Laboratoires IoT avec PlatformIO

SmartComputerLab

Contenu

Laboratoire 5 - Développement de simples passerelles IoT

Dans ce laboratoire nous allons développer une architecture intégrant plusieurs dispositifs essentiels pour la création d'un système IoT complet. Le dispositifs central sera la passerelle (*gateway*) entre les liens LoRa et la communication par WiFI.

5.1 Passerelle `LoRa-ThingSpeak`

La passerelle permettra de retransmettre les données reçues sur un lien **LoRa** et de les envoyer par sur une connexion WiFi vers un serveur **`ThingSpeak`**

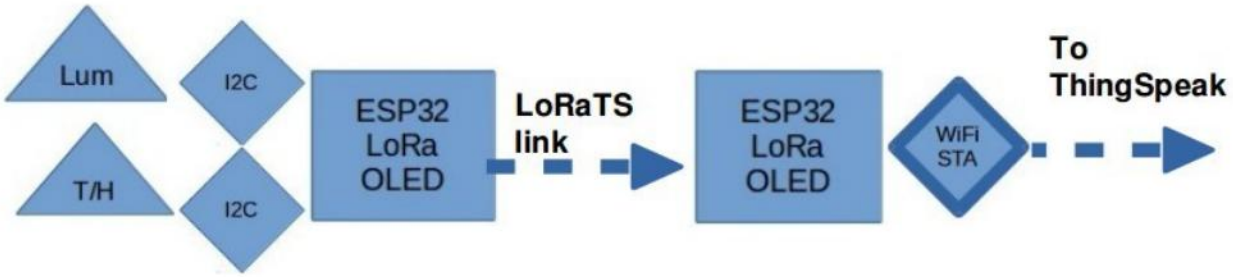

Figure 5.1 Une simple architecture IoT avec un terminal à 2 capteurs et une passerelle **LoRa-WiFi**

5.1.1 Le principe de fonctionnement

La passerelle attend les messages LoRa envoyés dans le format prédéfinie **LoRaTS** sur une interruption I/O redirigée vers la fonction (ISR) **`onReceive()`**. Elle les stocke dans une file de messages (*queue*) en gardant seulement le dernier paquet. La tâche principale, dans la boucle **`loop()`** récupère ce paquet dans la file par la fonction **`xQueueReceive()`** puis l'envoie sur le serveur **`ThingSpeak.`**

Le contenu d'un paquet en format **LoRaTS** est transformé en un ou plusieurs fonctions :
`ThingSpeak.setField(fn,value);`
et la fonction
`ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);`

5.1.2 Les éléments du code

Ci-dessous nous présentons les éléments du code de la passerelle. Les messages LoRa sont envoyés dans un **format simplifié** de **LoRaTS**. Nous avons définie le type de l'union/structure d'un paquet (**`pack_t`**) de la façon suivante. Le format simplifié correspond au transfert des paquets LoRa dans le laboratoire précédent.

```
typedef union      // format simplifie
{
  uint8_t frame[16]; // trames avec octets
  float  data[4];    // 4 valeurs en virgule flottante
} pack_t ;  // paquet d'émission

typedef union // format de base
  {
    uint8_t frame[40];
    struct
      {
        uint8_t head[4];    // packet header
        int chnum;          // channel number
        char key[16];       // write or read key
        float sensor[4];
      } pack;
  } pack_t;
```

Pour le format de base dans l'en-tête **head[4]** nous indiquons :
head[0] – adresse de destination, **0x00** – passerelle, **0xff** – diffusion
head[1] – adresse de source
head[2] – type du message – data **write**/**read**, **control**, ..
head[3] – le masque **0xC0** signifie **field1** et **field2**

Les paquets qui arrivent par le lien LoRa sont captés par l'ISR **onReceive()** et déposés dans une file d'attente **FreeRTOS**. Cette file doit être déclarée par :

```
QueueHandle_t dqueue;  // queues for data packets
```

Puis, dans le **setup()** on instancie la file en lui fournissant le nombre d'éléments et la taille d'un élément:

```
dqueue = xQueueCreate(4,16); // queue for 4 data packets in simple format

void onReceive(int packetSize)
{
pack_t rbuff;  // receive buffer with pack_t format
int i=0;
  if (packetSize == 0) return;   // if there's no packet, return
  i=0;
  if (packetSize==16)   // 16 pour le format simplifie , 40 format de base
    {
    while (LoRa.available()) {
      rbuff.frame=LoRa.read();i++;
      }
      xQueueReset(dqueue); // to keep only the last element
      xQueueSend(dqueue, rbuff, portMAX_DELAY);
    }
  delay(200);
}
```

La fonction de **setup()** :

```
void setup()
{
  Serial.begin(9600);
  WiFi.mode(WIFI_STA);
  WiFiManager wm;
  // wm.resetSettings();
  bool res;
  res = wm.autoConnect("ESP32AP",NULL); // password protected ap
  if(!res)
    {
    Serial.println("Failed to connect");// ESP.restart();
    }
  else
    {
    //if you get here you have connected to the WiFi
    Serial.println("connected...yeey :)");
    }
  ThingSpeak.begin(client); // connexion (TCP) du client au serveur
  delay(1000);
  Serial.println("ThingSpeak begin");
  Serial.println("Start Lora");
  SPI.begin(SCK,MISO,MOSI,SS);
  LoRa.setPins(SS,RST,DI0);
  Serial.println();delay(100);Serial.println();
  if (!LoRa.begin(freq)) {
    Serial.println("Starting LoRa failed!");
    while (1);
  }
Serial.println("Starting LoRa OK!");
LoRa.setSpreadingFactor(sf);
LoRa.setSignalBandwidth(sb);
dqueue = xQueueCreate(4,16); // queue for 4 simple data packets
LoRa.onReceive(onReceive);  // pour indiquer la fonction ISR
LoRa.receive();             // pour activer l'interruption (une fois)
}
```

La fonction de la tâche en **`loop()`** :

Pour le format simplifié :

```
void loop()
{
pack_t rp;    // packet elements to send
  xQueueReceive(dqueue,rp.frame,portMAX_DELAY); // 6s,default:portMAX_DELAY
  ThingSpeak.setField(1,rp.data[0]);
  ThingSpeak.setField(2,rp.data[1]);
  ThingSpeak.setField(3,rp.data[2]);
  ThingSpeak.setField(4,rp.data[3]);
```

Figure 5.1 Une simple architecture IoT avec un terminal à 2 capteurs et une passerelle LoRa-WiFi

```
  Serial.printf("d1=%2.2f,d2=%2.2f,d3=%2.2f,d4=%2.2f\n",
rp.data[0],rp.data[1],rp.data[2],rp.data[3]);
  while (WiFi.status() != WL_CONNECTED) {  delay(500);  }
  ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
  delay(mindel); // mindel is the min waiting time before sending to ThingSpeak
 //LoRa.receive();
}
```

Pour le format de base :

```
loop()
{
pack_t sb;    // packet elements to send
xQueueReceive(dqueue,sb.frame, portMAX_DELAY); // 6s,default:portMAX_DELAY
    if(sb.pack.head[1]==0x01 || sb.pack.head[1]==0x00)
    {
        if(sb.pack.head[2]&0x80) ThingSpeak.setField(1,sb.pack.sensor[0]);
        if(sb.pack.head[2]&0x40) ThingSpeak.setField(2,sb.pack.sensor[1]);
        if(sb.pack.head[2]&0x20) ThingSpeak.setField(3,sb.pack.sensor[2]);
        if(sb.pack.head[2]&0x10) ThingSpeak.setField(4,sb.pack.sensor[3]);
        while (WiFi.status() != WL_CONNECTED) {  delay(500);  }
        ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
    }
    LoRa.receive();
  }
```

5.1.3 Code complet pour une passerelle des paquets en format de base

```
#include <Arduino.h>
#include <WebServer.h>
#include <WiFiManager.h>
#include <SPI.h>
#include <LoRa.h>
#include "ThingSpeak.h"
char ssid[] = "Livebox-08B0"; // your network SSID (name)
char pass[] = "G79ji6dtEptVTPWmZP"; // your network passw
unsigned long myChannelNumber = 1538804;
const char * myWriteAPIKey="YOX31M0EDKO0JATK" ;
int dout=15;
WiFiClient client;

#define SCK 18 // GPIO18 -- SX127x's SCK
#define MISO 19 // GPIO19 -- SX127x's MISO
#define MOSI 23 // GPIO23 -- SX127x's MOSI
#define SS 5 // GPIO05 -- SX127x's CS
#define RST 15 // GPIO15 -- SX127x's RESET
#define DI0 26 // GPIO26 -- SX127x's IRQ(Interrupt Request)
#define freq 434E6
#define sf 7
#define sb 125E3

union
{
uint8_t frame[16]; // trames avec octets
float data[4]; // 4 valeurs en virgule flottante
} rdp ; // paquet de reception

QueueHandle_t msg_queue; // queues for data packets


void onReceive(int packetSize)
{
int rssi=0;
uint8_t buff[16];
if (packetSize == 0) return; // if there's no packet, return
int i=0;
if (packetSize==16)
{
while (LoRa.available()) { buff[i]=LoRa.read();i++; }
xQueueReset(msg_queue);
xQueueSend(msg_queue, (void *)buff, 16);
}
}

void setup()
{
WiFi.mode(WIFI_STA);
Serial.begin(9600);
WiFiManager wm;
//wm.resetSettings();
bool res;
res = wm.autoConnect("ESP32AP",NULL); // password protected ap
if(!res) {
Serial.println("Failed to connect"); // ESP.restart();
}
else {
//if you get here you have connected to the WiFi
Serial.println("connected...yeey :)");
}
msg_queue = xQueueCreate(4,16); // queue for 4 data packets in simple format
ThingSpeak.begin(client); // connexion (TCP) du client au serveur
delay(1000);
Serial.println("ThingSpeak begin");

SPI.begin(SCK,MISO,MOSI,SS);
LoRa.setPins(SS,RST,DI0);
```

```
Serial.println();delay(100);Serial.println();
if (!LoRa.begin(freq)) {
Serial.println("Starting LoRa failed!");
while (1);
}
Serial.println("Starting LoRa OK!");
LoRa.setSpreadingFactor(sf);
LoRa.setSignalBandwidth(sb);
LoRa.onReceive(onReceive); // pour indiquer la fonction ISR
LoRa.receive(); // pour activer l'interruption (une fois)
// puis après chaque émission
}

void loop()
{
// packet elements to send
xQueueReceive(msg_queue,(void *)rdp.frame,6000); // 6s,default:portMAX_DELAY
//LoRa.sleep();
ThingSpeak.setField(1,rdp.data[0]);
ThingSpeak.setField(2,rdp.data[1]);
ThingSpeak.setField(3,rdp.data[2]);
ThingSpeak.setField(4,rdp.data[3]);
Serial.println("in the loop");
Serial.printf("d1=%2.2f,d2=%2.2f,d3=%2.2f,d4=%2.2f\n",
rdp.data[0],rdp.data[1],rdp.data[2],rdp.data[3]);
while (WiFi.status() != WL_CONNECTED) { delay(500); }
ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
delay(15000); // 15 sec min pour ThingSpeak.com
//LoRa.receive();
}
```

A faire :

1. Tester le code de base
2. Ecrire le code complet pour la version de base avec le transfert du numéro du canal et de clé d'écriture dans les paquets LoRa. Tester ce nouveau programme de passerelle avec un puis avec 2 terminaux.
3. Utiliser les capteurs SHT21 et BH1750 pour envoyer des données « réelles »

5.2 Passerelle LoRa – WiFi – broker MQTT

Dans cette partie du laboratoire nous allons développer une passerelle entre un lien LoRa et une connexion WiFi qui porte les messages MQTT vers un broker MQTT.

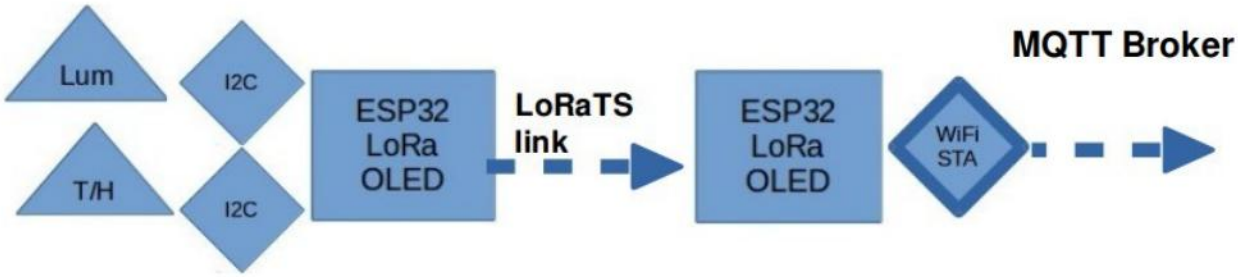

Figure 5.2 Une rchitecture IoT avec un terminal à 2 capteurs et une passerelle **LoRa-WiFi (MQTT Broker)**

5.2.1 L'émetteur des messages MQTT sur LoRa

```
#include <Arduino.h>
#include <SPI.h>
#include <LoRa.h>
#define SCK 18 // GPIO18 -- SX127x's SCK
#define MISO 19 // GPIO19 -- SX127x's MISO
#define MOSI 23 // GPIO23 -- SX127x's MOSI
#define SS 5 // GPIO05 -- SX127x's CS
#define RST 15 // GPIO15 -- SX127x's RESET
#define DI0 26 // GPIO26 -- SX127x's IRQ(Interrupt Request)
#define freq 434E6
#define sf 7
#define sb 125E3
union
{
uint8_t frame[64]; // trame avec octets
struct {
        char topic[32]; // 4 valeurs en virgule flottante
        char mess[32]; // 4 valeurs en virgule flottante
        } mqtt;
} sdp ; // paquet d'émission

void setup() {
Serial.begin(9600);
SPI.begin(SCK,MISO,MOSI,SS);
LoRa.setPins(SS,RST,DI0);
Serial.println();delay(100);Serial.println();
if (!LoRa.begin(freq)) { Serial.println("Starting LoRa failed!");while (1);}
Serial.println("Starting LoRa OK!");
LoRa.setSpreadingFactor(sf);
LoRa.setSignalBandwidth(sb);
}

int counter=0;

void loop() // la boucle de l'emetteur
{
Serial.print("New Packet:") ;
LoRa.beginPacket(); // start packet
strcpy(sdp.mqtt.topic,"/esp32/test");
scanf(sdp.mqtt.mess,"%d",counter);
LoRa.write(sdp.frame,64);
LoRa.endPacket();
Serial.println(counter);
counter++ ;
delay(2000);
}
```

5.2.2 La passerelle des messages MQTT sur LoRa vers WiFi et broker MQTT

```
#include <Arduino.h>

#include <SPI.h>
#include <LoRa.h>
#include <WiFi.h>
#include <MQTT.h>

#define NT 4 // max number of terminals
#define NTS 4 // max number of sensors per terminal

const char* ssid = "Livebox-08B0";
const char* pass = "G79ji6dtEptVTPWmZP";

const char* mqttServer = "broker.emqx.io";
//const char* mqttServer = "192.168.1.68";

#define SCK 18 // GPIO18 -- SX127x's SCK
#define MISO 19 // GPIO19 -- SX127x's MISO
#define MOSI 23 // GPIO23 -- SX127x's MOSI
#define SS 5 // GPIO05 -- SX127x's CS
#define RST 15 // GPIO15 -- SX127x's RESET
#define DI0 26 // GPIO26 -- SX127x's IRQ(Interrupt Request)
#define freq 434E6
#define sf 7
#define sb 125E3

union
{
uint8_t frame[64]; // trame avec octets
struct {
        char topic[32]; // 4 valeurs en virgule flottante
        char mess[32]; // 4 valeurs en virgule flottante
        } mqtt;
} rdp ; // paquet d'émission

QueueHandle_t msg_queue; // queues for data packets

void onReceive(int packetSize)
{
int rssi=0;
uint8_t buff[64];
if (packetSize == 0) return; // if there's no packet, return
int i=0;
if (packetSize==64)
{
while (LoRa.available()) { buff[i]=LoRa.read();i++; }
xQueueReset(msg_queue);
Serial.println("MQTT packet received");
xQueueSend(msg_queue, (void *)buff, 64);
}
}


WiFiClient net;
MQTTClient client;
unsigned long lastMillis = 0;

void connect() {
Serial.print("checking wifi...");
while (WiFi.status() != WL_CONNECTED) {
Serial.print(".");
delay(1000);
}
Serial.print("\nconnecting...");
while (!client.connect("IoT.GW5")) {
Serial.print("."); delay(1000);
}
Serial.println("\nIoT.GW1 - connected!");
```

```
client.subscribe("/esp32/sensors");
delay(100);
client.subscribe("/esp32/lorarssi");
}

void messageReceived(String &topic, String &payload) {
Serial.println("incoming: " + topic + " - " + payload);
// send LoRa message depending on topic
}

void setup() {
Serial.begin(9600);
WiFi.begin(ssid, pass);
client.begin(mqttServer, net);
client.onMessage(messageReceived);
connect();
SPI.begin(SCK,MISO,MOSI,SS);
LoRa.setPins(SS,RST,DI0);
Serial.println();delay(100);Serial.println();
if (!LoRa.begin(freq)) {
Serial.println("Starting LoRa failed!");
while (1);
}
msg_queue = xQueueCreate(4,64); // queue for 4 data MQTT packets
Serial.println("Starting LoRa OK!");
LoRa.setSpreadingFactor(sf);
LoRa.setSignalBandwidth(sb);
LoRa.onReceive(onReceive); // pour indiquer la fonction ISR
LoRa.receive(); // pour activer l'interruption (une fois)
}

void loop() {
char vbuff[15];
client.loop();
delay(10); // <- fixes some issues with WiFi stability
if (!client.connected()) { connect(); }
xQueueReceive(msg_queue,(void *)rdp.frame,6000); // 6s,default:portMAX_DELAY
sprintf(vbuff,"%2.2f",rdp.data[0]);
client.publish(rdp.mqtt.topic, rdp.mqtt.mess);
Serial.printf("published to%s:%s\n",rdp.mqtt.topic,rdp.mqtt.mess);
delay(6000);
}
```

A faire :

1. Tester le code de base
2. Ecrire le code complet pour la version de base avec le transfert du topic et du message.
3. Utiliser les capteurs SHT21 et BH1750 pour envoyer des données « réelles »