos a tiempo real es la Integración de Cayenne. Existe tanto la versión web como la aplicación móvil y desde ahí se pueden monitorear los distintos dispositivos registrados. Otra gran funcionalidad que incluye Cayenne, es la creación de alarmas. Se puede programar la aplicación para que envíe un mensaje en caso de superar cierto voltaje o corriente en cualquier entrada del controlador.