

Laboratoires IoT sur Pomme-Pi C3

SmartComputerLab

Contenu

0. Introduction.....	2
0.1 ESP32 Soc – une unité avancée pour les architectures IoT.....	3
0.2 Carte LOLIN PICO.....	3
0.3 Pomme-Pi C3 (IoT DevKit) une plate-forme de développement IoT.....	4
0.4 Architectures IoT – présentation par composants.....	5
0.4 L'installation de l'Arduino IDE.....	7
0.4.1 Installation des nouvelles cartes ESP32.....	7
Laboratoire 1 : Capteurs et afficheurs.....	8
1.1 Premier exemple – l'affichage des données.....	8
A faire:.....	8
1.1.1 Test – scan du bus I2C.....	9
1.2 Capture et affichage des valeurs.....	9
1.2.1 Capture de la température/humidité par SHT21.....	9
Code :.....	10
A faire:.....	10
1.2.2 Capture de la luminosité par BH1750.....	11
A faire:.....	11
1.2.3 Capture de la pression/température avec capteur BMP180.....	11
A faire :.....	12
1.2.4 Capture les coordonnées GPS.....	12
A faire :.....	13
1.2.5 Capture de présence avec un capteur PIR SR602.....	13
1.2.5.1 Le code.....	13
A faire :.....	14
Laboratoire 2 : Communication en WiFi et serveurs MQTT et ThingSpeak.....	14
2.1 Introduction.....	14
2.1.1 Un programme de test – scrutation du réseau WiFi.....	14
2.2.1 Mode STA – lecture d'une page WEB.....	16
A faire :.....	17
2.2.2 Mode softAP avec un serveur WEB.....	17
A faire :.....	18
2.2.3 Mode WiFi – STA et WiFiManager.....	18
Le code :.....	18
2.2.4 Envoi et réception des messages MQTT.....	19
2.2.4.1 Le code.....	19
A faire :.....	20
2.2.5 Envoi des données sur ThingSpeak avec la connexion par WiFiManager.....	20
2.2.5.1 Voici le code.....	21
2.2.6 Réception des données de ThingSpeak.....	22
A faire :.....	23
2.2.7 Obtenir la data/l'heure du serveur NTP.....	23
A faire :.....	24
Laboratoire 3 : Communication longue distance avec LoRa (Long Range).....	25
3.1 Introduction.....	25
3.1.1 Modulation LoRa.....	25
3.1.1.1 Fréquence LoRa en France.....	25
3.1.1.2 Facteur d'étalement en puissance de 2.....	25
3.1.1.3 Bande passante.....	25
3.1.2 Paquets LoRa.....	25
3.2 Premier exemple – émetteur et récepteur des paquets LoRa.....	27
3.2.1 Sender - code complet.....	28
3.2.2 Récepteur - code complet.....	30
A faire :.....	31
3.3 onReceive() – récepteur des paquets LoRa avec une interruption.....	31
3.3.1 Code - récepteur avec un Callback.....	32
A faire :.....	33
Laboratoire 4 : Développement des simples passerelles IoT (LoRa-WiFi).....	34
4.1 Passerelle LoRa-WiFi (MQTT).....	34
A faire :.....	36
4.2 Passerelle LoRa-WiFi (ThingSpeak).....	37
4.2.1 Le principe de fonctionnement.....	37
4.1.2 Les éléments du code.....	37
4.2.3 Code complet pour une passerelle des paquets en format de base.....	38
A faire :.....	40

Laboratoire 5 : Protocoles UDP, TCP et Web Sockets.....	41
5.1 Sockets UDP - envoi et réception de datagrammes.....	41
5.1.1 Architecture IoT : Client-Serveur (UDP-STA).....	41
5.1.1.1 Code - Client – émetteur UDP.....	42
5.1.1.2 Code - Serveur – récepteur UDP synchrone.....	43
5.1.2 Architecture IoT avec un serveur UDP synchrone et SoftAP.....	44
5.1.2.1 Code - Serveur UDP avec un softAP.....	44
5.1.2.2 Code - Client UDP synchrone pour le serveur avec SoftAP.....	45
A faire.....	45
5.2.1 Sockets TCP – établissement de connexions et envoi/réception de données (segments).....	46
5.2.2 Architecture IoT avec client-serveur TCP en mode STA.....	46
5.2.2.1 Code - Serveur TCP en mode STA.....	47
5.2.2.2 Code - Client TCP.....	47
5.2.3 Architecture IoT -TCP avec serveur en mode SoftAP.....	48
5.2.3.1 Code - Serveur TCP avec SoftAP.....	49
5.2.3.2 Client TCP pour un serveur avec SoftAP.....	49
A faire.....	50
5.3 Web Socket.....	51
5.3.1 Architecture IoT avec client-serveur sur WebSockets.....	52
5.3.1.1 Client.....	52
5.3.1.2 Server.....	52
5.3.2.1 Code - Client Websockets minimal.....	52
5.3.2.2 Serveur Websockets minimal.....	53
Laboratoire 6 : WiFi avancée.....	55
6.0 Introduction.....	55
2.1.1 Générateur des trames WiFi (balises).....	56
6.1.1.1 Code complet.....	57
A faire.....	58
6.2 Sniffer de WiFi.....	59
6.2.1 Les fonctions d'interface WiFi et les éléments du client renifleur.....	59
5.2.2 Code complet.....	60
A faire.....	62
Laboratoire 7 : WiFi directe (ESP-NOW) et WiFi Long Range.....	63
7.0 Introduction.....	63
7.1 Exemple simple serveur-client.....	64
7.1.1 L'émetteur.....	64
7.1.2 Le récepteur.....	65
7.1.3 Création d'un nœud de passerelle vers ThingSpeak.....	66
A faire.....	67
7.1.4 Création d'un nœud de passerelle vers un courtier MQTT.....	68
7.2 WiFi Long Range.....	70
7.2.1 Code du maître.....	70
7.2.1 Code de l'esclave.....	71
A faire :.....	73
Discussion :.....	73
Laboratoire 8 : Programmation OTA – Over The Air.....	74
8.0 Introduction.....	74
8.0.1 Flash memory partitions.....	74
8.0.2 Mécanisme OTA.....	74
8.1 Implémentation de l'OTA de base sur la carte ESP32C3.....	76
8.1.1 Implémentation - Basic OTA.....	76
8.1.2 Télécharger un nouveau code via WiFi.....	77
A faire :.....	79
8.2 OTA sur ESP32 avec serveur WEB.....	80
8.2.1 The starting program with Webserver.....	80
8.2.2 L'accès au serveur WEB.....	82
8.2.3 Télécharger un nouveau programme via WiFi(.bin).....	83
8.2.4 Générer un fichier .bin dans l'IDE Arduino.....	85
8.2.5 Télécharger un nouveau sketch en direct sur l'ESP32.....	85
A faire.....	85
8.3 Implémenter OTA avec la bibliothèque WebOTA.....	86
8.3.1 Code initial.....	86
A faire.....	87

Laboratoires IoT sur Pomme-Pi C3

SmartComputerLab

0. Introduction

Dans les laboratoires IoT nous allons mettre en œuvre plusieurs architectures IoT intégrant les terminaux (T), les passerelles (*gateways* - G), et les serveurs (S) IoT type **ThingSpeak ou Thingier.io**. Le développement sera réalisé sur les cartes **IoTDevKit** de **SmartComputerLab**.

Le kit de développement contient une carte de base "basecard" pour y accueillir une unité centrale et un ensemble de cartes d'extension pour les capteurs, les actionneurs et les modems supplémentaires. L'unité centrale est une carte équipée d'un **SoC ESP32** et d'un modem **LoRa (Long Range)**.

Le premier laboratoire permet de préparer l'environnement de travail et de tester l'utilisation des capteurs connectés sur le bus I2C (température/humidité/luminosité/pression) et d'un afficheur. Pour nos développements et nos expérimentations nous utilisons le **IDE Arduino** comme outils de programmation et de chargement des programmes dans la mémoire flash de l'unité centrale.

Le deuxième laboratoire est consacré à la prise en main du modem WiFi intégré dans la carte principale (ESP32-LOLIN). La communication WiFi en mode station et un client permet d'envoyer les données des capteurs vers un serveur **WEB**. Dans notre cas nous utilisons le serveur de type **ThingSpeak; (ThingSpeak.fr/com)**. Ces serveurs sont accessibles gratuitement, mais leur utilisation peut être limitée un temps (le cas de **ThingSpeak.com**).

Le troisième laboratoire permet d'expérimenter avec les liens à longue distance (1Km). Il s'agit de la technologie (et modulation) LoRa. Le modem LoRa est intégré sur la carte centrale. Nous allons envoyer les données d'un capteur géré par un terminal sur un lien LoRa vers une passerelle **LoRa-WiFi**. La passerelle retransmettra ces données vers un serveur **ThingSpeak**.

Le quatrième laboratoire permettra d'intégrer l'ensemble de liens WiFi et LoRa pour créer une application complète avec les terminaux, la *gateway* LoRa-WiFi et les serveurs **ThingSpeak**.

Après ces laboratoires préparatifs nous vous proposons **2 séries de laboratoires IoT** qui permettront d'approfondir les connaissances et le savoir faire dans le domaine des Architectures IoT.

La première série de 6 laboratoires est dédiée aux architectures IoT centrées sur la communication WiFi, Bluetooth, BLE et les modes de programmation possibles.

La deuxième série de 6 laboratoires introduit les architectures plus complexes avec l'intégration de LoRa, programmation faible consommation, programmation temps-réel et le développement des protocoles LoRa-WiFi pour les serveurs ThingSpeak et brokers MQTT.

Les multiples cartes d'extension avec les capteurs/actionneurs, modems, afficheurs permettront le développement des applications sur mesure.

0.1 ESP32 Soc – une unité avancée pour les architectures IoT

ESP32C3 est une unité de microcontrôleur avancée conçue pour le développement d'architectures IoT. Un ESP32 intègre un processeur RISC-V 32-bit fonctionnant à 180 MHz et plusieurs unités de traitement et de communication supplémentaires, notamment un processeur ULP (*Ultra Low Power*), des modems WiFi / Bluetooth /BLE et un ensemble de contrôleurs E/S pour bus série (UART, I2C, SPI). , ..). Ces blocs fonctionnels sont décrits ci-dessous dans la figure suivante.

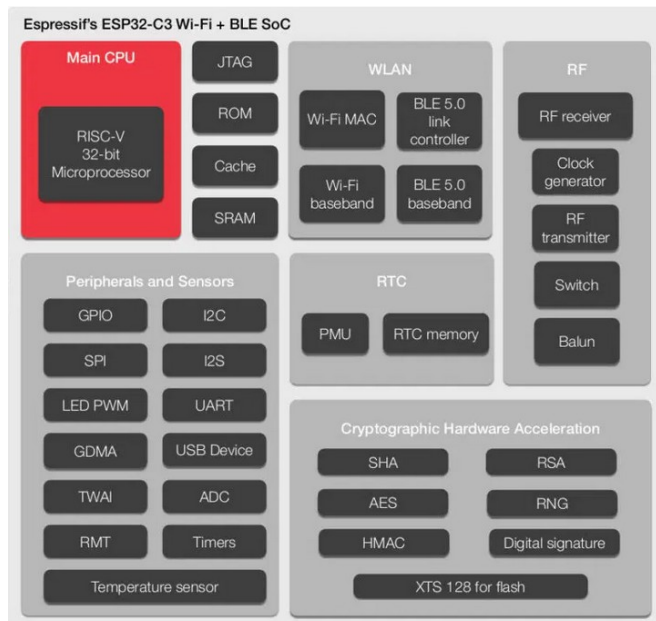


Figure 0.1 SoC ESP32C3 – architecture interne (image : Espressif)

0.2 Carte LOLIN PICO

De nos jours, les SoC ESP32 sont intégrés dans un certain nombre de cartes de développement qui incluent des circuits supplémentaires et des modems de communication. Notre choix est la carte **ESP32C3 PICO** qui intègre une interface avec les batteries LiPo (3V7)

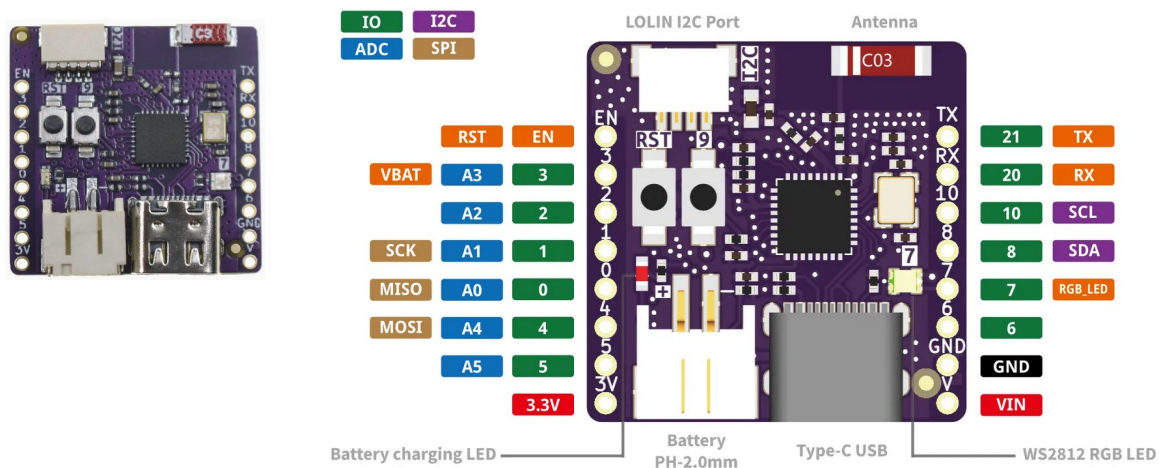


Figure 0.2 Carte MCU LOLIN PICO et son *pinout*

Comme nous pouvons le voir sur la figure ci-dessus, la carte expose 2x8 broches. Ces broches portent les bus I2C (SCL-10,SDA-8), SPI (SCK-1,MISO-0, MOSI-4), UART (TX-21, RX-20) plus les signaux de contrôle (NSS,RST,INT,..)

0.3 Pomme-Pi C3 (IoT DevKit) une plate-forme de développement IoT

Une intégration efficace de la carte LOLIN sélectionnée dans les architectures IoT nécessite l'utilisation d'une plate-forme de développement telle que IoT DevKit proposée par **SmartComputerLab**. L'**IoTDevKit** est composé d'une carte de base et de plusieurs cartes d'extension conçues pour l'utilisation efficace des bus de connexion et de tous les types de capteurs et d'actionneurs.

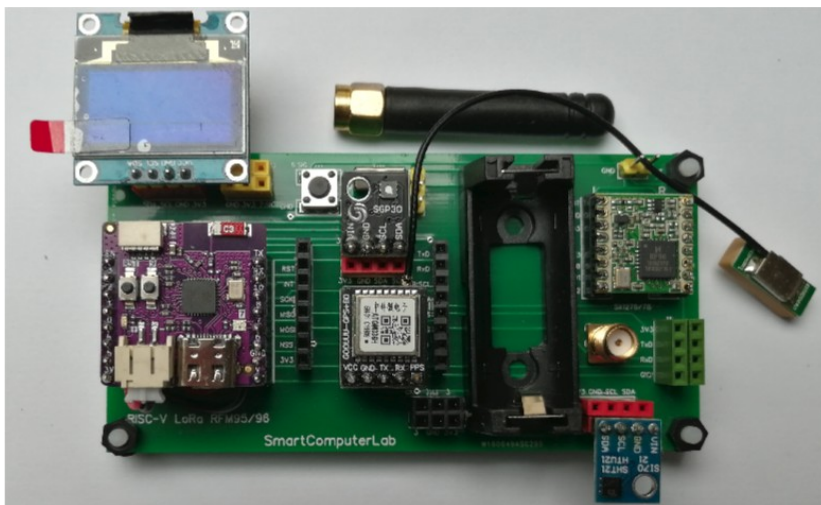


Figure 0.3 DevKit IoT - Pomme-Pi C3 - PICO

Sur la photo ci-dessus vous avez un DevKit à la base de la carte **ESP32C3 PICO** avec une batterie et un modem LoRa (module RFM96/5) intégré. Un écran OLED est connecté sur l'interface I2C (rouge), le capteur SHT21 est également connecté sur le bus partagé I2C. Un modem GPS est connecté sur le bus UART (vert).

La carte de base peut accueillir directement plusieurs types de capteurs ou modems de communication. Pour y connecter un ensemble plus complet de capteurs/modems/afficheurs on utilise les **cartes d'extension** en format **mini D1**.

La carte de base intègre un switch et un chevalier (**jump**) permet de connecter un multimètre ou un PPK2 pour y d'effectuer les mesures du courant.

En mode faible consommation (**deep_sleep**) le courant descend à quelques dizaines de micro-ampers.

Ci dessous quelques exemples de cartes d'extension.

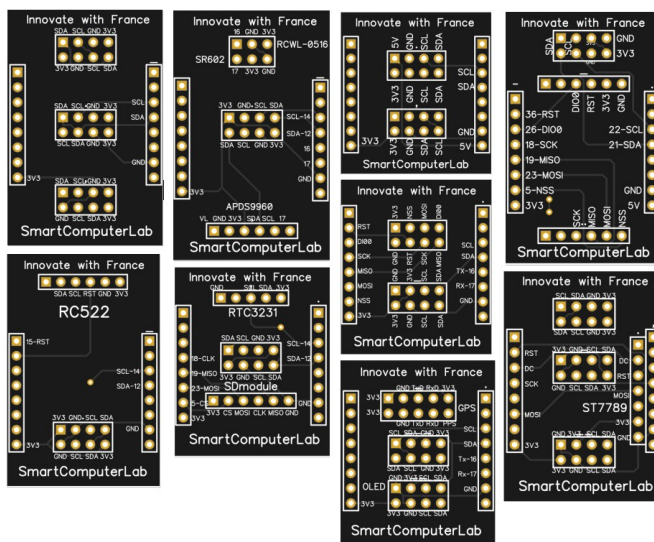


Figure 0.4 Cartes d'extension pour les différents composants IoT: capteurs, afficheurs, modems, ..

0.4 Architectures IoT – présentation par composants

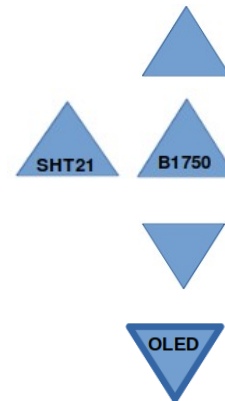
Les architectures IoT impliquent l'utilisation de multiples types de composants avec de multiples types de connexion et de liens de communication.

Pour rendre la complexité de ces architectures plus explicite et plus « visible » nous avons introduit les simple symboles pour les composants de base :

1. carte de base
2. capteurs
3. actionneurs incluant afficheurs
4. connexions type bus
5. liens radio et leurs points d'accès

.. Par exemple un capteur générique est représenté comme triangle :

Les capteurs instanciés comme triangles avec lo nom du capteur :



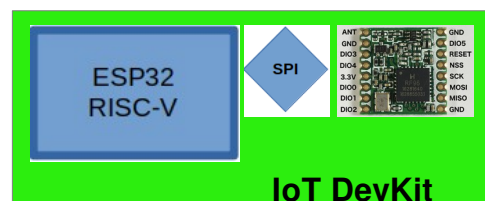
Les actionneurs génériques sont représentés comme triangles retournés :

Un actionneur instancié – écran OLED :

Les capteurs et les actionneurs sont connectés par le biais des bus : I2C, UART, SPI, .. Leur présentation graphique est sous la forme de losanges :



Le DevKit de base (ici Pomme-Pi C3) incluant leSoC ESP32C3 et le module du modem LoRa RFM95/6 est représenté par un rectangle :



Le SoC ESP32C3 intègre le modem WiFi fonctionnant en mode **STA** ou **SoftAP** :



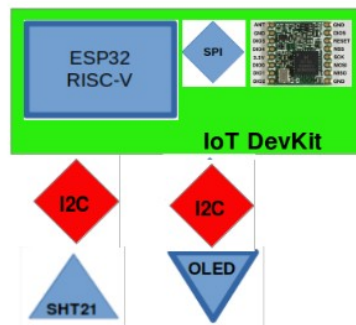
Symboliquement le modem Lora connecté par le bus SPI est représenté comme :



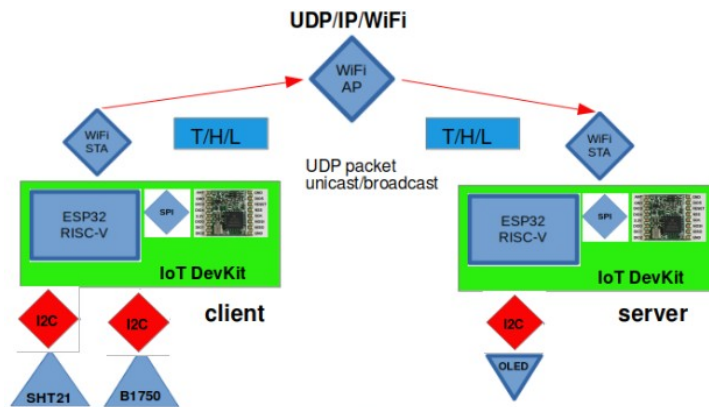
Finalement le serveurs IoT dans le cloud Internet sont symbolisés par:



Un exemple de l'architecture IoT très simple peut être le suivant :



Un autre beaucoup plus compliqué :



0.4 L'installation de l'Arduino IDE

Pour nos développements et nos expérimentations nous utilisons l'**IDE Arduino** comme outils de programmation et de chargement des programmes dans la mémoire flash de l'unité centrale. Afin de pouvoir développer le code pour les application nous avons besoin d'un environnement de travail comprenant; un PC, un OS type Ubuntu 20.04 LTS, l'environnement Arduino IDE, les outils de compilation/chargement pour les cartes ESP32 et les bibliothèque de contrôle pour les capteurs, actionneurs (par exemple **relais**), et les modems de communication.

Pour commencer mettez le système à jour par :

```
sudo apt-upgrade et sudo apt update
```

Ensuite il faut installer le dernier Arduino IDE à partir de <https://www.arduino.cc>

0.4.1 Installation des nouvelles cartes ESP32

Après son installation allez dans les **Preferences** et ajoutez deux **Boards Manager URLs** (séparées par une virgule) :

https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json

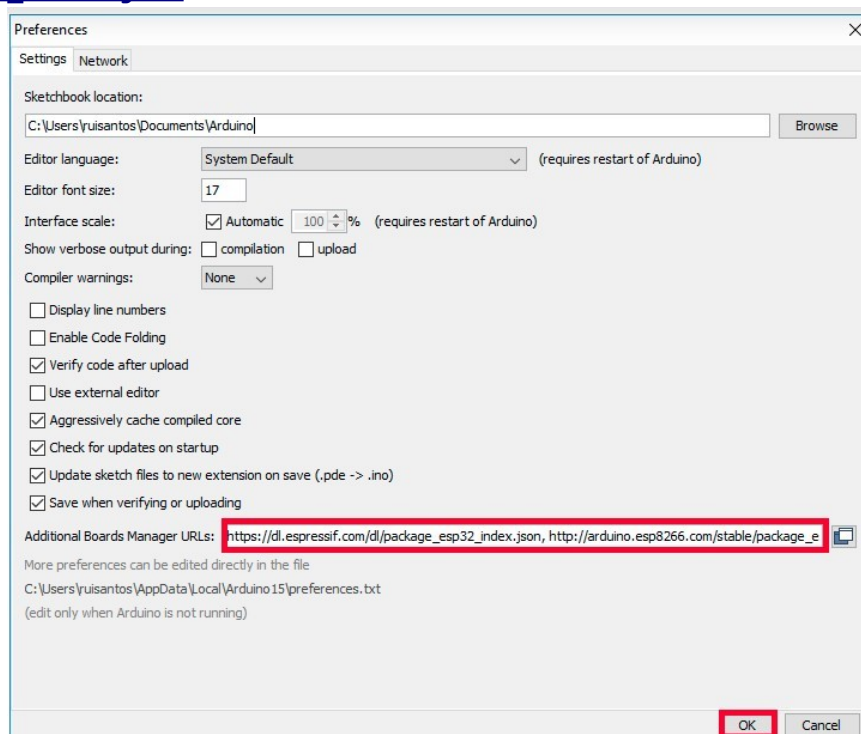


Figure 0.5 Ajout des outils Espressif (cross-compilateurs) dans l'environnement Arduino IDE

Laboratoire 1

1.1 Premier exemple – l'affichage des données

Dans cet exercice nous allons simplement afficher un titre et 4 valeurs numériques sur l'écran OLED ajouté à notre DevKit.

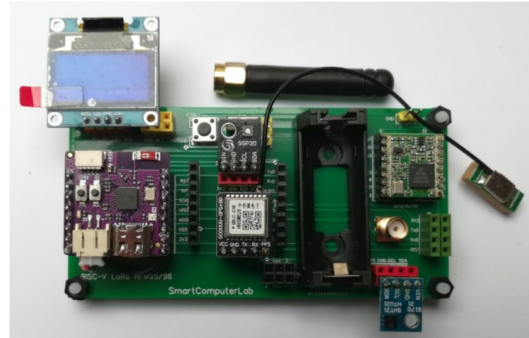


Figure 1.1 Ecran OLED connecté sur le bus I2C

Vous devez installer la bibliothèque **OLED** compatible avec ESP32. Cela peut être trouvée dans le gestionnaire de bibliothèque IDE Arduino.



Le code

```
#include <Wire.h>
#include "SSD1306Wire.h"
SSD1306Wire display(0x3c, 8, 10);    // SDA, SCL

void display_SSD1306(float d1,float d2,float d3,float d4)
{
    char buff[64];
    display.init();    //display.flipScreenVertically();
    display.setTextAlignment(TEXT_ALIGN_LEFT);
    display.setFont(ArialMT_Plain_16);
    display.drawString(0, 0, "IoT DevKit");
    display.setFont(ArialMT_Plain_10);
    sprintf(buff,"d1: %2.2f, d2: %2.2f\nd3: %2.2f, d4: %2.2f",d1,d2,d3,d4);
    display.drawString(0, 22, buff);
    display.drawString(20, 52, "SmartComputerLab");
    display.display();
}

void setup() {
    Serial.begin(9600);
    Wire.begin(8,10); // SDA, SCL
    Serial.println("Wire started");
}

float v1=0.0,v2=0.0,v3=0.0,v4=0.0;

void loop()
{
    Serial.println("in the loop");
    //
    delay(2000);
}
```

A faire:

Compléter, compiler, et charger ce programme.

1.1.1 Test – scan du bus I2C

Avant de continuer l'exemple suivant nous allons utiliser le code suivant pour détecter la présence de capteurs/actionneurs sur le bus I2C.

```
#include <Wire.h>

void I2Cscan()
{
  byte address; int nDevices; delay(200);
  Serial.println("Scanning...");
  nDevices = 0;
  for(address = 1; address < 127; address++)
  {
    Wire.beginTransmission(address);
    if(! Wire.endTransmission()) // une adresse active trouvee
    {
      Serial.print("I2C device found at address 0x");
      if (address<16) Serial.print("0");
      Serial.print(address,HEX); Serial.println(" !");
      nDevices++;
    }
  }
  if (nDevices == 0)
    Serial.println("No I2C devices found\n");
  else
    Serial.println("done\n");
}

void setup()
{
  Serial.begin(9600);
  Wire.begin(8,10,400000); // SDA, SCL avec 400 kHz
  delay(1000); Serial.println();Serial.println();
  I2Cscan(); delay(1000);
}

void loop(){ }
```

Scanning...
I2C device found at address 0x3C !
I2C device found at address 0x40 !
done

1.2 Capture et affichage des valeurs

1.2.1 Capture de la température/humidité par SHT21

Dans cet exercice nous allons lire les valeurs fournies par le capteur Température/Humidité SHT21 et afficher les 2 valeurs sur l'écran OLED ajouté sur le DevKit. Le capteur SHT21 doit également être connecté sur le bus I2C.

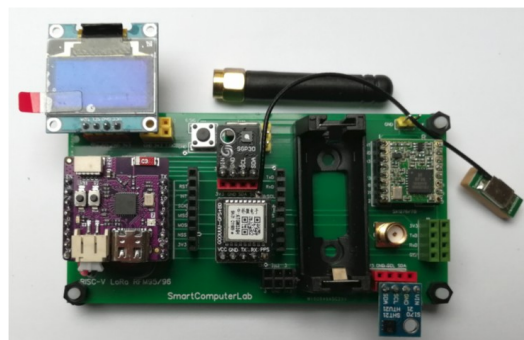
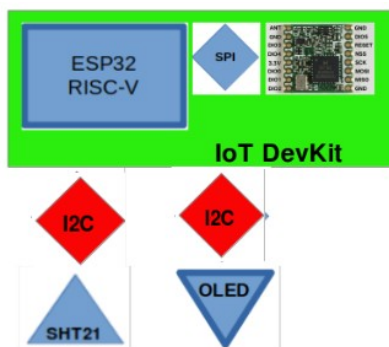


Figure 1.2. IoT DevKit avec un capteur de température/humidité SHT21 et l'écran OLED.

Code :

Avant de commencer le codage il faut télécharger et installer la bibliothèque **SHT2x.h** compatible avec notre carte. Elle se trouve dans le **gestionnaire des bibliothèques**.



```
#include <Wire.h>
#include "SSD1306Wire.h"
SSD1306Wire display(0x3c, 8, 10); // SDA, SCL
#include "SHT2x.h"
SHT2x sht;
float stab[4]= {0.0,0.0,0.0,0.0};

void get_SHT21()
{
    sht.read();delay(100);
    stab[0]=sht.getTemperature();
    delay(100);
    stab[1]=sht.getHumidity();
    Serial.printf("T:%2.2f, H:%2.2f\n",stab[0],stab[1]);
}

void display_SSD1306(float d1,float d2,float d3,float d4)
{
    char buff[64];
    display.init();//display.flipScreenVertically();
    display.setTextAlignment(TEXT_ALIGN_LEFT);
    display.setFont(ArialMT_Plain_16);
    display.drawString(0, 0, "ETN - IoT DevKit");
    display.setFont(ArialMT_Plain_10);
    sprintf(buff,"T: %2.2f, H: %2.2f\n d3: %2.2f, d4: %2.2f",d1,d2,d3,d4);
    display.drawString(0, 22, buff);
    display.drawString(20, 52, "SmartComputerLab");
    display.display();
}

void setup() {
    Serial.begin(9600);
    Serial.println();
    sht.begin(8, 10);
    delay(200);
}

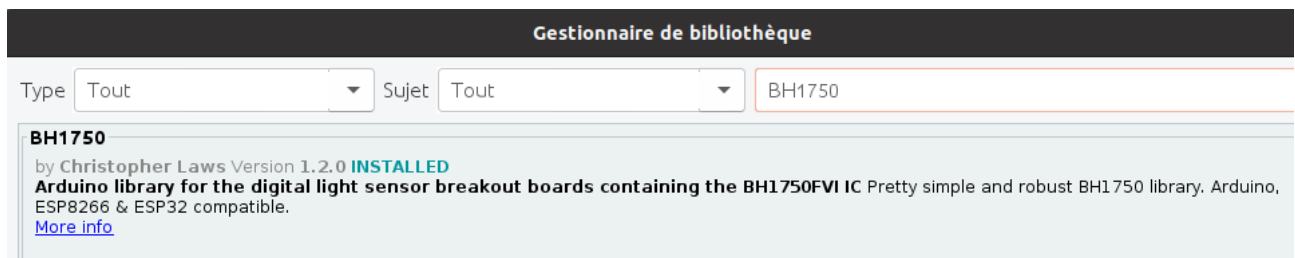
void loop()
{
    Serial.println("in the loop");
    // a completer
    delay(2000);
}
```

A faire:

1. Compléter, compiler, et charger ce programme.

1.2.2 Capture de la luminosité par BH1750

Dans cet exercice nous utilisons le capteur de la luminosité BH1750



```
#include <Wire.h>
#include <BH1750.h>
BH1750 lightMeter;

void setup(){
  Wire.begin(8,10);    // SDA, SCL
  Serial.begin(9600);
  lightMeter.begin();
  Serial.println("Running...");
  delay(1000);
}

void loop() {
  uint16_t lux = lightMeter.readLightLevel();
  delay(1000);
  Serial.print("Light: ");
  Serial.print(lux);
  Serial.println(" lx");
  delay(1000);
}
```

A faire:

1. Compléter, compiler, et charger ce programme.
2. Ajouter l'affichage sur l'écran OLED

1.2.3 Capture de la pression/température avec capteur BMP180

Le capteur BMP180 permet de capter la pression atmosphérique et la température. Sa précision est seulement de +/- 100 Pa et +/-1.0C.

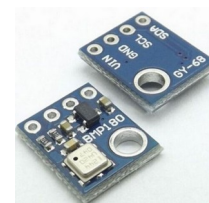


Figure 1.6. Capteur de pression/température BMP180

La valeur standard de la pression atmosphérique est :

$$101\,325\text{ Pa} = 1,013\,25\text{ bar} = 1\text{ atm}$$

```
#include <Wire.h>
#include <Adafruit_BMP085.h>
Adafruit_BMP085 bmp;

void setup() {
  Serial.begin(9600);
  Wire.begin(12,14);
  bmp.begin();
}

void loop() {
  Serial.print("Temperature = ");
  Serial.print(bmp.readTemperature()); Serial.println(" *C");
  Serial.print("Pressure = "); Serial.print(bmp.readPressure());
  Serial.println(" Pa");
  // Calculate altitude assuming 'standard' barometric
```

```

// pressure of 1013.25 millibar = 101325 Pascal
Serial.print("Altitude = ");
Serial.print(bmp.readAltitude());
Serial.println(" meters");
// you can get a more precise measurement of altitude
// if you know the current sea level pressure which will
// vary with weather and such. If it is 1015 millibars
// that is equal to 101500 Pascals.
Serial.print("Real altitude = ");
Serial.print(bmp.readAltitude(104300)); // a trouver la pression au niveau de la mere
Serial.println(" meters");
Serial.println();
delay(500);
}

```

Résultat :

```

Pressure = 103574 Pa
Altitude = -185.58 meters
Real altitude = 58.89 meters

```

```

Temperature = 23.10 *C
Pressure = 103571 Pa
Altitude = -185.42 meters
Real altitude = 58.97 meters

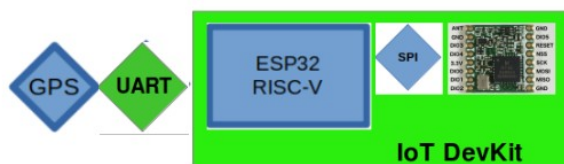
```

A faire :

1. Complétez le programme ci-dessus afin d'afficher les données de la température , pression atmosphérique et l'altitude sur l'écran OLED

a

1.2.4 Capture les coordonnées GPS



Le modem GPS (**ATGM33**) peut être utilisé avec un simple code de lecture en continu sur le bus **UART** (interface vert)

```

#include <SoftwareSerial.h>
SoftwareSerial ss(7,6);

void loop()
{
  while (ss.available() > 0)
    Serial.write(ss.read());
}

void setup()
{
  Serial.begin(115200);
  ss.begin(9600); //start softserial for GPS at defined baud rate
}

```

Voici l'extrait de l'affichage sur le terminal :

```

..
GNLL,4713.00643,N,00141.61912,W,152023.000,A,A*5B
$GPGSA,A,3,02,29,31,,,,,,,,,3.7,2.3,2.9*37
$BDGSA,A,3,23,24,34,,,,,,,,,3.7,2.3,2.9*2D
$GPGSV,3,1,12,02,48,122,22,04,06,334,,11,14,036,,12,15,094,*7B
$GPGSV,3,2,12,18,21,166,,20,07,060,,22,28,248,,25,42,094,20*79
$GPGSV,3,3,12,26,34,289,04,29,77,067,21,31,67,265,24,32,07,214,*72
$BDGSV,1,1,03,23,11,306,23,24,56,135,28,34,50,195,21*5A
$GNRMC,152023.000,A,4713.00643,N,00141.61912,W,1.04,217.33,040223,,A*6A
$GNVTG,217.33,T,,M,1.04,N,1.92,K,A*28
$GNZDA,152023.000,04,02,2023,00,00*4A
$GPTXT,01,01,01,ANTENNA OK*35

```

```

$GNGGA,152024.000,4713.00599,N,00141.61945,W,1,06,2.3,57.3,M,0.0,M,,*59
$GNGLL,4713.00599,N,00141.61945,W,152024.000,A,A*5A
$GPGSA,A,3,02,29,31,,,,,,,,,3.7,2.3,2.9*37
$BDGSA,A,3,23,24,34,,,,,,,,,3.7,2.3,2.9*2D
$GPGSV,3,1,12,02,48,122,22,04,06,334,,11,14,036,,12,15,094,*7B
$GPGSV,3,2,12,18,21,166,,20,07,060,,22,28,248,,25,42,094,20*79
$GPGSV,3,3,12,26,34,289,05,29,77,067,21,31,67,265,24,32,07,214,*73
$BDGSV,1,1,03,23,11,306,23,24,56,135,28,34,50,195,21*5A
$GNRMC,152024.000,A,4713.00599,N,

```

A faire :

Dans ces trois lignes vous retrouverez la **date**, l'**heure** et la **position** du modem.
C'est à vous de les décoder !

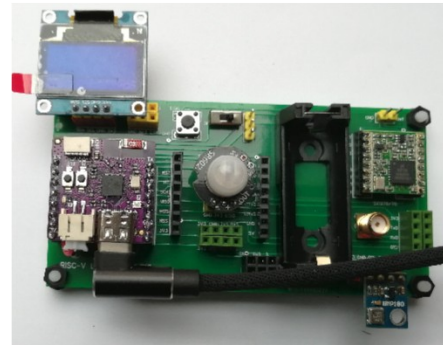
```

$GNRMC,152023.000,A,4713.00643,N,00141.61912,W,1.04,217.33,040223,,A*6A
$GNVTG,217.33,T,,M,1.04,N,1.92,K,A*28
$GNZDA,152023.000,04,02,2023,00,00*4A

```

1.2.5 Capture de présence avec un capteur PIR SR602

Dans l'exemple suivant nous utilisons un capteur de type **PIR - SR602** qui permet de détecter la présence d'une personne en mouvement. Il s'agit d'un capteur simple qui signale l'événement par le basculement de son signal de sortie. Le capteur est connecté sur l'interface simple (jaune) comme sur la photo ci-dessous.



1.2.5.1 Le code

```

#define PIR 6 // signal simple
bool MOTION_DETECTED = false;

void pinChanged()
{
    MOTION_DETECTED = true;
}
void setup()
{
    Serial.begin(9600);
    attachInterrupt(PIR, pinChanged, RISING);
}

int counter=0;

void loop()
{
    int i=0;
    if(MOTION_DETECTED)
    {
        Serial.println("Motion detected.");
        delay(1000);counter++;
        MOTION_DETECTED = false;
        Serial.println(counter);
    }
}

```

Affichage sur le terminal :

```

1
Motion detected.
2
Motion detected.
3
Motion detected.
4
Motion detected.
5

```

A faire :

1. Complétez le programme ci-dessus afin d'afficher la valeur du compteur de détection des mouvements sur l'écran OLED

Laboratoire 2

Communication en WiFi et serveurs MQTT et ThingSpeak

2.1 Introduction

Dans ce laboratoire nous allons nous intéresser aux moyens de la **communication** et de la **présentation** (**stockage**) des résultats. Etant donné que le SoC ESP32C3 intègre les modems de WiFi et de BLE nous allons étudier la communication par **WiFi**.

Principalement nous avons deux modes de fonctionnement **WiFi** – mode station **STA**, et mode point d'accès **softAP**. Dans le mode **STA** nous pouvons connecter notre carte à un point d'accès WiFi (qui peut être notre smartphone). Dans le mode **AP** nous créons un point d'accès sur la carte ESP32C3. Ce point d'accès peut être utilisé pour effectuer une configuration de la carte, par exemple en lui fournissant le nom du point d'accès (**ssid**) et le mot de passe pour ce point d'accès.

Un troisième mode appelé **ESP-NOW** permet de communiquer entre les cartes directement (sans un Point d'Accès), donc plus rapidement et à plus grande distance, par le biais des trames physiques **MAC**.

Une fois la communication WiFi est opérationnelle nous pouvons choisir un des multiples protocoles pour transmettre nos données : **UDP**, **TCP**, **HTTP/TCP**, ..

Par exemple la carte peut fonctionner comme **client** ou serveur **WEB**. Dans une application fonctionnant comme un client **WEB** nous allons contacter un serveur externe type **ThingSpeak** pour y envoyer et stocker nos données.

Bien sur nous aurons besoin des bibliothèques relatives à ces protocoles.

2.1.1 Un programme de test – scrutation du réseau WiFi

Pour commencer notre étude de la communication WiFi essayons un exemple permettant de scruter notre environnement dans la recherche de point d'accès visibles par notre modem.

```
#include "WiFi.h"

void setup()
{
    Serial.begin(115200);
    // Set WiFi to station mode and disconnect from an AP if it was previously connected.
    WiFi.mode(WIFI_STA);
    WiFi.disconnect();
    delay(100);
    Serial.println("Setup done");
}

void loop()
{
    Serial.println("Scan start");
    // WiFi.scanNetworks will return the number of networks found.
    int n = WiFi.scanNetworks();
    Serial.println("Scan done");
    if (n == 0) {
        Serial.println("no networks found");
    } else {
        Serial.print(n);
        Serial.println(" networks found");
        Serial.println("Nr | SSID | RSSI | CH | Encryption");
        for (int i = 0; i < n; ++i) {
            // Print SSID and RSSI for each network found
            Serial.printf("%2d", i + 1);
            Serial.print(" | ");
            Serial.printf("%-32.32s", WiFi.SSID(i).c_str());
            Serial.print(" | ");
            Serial.printf("%4d", WiFi.RSSI(i));
            Serial.print(" | ");
            Serial.printf("%2d", WiFi.channel(i));
            Serial.print(" | ");
            switch (WiFi.encryptionType(i))
```

```

    {
        case WIFI_AUTH_OPEN:
            Serial.print("open");
            break;
        case WIFI_AUTH_WEP:
            Serial.print("WEP");
            break;
        case WIFI_AUTH_WPA_PSK:
            Serial.print("WPA");
            break;
        case WIFI_AUTH_WPA2_PSK:
            Serial.print("WPA2");
            break;
        case WIFI_AUTH_WPA_WPA2_PSK:
            Serial.print("WPA+WPA2");
            break;
        case WIFI_AUTH_WPA2_ENTERPRISE:
            Serial.print("WPA2-EAP");
            break;
        case WIFI_AUTH_WPA3_PSK:
            Serial.print("WPA3");
            break;
        case WIFI_AUTH_WPA2_WPA3_PSK:
            Serial.print("WPA2+WPA3");
            break;
        case WIFI_AUTH_WAPI_PSK:
            Serial.print("WAPI");
            break;
        default:
            Serial.print("unknown");
    }
    Serial.println();
    delay(10);
}
}
Serial.println("");
// Delete the scan result to free memory for code below.
WiFi.scanDelete();
// Wait a bit before scanning again.
delay(5000);
}

```

Résultat affiché sur le terminal :

```

Scan start
Scan done
13 networks found

```

Nr	SSID	RSSI	CH	Encryption
1	DIRECT-G8M2070 Series	-55	11	WPA2
2	VAIO-MQ35AL	-66	5	WPA+WPA2
3	Livebox-08B0	-68	1	WPA2
4	PIX-LINK-2.4G	-69	11	open
5	Livebox-3D36	-83	1	WPA2
6	SFR_EOB0	-86	6	WPA
7	SFR WiFi FON	-87	6	open
8	freebox_ODTMTH	-87	11	WPA
9	FreeWifi_secure	-87	11	WPA2-EAP
10	SFR WiFi Mobile	-88	6	WPA2-EAP
11	freebox_DZHECV	-88	11	WPA+WPA2
12	FreeWifi_secure	-88	11	WPA2-EAP
13	Livebox-C6E0	-89	1	WPA2

2.2 Mode WiFi – STA et WiFiManager

La carte ESP32 permet de fonctionner en mode Point d'Accès qui permet de déployer un simple serveur WEB avec une page de configuration des paramètres sur la carte ESP32.

Parmi ces paramètres il y a la possibilité de proposer (et enregistrer) le nom d'un point d'accès et son mot de passe.



2.2.1 Mode STA – lecture d'une page WEB

L'exemple suivant montre l'utilisation du mode STA pour connecter la carte au WiFi puis avec la librairie `HTTPClient.h` pour se connecter par le biais du protocole TCP/HTTP à un serveur WEB et lire sa page initiale.

`WiFiMulti` permet de trouver un point d'accès disponible parmi plusieurs accessibles.

```
#include <WiFi.h>
#include <WiFiMulti.h>
#include <HTTPClient.h>

WiFiMulti wifiMulti;

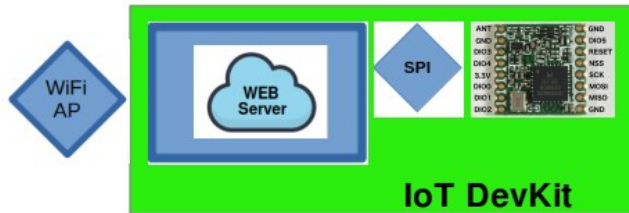
void setup()
{
    Serial.begin(115200);
    for(uint8_t t = 4; t > 0; t--) {
        Serial.printf("[SETUP] WAIT %d...\n", t);
        Serial.flush();
        delay(1000);
    }
    wifiMulti.addAP("Livebox-08B0", "G79ji6dtEptVTPWmZP");
}

void loop() {
    // wait for WiFi connection
    if((wifiMulti.run() == WL_CONNECTED)) {
        HTTPClient http;
        Serial.print("[HTTP] begin...\n");
        http.begin("http://www.smartcomputerlab.org"); //HTTP
        Serial.print("[HTTP] GET...\n");
        // start connection and send HTTP header
        int httpCode = http.GET();
        // httpCode will be negative on error
        if(httpCode > 0) {
            // HTTP header has been send and Server response header has been handled
            Serial.printf("[HTTP] GET... code: %d\n", httpCode);
            // file found at server
            if(httpCode == HTTP_CODE_OK) {
                String payload = http.getString();
                Serial.println(payload);
            }
        } else {
            Serial.printf("[HTTP] GET failed, error: %s\n", http.errorToString(httpCode).c_str());
        }
        http.end();
    }
    delay(5000);
}
```

A faire :

1. Complétez le programme ci-dessus et afin d'afficher la valeur de l'adresse IP reçue après la connexion sur l'écran OLED.

2.2.2 Mode softAP avec un serveur WEB



```
#include <WiFi.h>
#include <WiFiClient.h>
#include <WiFiAP.h>
#include <Adafruit_NeoPixel.h>
#define PIN 7 // Set the GPIO pin
#define NUMPIXELS 1
Adafruit_NeoPixel pixels(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);
const char *ssid = "ESP32AP";
const char *password = "";
WiFiServer server(80);

void setup()
{
  Serial.begin(115200);
  Serial.println();
  pixels.begin();
  Serial.println("Configuring access point...");
  WiFi.softAP(ssid, password); // open if password ""
  IPAddress myIP = WiFi.softAPIP();
  Serial.print("AP IP address: ");
  Serial.println(myIP);
  server.begin();
  Serial.println("Server started");
}

void loop() {
  WiFiClient client = server.available(); // listen for incoming clients
  if (client) { // if you get a client,
    Serial.println("New Client."); // print a message out the serial port
    pixels.setPixelColor(0, pixels.Color(0, 150, 0));
    pixels.show();
    String currentLine = ""; // make a String to hold incoming data from the client
    while (client.connected()) { // loop while the client's connected
      if (client.available()) { // if there's bytes to read from the client,
        char c = client.read(); // read a byte, then
        Serial.write(c); // print it out the serial monitor
        if (c == '\n') { // if the byte is a newline character
          // if the current line is blank, you got two newline characters in a row.
          // that's the end of the client HTTP request, so send a response:
          if (currentLine.length() == 0) {
            client.println("HTTP/1.1 200 OK");
            client.println("Content-type:text/html");
            client.println();
            // the content of the HTTP response follows the header:
            client.print("Click <a href=\"/H\">here</a> to turn ON the LED.<br>");
            client.print("Click <a href=\"/L\">here</a> to turn OFF the LED.<br>");
            // The HTTP response ends with another blank line:
            client.println();
            // break out of the while loop:
            break;
          } else { // if you got a newline, then clear currentLine:
            currentLine = "";
          }
        } else if (c != '\r') { // if you got anything else but a carriage return character,
          currentLine += c; // add it to the end of the currentLine
        }
      }
    }
    // Check to see if the client request was "GET /H" or "GET /L":
    if (currentLine.endsWith("GET /H")) {
      pixels.setPixelColor(0, pixels.Color(150, 0, 0));
      pixels.show(); // GET /H turns the LED on
    }
    if (currentLine.endsWith("GET /L")) {
      pixels.setPixelColor(0, pixels.Color(0, 0, 150));
    }
  }
}
```

```

        pixels.show();
    }
}
// close the connection:
client.stop();
pixels.setPixelColor(0, pixels.Color(0, 0, 150));
pixels.show();
Serial.println("Client Disconnected.");
}
}

```

A faire :

Complétez le programme ci-dessus et afin d'afficher la valeur de l'adresse IP du serveur sur l'écran OLED.

2.2.3 Mode WiFi – STA et WiFiManager

La carte ESP32C3 permet de fonctionner en mode **softAP** - Point d'Accès qui permet de déployer un simple serveur WEB avec une page de configuration des paramètres sur la carte ESP32C3.

Parmi ces paramètres il y a la possibilité de proposer (et enregistrer) le nom d'un point d'accès et son mot de passe.

WiFiManager est une fonction qui réalise ce type d'opérations.

Voici notre programme de test pour l'utilisation du **WiFiManager**.



Le code :

```

#include <WiFiManager.h> // https://github.com/tzapu/WiFiManager

void setup() {
    WiFi.mode(WIFI_STA); // set mode to STA+AP
    Serial.begin(115200);
    WiFiManager wm;
    //reset settings - wipe credentials for testing
    wm.resetSettings(); // à tester , puis à commenter
    // Automatically connect using saved credentials,
    bool res;
    res = wm.autoConnect("ESP32C3AP", NULL); // no password
    if(!res) {
        Serial.println("Failed to connect");
        // ESP.restart();
    }
    else {
        //if you get here you have connected to the WiFi
        Serial.println("connected...yeey :)");
    }
}

void loop()
{
    IPAddress ip = WiFi.localIP();
    Serial.print("IP Address: ");
    Serial.println(ip);
    delay(4000);
}

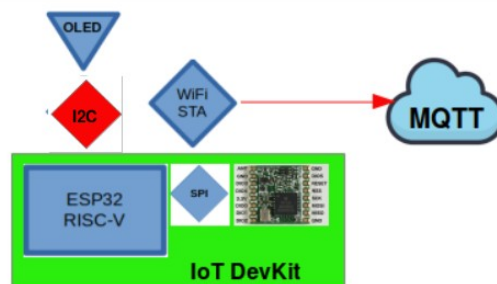
```

L'affichage sur le terminal – à noter l'adresse IP : 192.168.4.1 associée au serveur WEB sur notre carte. Cette adresse doit être utilisée pour la connexion avec le smartphone sur **softAP- ESP32C3AP**. Notez que l'adresse obtenu après la connexion au point d'accès Internet est : 192.168.1.65.

```
*wm:resetSettings
*wm:SETTINGS ERASED
*wm:AutoConnect
*wm:No wifi saved, skipping
*wm:AutoConnect: FAILED
*wm:StartAP with SSID: ESP32C3AP
*wm:AP IP address: 192.168.4.1
*wm:Starting Web Portal
*wm:13 networks found
*wm:Connecting to NEW AP: Livebox-08B0
*wm:connectTimeout not set, ESP waitForConnectResult...
*wm:Connect to new AP [SUCCESS]
*wm:Got IP Address:
*wm:192.168.1.65
*wm:config portal exiting
connected...yeey :)
IP Address: 192.168.1.65
IP Address: 192.168.1.65
IP Address: 192.168.1.65
IP Address: 192.168.1.65
..
```

2.2.4 Envoi et réception des messages MQTT

Un serveur-broker MQTT reçoit et retransmet les simples messages MQTT. Pour recevoir les messages postés (**published**) sur un sujet (**topic**) il faut souscrire (**subscribe**) votre code sur le serveur MQTT recevant les messages sur ce **topic**.



2.2.4.1 Le code

```
#include <WiFi.h>
#include <MQTT.h>
const char ssid[] = "Livebox-08B0";
const char pass[] = "G79ji6dtEptVTPWmZP";
// Attention: ici vous pouvez proposer les credentials sur votre smartphone
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("ESP32C3", "public", "public")) {
    Serial.print(".");
    delay(1000);
  }
  Serial.println("\nconnected!");
  client.subscribe("/esp32c3/test");
  // client.unsubscribe("/hello");
}

void messageReceived(String &topic, String &payload) {
  Serial.println("incoming: " + topic + " - " + payload);
  // Note: Do not use the client in the callback to publish, subscribe or
  // unsubscribe as it may cause deadlocks when other things arrive while
  // sending and receiving acknowledgments. Instead, change a global variable,
  // or push to a queue and handle it in the loop after calling `client.loop()`.
}
```

```

void setup() {
  Serial.begin(115200);
  WiFi.begin(ssid, pass);
  client.begin("broker.emqx.io",net);
  client.onMessage(messageReceived);
  connect();
}

void loop() {
  client.loop();
  delay(10); // <- fixes some issues with WiFi stability
  if (!client.connected()) {
    connect();
  }
  // publish a message roughly every second.
  if (millis() - lastMillis > 4000) {
    lastMillis = millis();
    client.publish("/esp32c3/test", "world");
  }
}

```

L'affichage sur le terminal :

```

..
incoming: /esp32c3/test - world
incoming: /esp32c3/test - world
incoming: /esp32c3/test - world
incoming: /esp32c3/test - world
incoming: /esp32c3/test - world
incoming: /esp32c3/test - world
..

```

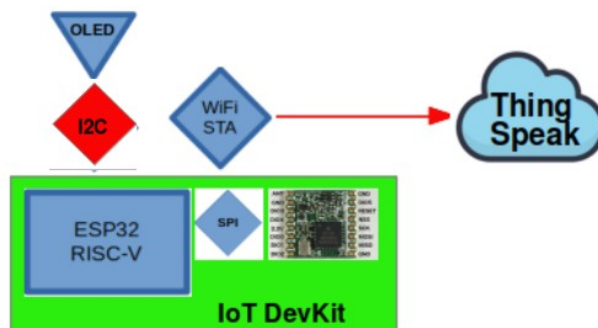
Notez que vous pouvez suivre les messages envoyés sur une application typée **MyMQTT** disponible sur le smartphone.

A faire :

1. Complétez le programme ci-dessus et afin d'afficher le nom du **topic** et le contenu du message reçu sur l'écran OLED.

2.2.5 Envoi des données sur ThingSpeak avec la connexion par WiFiManager

Dans l'exemple ci-dessous nous allons nous connecter au serveur **ThingSpeak.com**. Pour commencer il faut ouvrir un compte gratuit. Ce compte est limité par l'envoi d'un message toutes les 15 secondes, et par le nombre d'enregistrements limité à 256. Voici exemple de création de **channels** sur ce compte. Chaque **channel** peut accueillir jusqu'à 8 **fields**.



ThingSpeak™ Channels Apps Devices Support			
My Channels			
New Channel		Search by tag	
Name	Created	Updated	
SmartIoTBox-0 smartiotlabs,zero	2021-10-16	2023-01-24 15:04	
Private Public Settings Sharing API Keys Data Import / Export			
SmartIoTBox-1 smartiotlabs,one	2022-01-05	2022-04-07 14:00	
Private Public Settings Sharing API Keys Data Import / Export			
SmartIoTBox-2 smartiotlabs-2	2022-04-07	2023-01-26 17:08	

2.2.5.1 Voici le code

```
#include <WiFiManager.h>
#include "ThingSpeak.h"
unsigned long myChannelNumber =1626377;
const char *myWriteAPIKey="3IN09682SQX3PT4Z" ;
const char *myReadAPIKey="9JVTP8ZHTB9G4TT" ;

WiFiClient client;

void setup() {
  WiFi.mode(WIFI_STA);
  Serial.begin(115200);
  WiFiManager wm;
  // wm.resetSettings();
  bool res;
  res = wm.autoConnect("ESP32C3AP",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");
}

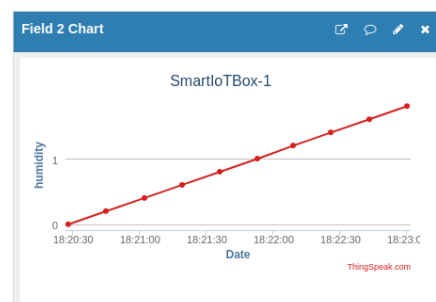
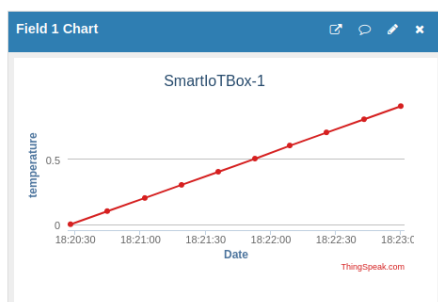
float temperature=0.0,humidity=0.0;

void loop() {
  Serial.println("Fields update");
  ThingSpeak.setField(1, temperature);
  ThingSpeak.setField(2, humidity);
  ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
  delay(16000);
  temperature+=0.1;
  humidity+=0.2;
}
```

Le résultat de réception de nos données sur le serveur **ThingSpeak.com**.

Channel Stats

Created: [about a year ago](#)
Last entry: [less than a minute ago](#)
Entries: 10



2.2.6 Réception des données de ThingSpeak

Voici un exemple complet dans lequel on envoi plus on récupère les mêmes données sur le serveur **ThingSpeak.com**.

```
#include <WiFiManager.h>
#include "ThingSpeak.h"

unsigned long myChannelNumber =1626377;
const char *myWriteAPIKey="3IN09682SQX3PT4Z" ;
const char *myReadAPIKey="9JVTP8ZHTB9G4TT" ;
```

```

WiFiClient client;

void setup() {
  WiFi.mode(WIFI_STA);
  Serial.begin(9600);
  WiFiManager wm;
  //reset settings - wipe credentials for testing
  // 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");
}
float temperature=0.0,humidity=0.0;
float tem,hum;

void loop() {
  Serial.println("Fields update");
  ThingSpeak.setField(1, temperature);
  ThingSpeak.setField(2, humidity);
  ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
  delay(16000);
  temperature+=0.1;
  humidity+=0.2;
  tem =ThingSpeak.readFloatField(myChannelNumber,1);
  // Note that if channel is private you need to add the read key
  Serial.print("Last temperature:");
  Serial.println(tem);
  delay(1000);
  hum =ThingSpeak.readFloatField(myChannelNumber,2,myReadAPIKey);
  Serial.print("Last humidity:");
  Serial.println(hum);
  delay(16000);
}

```

L'affichage sur le terminal :

```

Fields update
Fields update
*wm:AutoConnect
*wm:Connecting to SAVED AP: Livebox-08B0
*wm:connectTimeout not set, ESP waitForConnectResult...
*wm:AutoConnect: SUCCESS
*wm:STA IP Address: 192.168.1.65
connected...yeey :)
ThingSpeak begin
Fields update
Last temperature:0.00
Last humidity:0.00
Fields update
Last temperature:0.10
Last humidity:0.20
Fields update
Last temperature:0.20
Last humidity:0.40
Fields update
Last temperature:0.30
Last humidity:0.60

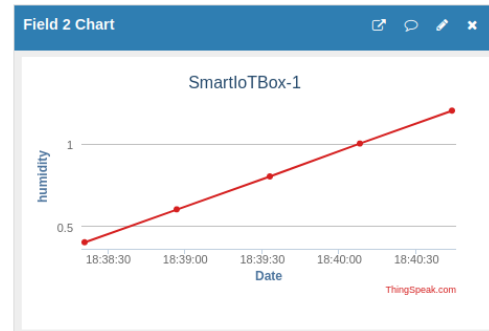
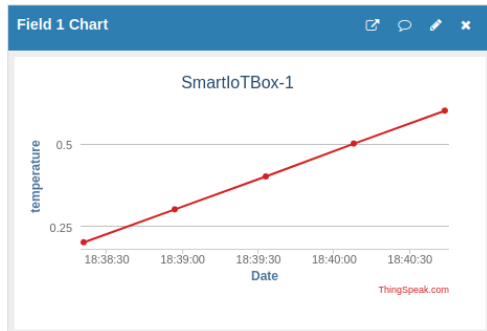
```

Channel Stats

Created: [about a year ago](#)

Last entry: [less than a minute ago](#)

Entries: 5



A faire :

1. Créez un compte gratuit sur [ThingSpeak.com](#)
2. Complétez le programme ci-dessus et afin d'afficher la valeur du message reçu sur l'écran OLED.

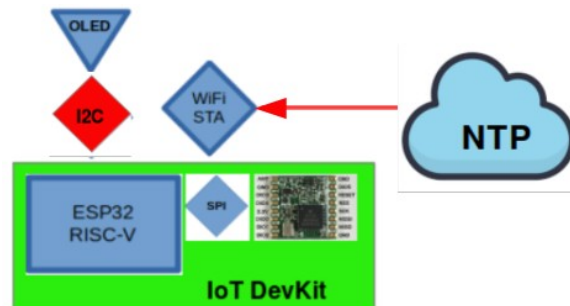
2.2.7 Obtenir la data/l'heure du serveur NTP

Le serveur du temps Internet **NTP – Network Time Protocol** donne la date/heure précise et permet de synchroniser les tâches sur la carte.

Voici le code :

```
#include <WiFiManager.h>
#include "time.h"
#include <Adafruit_NeoPixel.h>
#define NUMPIXELS 12
#define PIN 6 // build-in LED - 7, RGB LED ring - 6
Adafruit_NeoPixel pixels(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);
const char* ntpServer = "pool.ntp.org";
const long  gmtoffset_sec = 0;
const int   daylightOffset_sec = 3600;

void setup(){
  WiFi.mode(WIFI_STA);
  Serial.begin(115200);
  pixels.begin();
  for(int i=0;i<12;i++)pixels.setPixelColor(i, pixels.Color(0, 150, 0));
  pixels.show();
  WiFiManager wm;
  // wm.resetSettings();
  bool res;
  res = wm.autoConnect("ESP32C3AP",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 :)");
  }
  // Init and get the time
  configTime(gmtoffset_sec, daylightOffset_sec, ntpServer);
  printLocalTime();
  //disconnect WiFi as it's no longer needed
  WiFi.disconnect(true);
```




```

WiFi.mode(WIFI_OFF);
for(int i=0;i<12;i++)pixels.setPixelColor(i, pixels.Color(0, 0, 0));
pixels.show();
}

void loop(){
  delay(1000);
  printLocalTime();
}
/* tm structure:
   tm_sec: seconds after the minute;
   tm_min: minutes after the hour;
   tm_hour: hours since midnight;
   tm_mday: day of the month;
   tm_year: years since 1900;
   tm_wday: days since Sunday;
   tm_yday: days since January 1;
   tm_isdst: Daylight Saving Time flag;
*/

void printLocalTime(){
  struct tm timeinfo;
  if(!getLocalTime(&timeinfo)){
    Serial.println("Failed to obtain time");
    return;
  }
  Serial.println(&timeinfo, "%A, %B %d %Y %H:%M:%S");
  Serial.print("Day of week: ");
  Serial.println(&timeinfo, "%A");
  Serial.print("Month: ");
  Serial.println(&timeinfo, "%B");
  Serial.print("Day of Month: ");
  Serial.println(&timeinfo, "%d");
  Serial.print("Year: ");
  Serial.println(&timeinfo, "%Y");
  Serial.print("Hour: ");
  Serial.println(&timeinfo, "%H");
  Serial.print("Hour (12 hour format): ");
  Serial.println(&timeinfo, "%I");
  pixels.setPixelColor(timeinfo.tm_hour, pixels.Color(150, 0, 0));
  pixels.show();
  Serial.print("Minute: ");
  Serial.println(&timeinfo, "%M");
  Serial.print("Second: ");
  Serial.println(&timeinfo, "%S");
  Serial.println();
}

```

Notez que la communication avec le serveur NTP se termine après la première réception de données. La suite de synchronisation est réalisée par l'horloge interne du circuit.

A faire :

1. Analysez, compilez et exécutez le code
2. Complétez le code pour y ajouter affichage des minutes et des secondes sur l'anneau LED

Laboratoire 3

Communication longue distance avec LoRa (*Long Range*)

3.1 Introduction

Dans ce laboratoire nous allons nous intéresser à la technologie de transmission **Long Range** essentielle pour la communication entre les objets.

Longe Range ou **LoRa** permet de transmettre les données à la distance d'un kilomètre ou plus avec les débits allant de quelques centaines de bits par seconde aux quelques dizaines de kilobits (100bit – 75Kbit).

3.1.1 Modulation LoRa

Modulation LoRa a trois paramètres :

- **freq** – *frequency* ou fréquence de la porteuse de 868 à 870 MHz,
- **sf** – *spreading factor* ou étalement du spectre ou le nombre de modulations par un bit envoyé (64-4096 exprimé en puissances de 2 – 7 à 12)
- **sb** – *signal bandwidth* ou bande passante du signal (31250 Hz à 500KHz)

Par défaut on utilise : **freq**=868MHz, **sf**=7, et **sb**=125KHz

Notre carte Heltec Wifi Lora 32 intègre un modem LoRa. Il est connecté avec le SoC ESP32 par le biais d'un bus **SPI**.

La paramétrisation du modem LoRa est facilité par la bibliothèque **LoRa.h** à inclure dans vos programmes.

```
#include <LoRa.h>
```

3.1.1.1 Fréquence LoRa en France

```
LoRa.setFrequency(868E6);
```

définit la **fréquence** du modem sur **868,0 MHz**.

3.1.1.2 Facteur d'étalement en puissance de 2

Le **facteur d'étalement (sf)** pour la modulation LoRa varie de 2^7 à 2^{12} Il indique combien de **chirps** sont utilisés pour transporter un bit d'information.

```
LoRa.setSpreadingFactor(8);
```

définit le facteur d'étalement à 10 ou 1024 signaux par bit.

3.1.1.3 Bande passante

La **largeur de bande de signal (sb)** pour la modulation LoRa varie de **31,25 kHz à 500 kHz** (31,25, 62,5, 125,0, 250,0, 500,0). Plus la bande passante du signal est élevée, plus le débit est élevé.

```
LoRa.setSignalBandwidth(125E3);
```

définit la largeur de bande du signal à 125 KHz.

3.1.2 Paquets LoRa

La **classe LoRa** est activée avec le constructeur **begin()** qui prend éventuellement un argument, la fréquence.

```
int begin(longue freq);
```

Pour créer un paquet LoRa, nous commençons par:

```
int beginPacket();
```

pour terminer la construction du paquet que nous utilisons

```
int endPacket();
```

Les données sont **envoyées** par les fonctions:

```
virtual size_t write (uint8_t byte);  
virtual size_t write (const uint8_t * buffer, taille_taille);
```

Dans notre cas, nous utilisons fréquemment la deuxième fonction avec le tampon contenant 4 données en virgule flottante.

La réception **en continue** des paquets LoRa peut être effectuée via la méthode `parsePacket()`.

```
int parsePacket();
```

Si le paquet est présent dans le tampon de réception du modem LoRa, cette méthode renvoie une valeur entière égale à la taille du paquet (charge utile de trame).

Pour lire efficacement le paquet du tampon, nous utilisons les méthodes :

```
LoRa.available() et LoRa.read()
```

Les paquets LoRa sont envoyés comme chaînes des octets. Ces chaînes doivent porter différents types de données.

Afin d'accommoder ces différents formats nous utilisons les **union**.

Par exemple pour formater un paquet avec 4 valeurs en virgule flottante nous définissons une union type :

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

Pour les paquets plus évolués nous avons besoin d'ajouter les en-têtes. Voici un exemple d'une union avec les paquets structurés.

```
union tspack  
{  
    uint8_t frame[20];  
    struct packet  
    {  
        uint8_t head[4]; // 4 bytes  
        float sensor[4]; // 16 bytes  
    } pack;  
} sdp, rdp; // send, recv data packet
```

3.2 Premier exemple – émetteur et récepteur des paquets LoRa

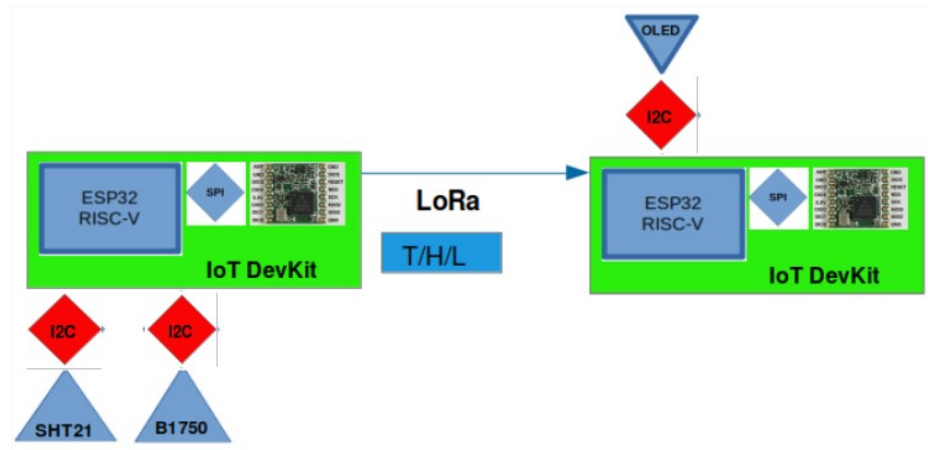
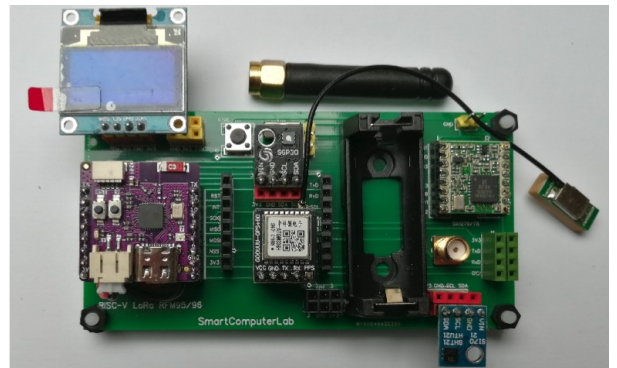


Figure 3.1 Module LoRa – RFM95/6 intégré sur la carte avec une connexion sur bus SPI

La partie initiale du code est **la même pour l'émetteur et pour le récepteur**. Dans cette partie nous insérons les bibliothèques de connexion par le bus SPI (**SPI.h**) et de communication avec le modem LoRa (**LoRa.h**). Nous proposons les paramètres par défaut pour le lien radio LoRa.

```
#include <SPI.h>
#include <LoRa.h>
#define SS      5      // NSS
#define RST      3      // RST
#define DI0      2      // INTR - DIO0
#define SCK      1      // CLK
#define MISO      0      // MISO
#define MOSI      4      // MOSI
#define BAND      868E6

int sf=7;
long sbw=125E3;
```



Nous définissons également le format d'un paquet dans la trame LoRa avec 4 champs **float** pour les données des capteurs.

```
union
{
    uint8_t frame[64];
    char mess[64];
    float sensor[8];
} sdp;
```

Dans la fonction **setup()** nous initialisons les connexions avec le modem LoRa et les paramètres radio.

```
void setup() {
    Serial.begin(115200);
    SPI.begin(SCK, MISO, MOSI, SS); // SCK, MISO, MOSI, SS
    LoRa.setPins(SS, RST, DI0);
    Serial.begin(9600);
    delay(1000);
    Serial.println();Serial.println();
    Serial.println("Starting LoRa Sender");
    if (!LoRa.begin(BAND)) {
        Serial.println("Starting LoRa failed!");
        while (1);
    }
    else
    {
        Serial.println("Starting LoRa ok!");
        disp("start", "LoRa OK", "end");
        LoRa.setSpreadingFactor(sf);
        LoRa.setSignalBandwidth(sbw);
    }
}
```

Enfin dans la fonction `loop()` nous préparons les données à envoyer dans le paquet LoRa de façon cyclique , une fois toutes les 5 secondes.

```
int counter=0;

void loop() {
char buff[32];
  Serial.print("Sending packet: ");
  Serial.println(counter);
  // send packet
  LoRa.beginPacket();
  LoRa.write(sdp.frame, 64);
  LoRa.endPacket();
  sdp.sensor[0]+=0.1;
  sdp.sensor[1]+=0.2;
  sdp.sensor[2]+=0.3;
  sdp.sensor[3]+=0.4;
  counter++;
  delay(5000);
}
```

Le programme du récepteur contient les mêmes déclarations et la fonction de `setup()` que le programme de l'émetteur. Le code ci dessous illustre seulement la partie différente.

```
void loop() {
char buff[32];
char currentid[64];
char receivedText[64]; int i=0;
  // try to parse packet
  int packetSize = LoRa.parsePacket();
  if (packetSize==64) {
    // received a packet
    Serial.print("Received packet ");
    // read packet
    while (LoRa.available()) {
      rdp.frame[i] = LoRa.read();i++;
    }
    Serial.println(rdp.sensor[0]);
    Serial.println(rdp.sensor[1]);
    Serial.println(rdp.sensor[2]);
    Serial.println(rdp.sensor[3]);
    // print RSSI of packet
    Serial.print("' with RSSI ");
    Serial.println(LoRa.packetRssi());
  }
}
```

3.2.1 Sender - code complet

```
#include <LoRa.h>
#include <Wire.h> // Only needed for Arduino 1.6.5 and earlier
#include "SSD1306Wire.h" // legacy: #include "SSD1306.h"
// Initialize the OLED display using Arduino Wire:
SSD1306Wire display(0x3c, 8, 10); // ADDRESS, SDA, SCL

String receivedText;
String receivedRssi;
// with LoRa modem RFM95 and green RFM board - int and RST to solder
#define SS 5 // 26 // D0 - to NSS
#define RST 3 //16 // D4 - RST
#define DI0 2 // D8 - INTR
#define SCK 1 // D5 - CLK
#define MISO 0 // D6 - MISO
#define MOSI 4 // D7 - MOSI
#define BAND 868E6
int sf=7;
long sbw=125E3;
char dbuff[24];

union
{
  uint8_t frame[64];
  char mess[64];
  float sensor[8];
} sdp;
```

```

void disp(char *t1, char *t2, char *t3)
{
    display.clear();
    display.setFont(ArialMT_Plain_10);
    display.setTextAlignment(TEXT_ALIGN_LEFT);
    display.drawString(0, 0, t1);
    display.drawString(0, 16, t2);
    display.drawString(0, 32, t3);
    display.display();
}

void setup() {
    Serial.begin(9600);
    SPI.begin(SCK, MISO, MOSI, SS); // SCK, MISO, MOSI, SS
    LoRa.setPins(SS, RST, DIO);
    Serial.begin(9600);
    delay(1000);
    Serial.println();Serial.println();
    Serial.println("Starting LoRa Sender");
    // Initialising the UI will init the display too.
    display.init();
    display.flipScreenVertically();
    if (!LoRa.begin(BAND)) {
        disp("start", "LoRa failed", "end");
        Serial.println("Starting LoRa failed!");
        while (1);
    }
    else
    {
        Serial.println("Starting LoRa ok!");
        disp("start", "LoRa OK", "end");
        LoRa.setSpreadingFactor(sf);
        LoRa.setSignalBandwidth(sbw);
    }
    sdp.sensor[0]=0.0;
    sdp.sensor[1]=0.0;
    sdp.sensor[2]=0.0;
    sdp.sensor[3]=0.0;
}

int counter=0;

void loop() {
    char buff[32];
    Serial.print("Sending packet: ");
    Serial.println(counter);
    sprintf(buff, "Count: %d", counter);
    disp("Sending packet",buff," ");
    // send packet
    LoRa.beginPacket();
    LoRa.write(sdp.frame, 64);
    LoRa.endPacket();
    sdp.sensor[0]+=0.1;
    sdp.sensor[1]+=0.2;
    sdp.sensor[2]+=0.3;
    sdp.sensor[3]+=0.4;
    counter++;
    delay(5000);
}

```

3.2.2 Récepteur - code complet

```

#include <LoRa.h>
#include <Wire.h> // Only needed for Arduino 1.6.5 and earlier
#include "SSD1306Wire.h" // legacy: #include "SSD1306.h"
// Initialize the OLED display using Arduino Wire:
SSD1306Wire display(0x3c, 8, 10); // ADDRESS, SDA, SCL

// with LoRa modem RFM95 and green RFM board - int and RST to solder

#define SS      5 // 26 // D0 - to NSS
#define RST     3 //16 // D4 - RST
#define DIO     2 // // D8 - INTR
#define SCK     1 // // D5 - CLK

```

```

#define MISO    0        // D6 - MISO
#define MOSI    4        // D7 - MOSI
#define BAND    868E6
int sf=7;
long sbw=125E3;
union
{
    uint8_t frame[64];
    char mess[64];
    float sensor[8];
} sdp,rdp;

void disp(char *t1, char *t2, char *t3, char *t4)
{
    display.clear();
    display.setFont(ArialMT_Plain_10);
    display.setTextAlignment(TEXT_ALIGN_LEFT);
    display.drawString(0, 0, t1);
    display.drawString(0, 16, t2);
    display.drawString(0, 32, t3);
    display.drawString(0, 48, t4);
    display.display();
}

void setup() {
    Serial.begin(115200);
    SPI.begin(SCK, MISO, MOSI, SS); // SCK, MISO, MOSI, SS
    LoRa.setPins(SS, RST, DI0);
    Serial.begin(9600);
    delay(1000);
    Serial.println();Serial.println();
    Serial.println("Starting LoRa Sender");
    display.init();
    display.flipScreenVertically();
    if (!LoRa.begin(BAND)) {Serial.println("Starting LoRa failed!"); while (1);}
    else
    {
        Serial.println("Starting LoRa ok!");
        LoRa.setSpreadingFactor(sf);
        LoRa.setSignalBandwidth(sbw);
    }
}

void loop() {
    int i=0, rssi;
    char bt1[32],bt2[32],bt3[32];
    // try to parse packet
    int packetSize = LoRa.parsePacket();
    if (packetSize==64) {
        // received a packet
        Serial.print("Received packet ");
        // read packet
        while (LoRa.available()) {
            rdp.frame[i] = LoRa.read();i++;
        }
        rssi=LoRa.packetRssi();
        Serial.println(rdp.sensor[0]);Serial.println(rdp.sensor[1]);
        Serial.println(rdp.sensor[2]);Serial.println(rdp.sensor[3]);
        sprintf(bt1,"S1:%2.2f, S2:%2.2f",rdp.sensor[0],rdp.sensor[1]);
        sprintf(bt2,"S3:%2.2f, S4:%2.2f",rdp.sensor[2],rdp.sensor[3]);
        sprintf(bt3,"RSSI:%d",rssi);
        disp("LoRa packet",bt1,bt2,bt3);
        Serial.print("' with RSSI ");
        Serial.println(rssi);
    }
}

```

A faire :

1. Tester le code ci-dessus et analyser la force du signal en réception. Par exemple **-60 dB** signifie que le signal reçu est 10^6 plus faible que le signal d'émission.
2. Modifier les paramètres LoRa par exemple **freq=8695E5**, **sf=10**, **sb=250E3** et tester le résultats de transmission.
3. Dans le *sender* envoyer les données captées sur un ou deux capteurs.

3.3 onReceive() – récepteur des paquets LoRa avec une interruption

Les interruptions permettent de ne pas exécuter en boucle l'opération d'attente d'une nouvelle trame dans le tampon du récepteur. Une fonction-tâche séparée est réveillée automatiquement après l'arrivée d'une nouvelle trame.

Cette fonction, souvent appelée `onReceive()` doit être marquée dans le `setup()` par :

```
void setup() {
  Serial.begin(115200);
  SPI.begin(SCK, MISO, MOSI, SS); // SCK, MISO, MOSI, SS
  LoRa.setPins(SS, RST, DI0);
  Serial.begin(9600);
  delay(1000);
  Serial.println();Serial.println();
  Serial.println("Starting LoRa Sender");
  display.init();
  display.flipScreenVertically();
  if (!LoRa.begin(BAND)) {
    Serial.println("Starting LoRa failed!");
    while (1);
  }
  else
  {
    Serial.println("Starting LoRa ok!");
    LoRa.setSpreadingFactor(sf);
    LoRa.setSignalBandwidth(sbw);
    LoRa.onReceive(onReceive); // pour indiquer la fonction ISR
    LoRa.receive();           // pour activer l'interruption (une fois)
    // puis après chaque émission
  }
}
```

La fonction **ISR** (*Interruption Service Routine*) ci-dessous permet de **capter** une nouvelle trame.

```
void onReceive(int packetSize)
{
  int rssi=0;
  union
  {
    uint8_t frame[64];
    char mess[64];
    float sensor[8];
  } sdp,rdp;

  if (packetSize == 0) return; // if there's no packet, return
  int i=0;
  if (packetSize==64)
  {
    while (LoRa.available())
    {
      rdp.frame[i]=LoRa.read();i++;
    }
    rssi=LoRa.packetRssi();
    Serial.printf("Received packet:%2.2f,%2.2f\n",rdp.sensor[0],rdp.sensor[1]);
    Serial.printf("RSSI=%d\n",rssi);
  }
}
```

3.3.1 Code - récepteur avec un Callback

```
#include <LoRa.h>
#include <Wire.h> // Only needed for Arduino 1.6.5 and earlier
#include "SSD1306Wire.h" // legacy: #include "SSD1306.h"
// Initialize the OLED display using Arduino Wire:
SSD1306Wire display(0x3c, 8, 10); // ADDRESS, SDA, SCL
// with LoRa modem RFM95 and green RFM board - int and RST to solder
#define SS 5 // 26 // D0 - to NSS
#define RST 3 //16 // D4 - RST
#define DI0 2 // D8 - INTR
#define SCK 1 // D5 - CLK
#define MISO 0 // D6 - MISO
#define MOSI 4 // D7 - MOSI
#define BAND 868E6
```



```

int sf=7;
long sbw=125E3;
union
{
    uint8_t frame[64];
    char mess[64];
    float sensor[8];
} sdp, rdp;

void disp(char *t1, char *t2, char *t3, char *t4)
{
    display.clear();
    display.setFont(ArialMT_Plain_10);
    display.setTextAlignment(TEXT_ALIGN_LEFT);
    display.drawString(0, 0, t1);
    display.drawString(0, 16, t2);
    display.drawString(0, 32, t3);
    display.drawString(0, 48, t4);
    display.display();
}

void setup() {
    Serial.begin(115200);
    SPI.begin(SCK, MISO, MOSI, SS); // SCK, MISO, MOSI, SS
    LoRa.setPins(SS, RST, DI0);
    Serial.begin(9600);
    delay(1000);
    Serial.println();Serial.println();
    Serial.println("Starting LoRa Sender");
    display.init();
    display.flipScreenVertically();
    if (!LoRa.begin(BAND)) {
        Serial.println("Starting LoRa failed!");
        while (1);
    }
    else
    {
        Serial.println("Starting LoRa ok!");
        LoRa.setSpreadingFactor(sf);
        LoRa.setSignalBandwidth(sbw);
        LoRa.onReceive(onReceive); // pour indiquer la fonction ISR
        LoRa.receive();           // pour activer l'interruption (une fois)
        // puis après chaque émission
    }
}

void onReceive(int packetSize)
{
    int rssi=0;
    union
    {
        uint8_t frame[64];
        char mess[64];
        float sensor[8];
    } sdp, rdp;

    if (packetSize == 0) return; // if there's no packet, return
    int i=0;
    if (packetSize==64)
    {
        while (LoRa.available())
        {
            rdp.frame[i]=LoRa.read();i++;
        }
        rssi=LoRa.packetRssi();
        Serial.printf("Received packet:%2.2f,%2.2f\n", rdp.sensor[0], rdp.sensor[1]);
        Serial.printf("RSSI=%d\n", rssi);
    }
}

void loop()
{
    Serial.println("in the loop");
    delay(5000);
}

```

L'affichage pendant l'exécution :

```
in the loop
Received packet:36.10,72.20
RSSI=-46
in the loop
Received packet:36.20,72.40
RSSI=-46
in the loop
Received packet:36.30,72.60
RSSI=-46
in the loop
Received packet:36.40,72.80
RSSI=-46
in the loop
Received packet:36.50,73.00
RSSI=-45
```

A faire :

1. Tester le code ci-dessus.
2. Afficher les données reçues sur l'écran OLED
3. Transférer les données reçues dans les variables externes (problème?).

Laboratoire 4

Développement des simples passerelles IoT (LoRa-WiFi)

Dans ce laboratoire nous allons développer des architectures intégrant plusieurs dispositifs essentiels pour la création d'un système IoT complet. Le dispositif central sera la passerelle (*gateway*) entre les liens LoRa et la communication par WiFi vers un serveur IoT.

4.1 Passerelle LoRa-WiFi (MQTT)

Voici notre architecture IoT qui consiste en deux nœuds, un terminal (T) et une passerelle (G). Les deux nœuds sont implémentés avec le même type de IoT DevKit intégrant le SoC ESP32C3 et le modem LoRa RFM95.

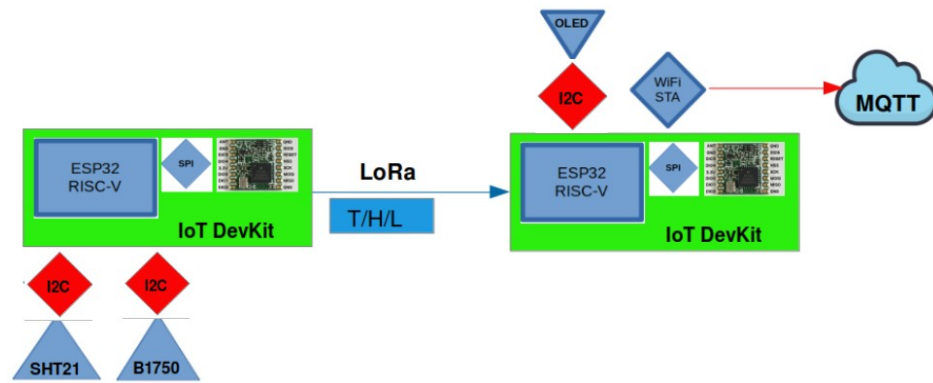


Fig 4.1 Architecture IoT avec un terminal LoRa et une passerelle LoRa-WiFi (MQTT).

Le code du terminal est le même que celui de *sender* préparé dans le laboratoire 3. Par contre l'implémentation de la passerelle nécessite plus des fonctionnalités.

```
#include <LoRa.h>
#include <Wire.h>

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

const char ssid[] = "Livebox-08B0";
const char pass[] = "G79ji6dtEptVTPWmZP";
// Attention: ici vous pouvez proposer les credentials sur votre smartphone

WiFiClient net;
MQTTClient client;

#include "SSD1306Wire.h" // legacy: #include "SSD1306.h"
SSD1306Wire display(0x3c, 8, 10); // ADDRESS, SDA, SCL

#define SS      5 // 26 // D0 - to NSS
#define RST     3 // 16 // D4 - RST
#define DI0     2 // D8 - INTR

#define SCK     1 // D5 - CLK
#define MISO    0 // D6 - MISO
#define MOSI    4 // D7 - MOSI
#define BAND    868E6
int sf=7;
long sbw=125E3;

union
{
  uint8_t frame[64];
  char mess[64];
  float sensor[8];
} sdp, rdp;
```

```

void disp(char *t1, char *t2, char *t3, char *t4)
{
    display.clear();
    display.setFont(ArialMT_Plain_10);
    display.setTextAlignment(TEXT_ALIGN_LEFT);
    display.drawString(0, 0, t1);
    display.drawString(0, 16, t2);
    display.drawString(0, 32, t3);
    display.drawString(0, 48, t4);
    display.display();
}

void connect() {
    Serial.print("checking wifi...");
    while (WiFi.status() != WL_CONNECTED) {
        Serial.print(".");
        delay(1000);
    }
    Serial.print("\nconnecting...");
    while (!client.connect("ESP32C3", "public", "public")) {
        Serial.print(".");
        delay(1000);
    }
    Serial.println("\nconnected!");
    client.subscribe("/esp32c3/test");
}

void messageReceived(String &topic, String &payload) {
    Serial.println("incoming: " + topic + " - " + payload);
}

void setup() {
    Serial.begin(115200);
    WiFi.begin(ssid, pass);
    client.begin("broker.emqx.io",net);
    client.onMessage(messageReceived);
    connect();
    SPI.begin(SCK, MISO, MOSI, SS); // SCK, MISO, MOSI, SS
    LoRa.setPins(SS, RST, DI0);
    Serial.begin(9600);
    delay(1000);
    Serial.println();Serial.println();
    Serial.println("Starting LoRa Receiver");
    display.init();
    display.flipScreenVertically();
    if (!LoRa.begin(BAND)) {
        Serial.println("Starting LoRa failed!");
        while (1);
    }
    else
    {
        Serial.println("Starting LoRa ok!");
        LoRa.setSpreadingFactor(sf);
        LoRa.setSignalBandwidth(sbw);
    }
}

unsigned long lastMillis = 0;

void loop() {
    int i=0, rssi;
    char bt1[32],bt2[32],bt3[32];
    // try to parse packet
    client.loop(); // to activate message received
    int packetSize = LoRa.parsePacket();
    if (packetSize==64) {
        // received a packet
        Serial.print("Received packet ");
        // read packet
        while (LoRa.available()) {
            rdp.frame[i] = LoRa.read();i++;
        }
        rssi=LoRa.packetRssi();
        Serial.println(rdp.sensor[0]);
        Serial.println(rdp.sensor[1]);
        Serial.println(rdp.sensor[2]);
    }
}

```

```

Serial.println(rdp.sensor[3]);
sprintf(bt1, "S1:%2.2f, S2:%2.2f", rdp.sensor[0], rdp.sensor[1]);
sprintf(bt2, "S3:%2.2f, S4:%2.2f", rdp.sensor[2], rdp.sensor[3]);
sprintf(bt3, "RSSI:%d", rssi);
disp("LoRa packet", bt1, bt2, bt3);
Serial.print("' with RSSI ");
Serial.println(rssi);
if (!client.connected()) {
  connect(); }
if (millis() - lastMillis > 4000) {
  lastMillis = millis();
  client.publish("/esp32c3/test", bt1);
}
}
}

```

Le résultat affiché sur le terminal

```

..
incoming: /esp32c3/test - S1:278.51, S2:557.02
Received packet '278.61
557.22
835.78
1114.43
' with RSSI -46
incoming: /esp32c3/test - S1:278.61, S2:557.22
Received packet '278.71
557.42
836.08
1114.83
' with RSSI -45
incoming: /esp32c3/test - S1:278.71, S2:557.42

```

A faire :

1. Tester le code de la passerelle
2. Ajouter les capteurs – SHT21 et BH1750 sur le terminal pour envoyer les « vrais » données
3. Remplacer l'utilisation directe de connexion WiFi par le **WiFiManager**.
4. Utiliser la réception avec la fonction de callback

4.2 Passerelle LoRa-WiFi (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**

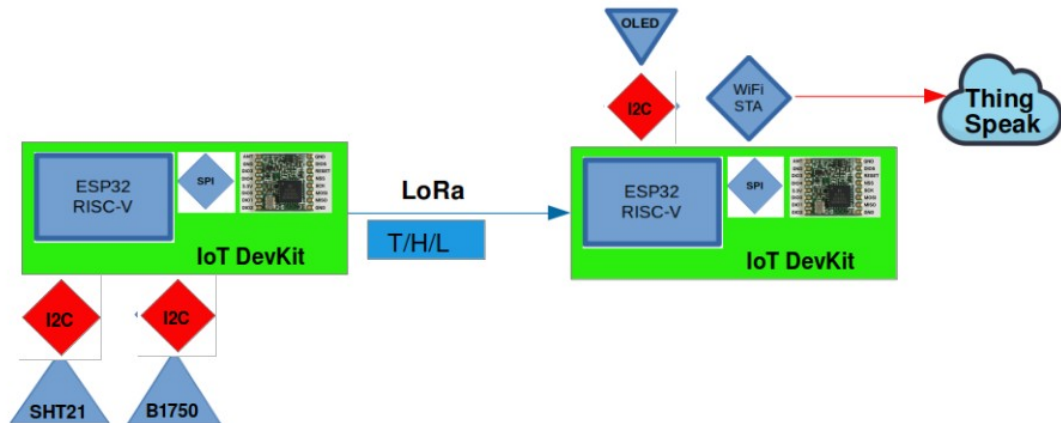


Figure 4.2 Une architecture IoT avec un terminal à 2 capteurs et une passerelle **LoRa-WiFi** vers **ThingSpeak.com**

4.2.1 Le principe de fonctionnement

La passerelle attend les messages LoRa envoyés dans le format prédéfinie sur une interruption I/O redirigée vers la fonction (ISR – **Interrupt SubRoutine**) `onReceive()`. La fonction ISR les stocke les données 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 est transmis par plusieurs fonctions :

```
ThingSpeak.setField(fn,value);
```

et la fonction

```
ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
```

4.1.2 Les éléments du code

Ci-dessous nous présentons les éléments du code de la passerelle. Les messages LoRa sont envoyés dans un format pré-défini par le type de l'union/structure d'un paquet (`pack_t`) de la façon suivante.

```
typedef union                // type definition
{
    uint8_t frame[64];
    char mess[64];
    float sensor[8];
} pack_t;
```

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:

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

void onReceive(int packetSize)
{
    pack_t rdp; // receive buffer with pack_t format
    int i=0;
    if (packetSize == 0) return; // if there's no packet, return
```

```

i=0;
if (packetSize==64)  //
{
  while (LoRa.available()) { rdp.frame[i]=LoRa.read();i++; }
  xQueueReset(dqueue); // to keep only the last element
  xQueueSend(dqueue, &rdp,portMAX_DELAY ); //portMAX_DELAY
}
}

```

La fonction de `setup()` :

```

void setup() {
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);
  WiFiManager wm;
  // wm.resetSettings();
  bool res;
  res = wm.autoConnect("ESP32AP",NULL); // no password
  if(!res) Serial.println("Failed to connect");// ESP.restart();
  else Serial.println("connected...yeey :)");
  ThingSpeak.begin(client); // connexion (TCP) du client au serveur
  delay(1000);
  Serial.println("ThingSpeak begin");
  SPI.begin(SCK, MISO, MOSI, SS); // SCK, MISO, MOSI, SS
  LoRa.setPins(SS, RST, DI0);
  Serial.begin(9600);
  delay(1000);
  Serial.println();Serial.println();
  Serial.println("Starting LoRa Receiver");
  display.init();
  display.flipScreenVertically();
  if (!LoRa.begin(BAND)) { Serial.println("Starting LoRa failed!");while (1); }
  else
  {
    Serial.println("Starting LoRa ok!");
    LoRa.setSpreadingFactor(sf);
    LoRa.setSignalBandwidth(sbw);
    dqueue = xQueueCreate(4,64); // queue for 4 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()` :

```

void loop()
{
  pack_t rp; // packet elements to send
  xQueueReceive(dqueue,rp.frame,portMAX_DELAY); // 6s,default:portMAX_DELAY
  ThingSpeak.setField(1,rp.sensor[0]);
  ThingSpeak.setField(2,rp.sensor[1]);
  ThingSpeak.setField(3,rp.sensor[2]);
  ThingSpeak.setField(4,rp.sensor[3]);
  Serial.printf("d1=%2.2f,d2=%2.2f,d3=%2.2f,d4=%2.2f\n",rp.sensor[0],rp.sensor[1],
rp.sensor[2],rp.sensor[3]);
  while (WiFi.status() != WL_CONNECTED) { delay(500); }
  ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
  delay(mindel); // mindel - min waiting time before sending to ThingSpeak
}

```

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

```

#include <LoRa.h>
#include <Wire.h> // Only needed for Arduino 1.6.5 and earlier

#include <WiFiManager.h>
#include "ThingSpeak.h"

unsigned long myChannelNumber =1626377;
const char *myWriteAPIKey="3IN09682SQX3PT4Z" ;
const char *myReadAPIKey="9JVTP8ZHVTB9G4TT" ;

WiFiClient client;

```

```

#include "SSD1306Wire.h"          // legacy: #include "SSD1306.h"

// Initialize the OLED display using Arduino Wire:
SSD1306Wire display(0x3c, 8, 10); // ADDRESS, SDA, SCL

// with LoRa modem RFM95 and green RFM board - int and RST to solder

#define SS      5 // 26      // D0 - to NSS
#define RST     3 //16      // D4 - RST
#define DIO     2           // D8 - INTR

#define SCK     1           // D5 - CLK
#define MISO    0           // D6 - MISO
#define MOSI    4           // D7 - MOSI
#define BAND    868E6

int sf=7;
long sbw=125E3;
typedef union          // type definition
{
    uint8_t frame[64];
    char mess[64];
    float sensor[8];
} pack_t;

QueueHandle_t dqueue; // queues for data packets

void onReceive(int packetSize)
{
    pack_t rdp; // receive buffer with pack_t format
    int i=0;
    if (packetSize == 0) return; // if there's no packet, return
    i=0;
    if (packetSize==64) //
    {
        while (LoRa.available()) { rdp.frame[i]=LoRa.read();i++; }
        xQueueReset(dqueue); // to keep only the last element
        xQueueSend(dqueue, &rdp, portMAX_DELAY ); //portMAX_DELAY
    }
}

void disp(char *t1, char *t2, char *t3, char *t4)
{
    display.clear();
    display.setFont(ArialMT_Plain_10);
    display.setTextAlignment(TEXT_ALIGN_LEFT);
    display.drawString(0, 0, t1);
    display.drawString(0, 16, t2);
    display.drawString(0, 32, t3);
    display.drawString(0, 48, t4);
    display.display();
}

void setup() {
    Serial.begin(115200);
    WiFi.mode(WIFI_STA);
    WiFiManager wm;
    // wm.resetSettings();
    bool res;
    res = wm.autoConnect("ESP32AP",NULL); // no password
    if(!res) Serial.println("Failed to connect");// ESP.restart();
    else Serial.println("connected...yeey :)");
    ThingSpeak.begin(client); // connexion (TCP) du client au serveur
    delay(1000);
    Serial.println("ThingSpeak begin");
    SPI.begin(SCK, MISO, MOSI, SS); // SCK, MISO, MOSI, SS
    LoRa.setPins(SS, RST, DIO);
    Serial.begin(9600);
    delay(1000);
    Serial.println();Serial.println();
    Serial.println("Starting LoRa Receiver");
    display.init();
    display.flipScreenVertically();
    if (!LoRa.begin(BAND)) {
        Serial.println("Starting LoRa failed!");
        while (1);
    }
}

```



```

else
{
  Serial.println("Starting LoRa ok!");
  LoRa.setSpreadingFactor(sf);
  LoRa.setSignalBandwidth(sbw);
  dqueue = xQueueCreate(4,64); // queue for 4 data packets
  LoRa.onReceive(onReceive); // pour indiquer la fonction ISR
  LoRa.receive();           // pour activer l'interruption (une fois)
}
}

uint32_t mindel=15000; // 15 seconds

void loop()
{
  pack_t rp; // packet elements to send
  xQueueReceive(dqueue, rp.frame, portMAX_DELAY); // 6s, default: portMAX_DELAY
  ThingSpeak.setField(1, rp.sensor[0]);
  ThingSpeak.setField(2, rp.sensor[1]);
  ThingSpeak.setField(3, rp.sensor[2]);
  ThingSpeak.setField(4, rp.sensor[3]);
  Serial.printf("d1=%2.2f, d2=%2.2f, d3=%2.2f, d4=%2.2f\n", rp.sensor[0], rp.sensor[1],
rp.sensor[2], rp.sensor[3]);
  while (WiFi.status() != WL_CONNECTED) { delay(500); }
  ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
  delay(mindel); // mindel - min waiting time before sending to ThingSpeak
}

```

A faire :

1. Tester le code de base
2. Tester le programme de passerelle avec un puis avec 2 terminaux.
3. Afficher les contenus des paquets sur l'écran OLED de la passerelle
4. Utiliser les capteurs SHT21 et BH1750 pour envoyer des données « réelles »

Laboratoire 5

Protocoles UDP, TCP et Web Sockets

Dans ce laboratoire, nous allons implémenter des protocoles de transport Internet de base tels que UDP, TCP ou WebSockets largement utilisés pour la messagerie IoT.

5.1 Sockets UDP - envoi et réception de datagrammes

Les fonctions du **socket UDP** permettent une communication IP simple à l'aide du protocole de datagramme utilisateur (UDP).

Le protocole de datagramme utilisateur (UDP - User Datagram Protocol) s'exécute au-dessus du protocole Internet (IP) et a été développé pour les applications qui ne nécessitent pas des fonctionnalités de fiabilité, d'accusé de réception ou de contrôle de flux au niveau de la couche de transport. Ce protocole simple fournit un adressage de la couche de transport sous la forme de **ports UDP** et un champs de somme de contrôle facultative.

L'en-tête du protocole UDP est très simple.

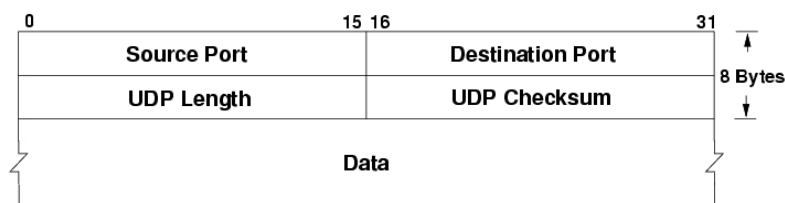
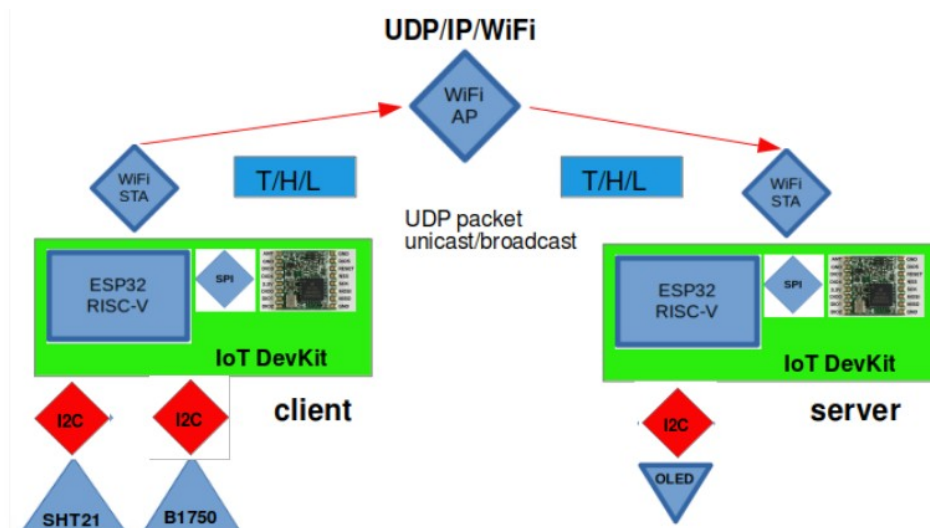


Fig 5.1 Datagramme UDP avec en-tête

Les messages, appelés **datagrammes**, sont envoyés à d'autres hôtes sur un réseau IP sans qu'il soit nécessaire de configurer au préalable des canaux de transmission ou des chemins de données spéciaux. Le socket UDP ne doit être ouverte que pour la communication. Il écoute les messages entrants et envoie les messages sortants sur demande.

Sur le réseau IP donné, les datagrammes peuvent être envoyés en mode **unicast** ou **broadcast**.

5.1.1 Architecture IoT : Client-Serveur (UDP-STA)



5.1.1.1 Code - Client – émetteur UDP

L'exemple suivant contient du code client UDP qui envoie des datagrammes simples en **mode diffusion (broadcast)** sur **ce** réseau.

Notez que les données ont des valeurs constantes. Ils sont préparés dans la structure type **union** :

```
union
{
    uint8_t frame[16];
    float sensor[4];
    char mess[16];
} sdp, rdp; // send and receive packets
```

La même union est utilisée côté récepteur UDP.

```
#include <WiFi.h>
#include <WiFiUdp.h>
/* WiFi network name and password */
const char * ssid= "Livebox-08B0";
const char * pwd = "G79ji6dtEptVTPWmZP";
const char * udpAddress = "192.168.1.255"; // on this network
const int udpPort = 1234;
//create UDP instance
WiFiUDP udp;

union
{
    uint8_t frame[16];
    float sensor[4];
    char mess[16];
} sdp, rdp; // send and receive packets

void setup()
{
    Serial.begin(115200);
    WiFi.begin(ssid, pwd);
    Serial.println("");
    // Wait for connection
    while (WiFi.status() != WL_CONNECTED) {delay(500); Serial.print(".");}
    Serial.println("");
    Serial.print("Connected to ");
    Serial.println(ssid);
    Serial.print("IP address: ");
    Serial.println(WiFi.localIP());
    //This initializes udp and transfer buffer
    udp.begin(udpPort);
}

void loop()
{
    Serial.println("Send unicast/broadcast packet ");
    udp.beginPacket(udpAddress, udpPort);
    sdp.sensor[0]=0.1;sdp.sensor[1]=0.2;sdp.sensor[2]=0.3;sdp.sensor[3]=0.4;
    udp.write(sdp.frame, 16);
    udp.endPacket();
    memset(rdp.frame,0,16);
    //processing incoming packet, must be called before reading the buffer
    udp.parsePacket();
    //receive response from server, it will be ACK
    if(udp.read(rdp.frame,16)==16)
    {
        Serial.print("Received from server: "); Serial.println(rdp.mess);
    }
    //Wait for 2 seconds
    delay(2000);
}
```

5.1.1.2 Code - Serveur – récepteur UDP synchrone

```
#include <WiFi.h>
#include <WiFiUdp.h>
/* WiFi network name and password */
const char * ssid= "Livebox-08B0";
const char * pwd = "G79ji6dtEptVTPWmZP";
const char * udpAddress = "192.168.1.255";
const int udpPort = 1234;
//create UDP instance
WiFiUDP udp;

union
{
  uint8_t frame[16];
  float sensor[4];
  char mess[16];
} rdp, sdp;

void setup()
{
  Serial.begin(115200);
  //Connect to the WiFi network
  WiFi.begin(ssid, pwd);
  Serial.println("");
  // Wait for connection
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500); Serial.print(".");
  }
  Serial.println("");
  Serial.print("Connected to ");
  Serial.println(ssid);
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
  //This initializes udp and transfer buffer
  udp.begin(udpPort);
}

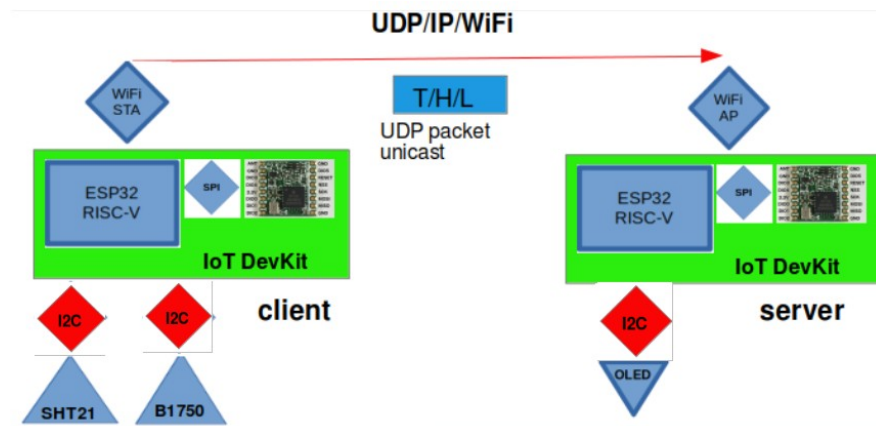
void loop()
{
  memset(rdp.frame, 0, 16);
  //processing incoming packet, must be called before reading the buffer
  udp.parsePacket();
  if(udp.read(rdp.frame, 16) == 16) // read packet from client
  {
    Serial.printf("\nRecv data: %f, %f, %f, %f\n", rdp.sensor[0], rdp.sensor[1], rdp.sensor[2], rdp.sensor[3]);
    //Wait for 1 second
    delay(1000);
    Serial.print("Send ACK packet ");
    udp.beginPacket(udpAddress, udpPort);
    strcpy(sdp.mess, "ACK data");
    udp.write(sdp.frame, 16);
    udp.endPacket(); // send packet to client
  }
}
```

Résultat de réception :

```
Connected to Livebox-08B0
IP address: 192.168.1.65

Recv data:0.100000,0.200000,0.300000,0.400000
Send ACK packet
Recv data:0.100000,0.200000,0.300000,0.400000
Send ACK packet
Recv data:0.100000,0.200000,0.300000,0.400000
Send ACK packet
Recv data:0.100000,0.200000,0.300000,0.400000
Send ACK packet
```

5.1.2 Architecture IoT avec un serveur UDP synchrone et softAP



5.1.2.1 Code - Serveur UDP avec un softAP

```
#include <WiFi.h>
#include <WiFiUdp.h>
/* WiFi network name and password */
const char * ssid= "ESP32_AP";
const char * pwd = "smartcomputerlab";
const int udpPort = 9999;
//create UDP instance
WiFiUDP udp;
union
{
  {
    uint8_t frame[16];
    float sensor[4];
    char mess[16];
  } rdp, sdp;

void setup()
{
  Serial.begin(115200);
  //Connect to the WiFi network
  WiFi.softAP(ssid, pwd);
  Serial.print("\nServer SoftIP: ");
  Serial.println(WiFi.softAPIP());
  udp.begin(udpPort);
}

void loop()
{
  memset(rdp.frame,0,16);
  //processing incoming packet, must be called before reading the buffer
  udp.parsePacket();
  //receive response from server, it will be HELLO WORLD
  if(udp.read(rdp.frame,16)==16)
  {
    IPAddress remaddr;
    remaddr=udp.remoteIP();
    uint16_t remport;
    remport=udp.remotePort();
    Serial.println(remaddr);
    Serial.printf("\nRecv data:%f,%f,%f,%f\n", rdp.sensor[0], rdp.sensor[1], rdp.sensor[2],
    rdp.sensor[3]);

    //Wait for 1 second
    delay(1000);
    Serial.print("Send ACK packet ");
    //udp.beginPacket(udpAddress, udpPort);
    udp.beginPacket(remaddr, remport);
    strcpy(sdp.mess,"ACK data");
    udp.write(sdp.frame,16);
    udp.endPacket();
  }
}
```

5.1.2.2 Code - Client UDP synchrone pour le serveur avec SoftAP

```
#include <WiFi.h>
#include <WiFiUdp.h>
/* WiFi network name and password */
const char * ssid= "ESP32_AP";
const char * pwd = "smartcomputerlab";
// a network broadcast address like below
const char * udpAddress = "192.168.4.255";
const int udpPort = 9999;
//create UDP instance
WiFiUDP udp;
union
{
  uint8_t frame[16];
  float sensor[4];
  char mess[16];
} sdp, rdp; // send and receive packets

void setup()
{
  Serial.begin(115200);
  //Connect to the WiFi network
  WiFi.begin(ssid, pwd);
  Serial.println("");
  // Wait for connection
  while (WiFi.status() != WL_CONNECTED) {delay(500); Serial.print(".");}
  Serial.println("");
  Serial.print("Connected to ");
  Serial.println(ssid);
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
  //This initializes udp and transfer buffer
  udp.begin(udpPort);
}

void loop()
{
  //data will be sent to server
  Serial.println("Send unicast/broadcast packet ");
  udp.beginPacket(udpAddress, udpPort);
  sdp.sensor[0]=0.1;sdp.sensor[1]=0.2;sdp.sensor[2]=0.3;sdp.sensor[3]=0.4;
  udp.write(sdp.frame, 16);
  udp.endPacket();
  memset(rdp.frame,0,16);
  //processing incoming packet, must be called before reading the buffer
  udp.parsePacket();
  //receive response from server, it will be ACK
  if(udp.read(rdp.frame,16)==16)
  {
    Serial.print("Received from server: ");
    Serial.println(rdp.mess);
  }
  //Wait for 2 seconds
  delay(2000);
}
```

A faire

1. Analysez les codes et exécutez-les
2. Ajouter le ou les capteurs côté client
3. Ajoutez l'écran OLED côté serveur

5.2.1 Sockets TCP – établissement de connexions et envoi/réception de données (segments)

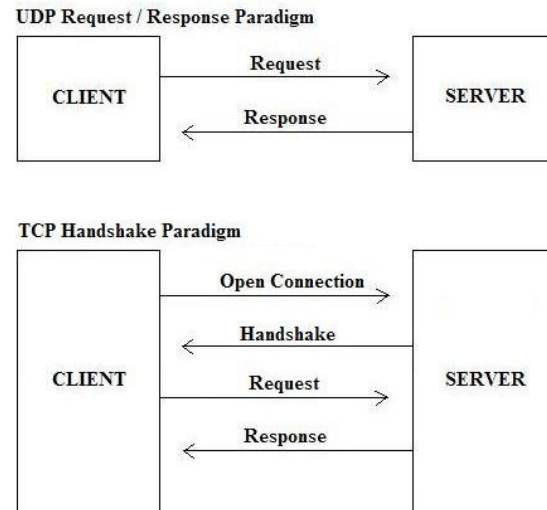


Fig 5.2 Segment TCP avec en-tête

Le protocole UDP est un **protocole sans connexion** où pour les messages - datagrammes . Les datagrammes peuvent être envoyés immédiatement après l'initialisation de la connexion WiFi.

TCP est un protocole **orienté connexion** dans lequel les données peuvent être envoyées après l'établissement d'une « connexion ». Cette connexion est identifiée par le **numéro de séquence initial (Initial Sequence Number)**.

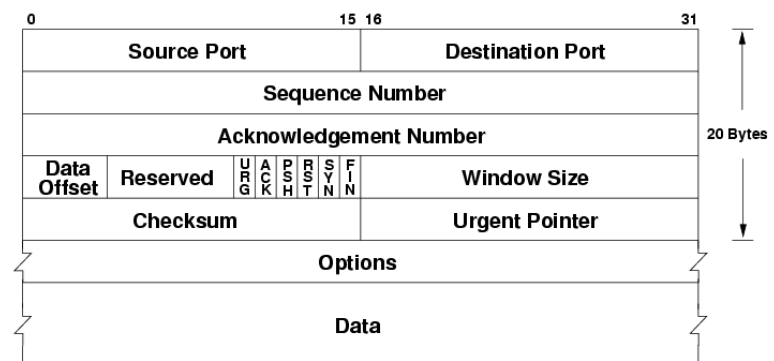
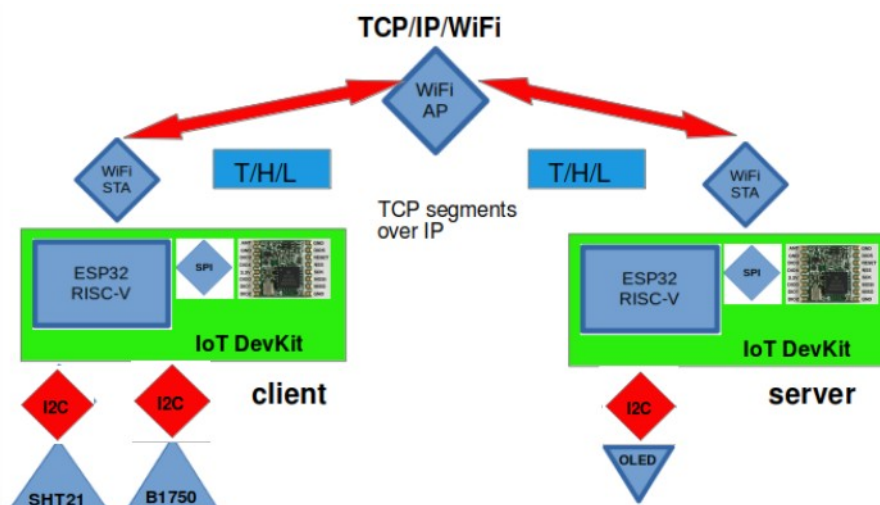


Fig 5.3 Segment TCP avec en-tête

5.2.2 Architecture IoT avec client-serveur TCP en mode STA



5.2.2.1 Code - Serveur TCP en mode STA

```
#include <WiFi.h>
const char* ssid = "Livebox-08B0"; // Enter SSID
const char* password = "G79ji6dtEptVTPWmZP"; // Enter Password
/* create a server and listen on port 8088 */
WiFiServer server(8088);
IPAddress local_IP(192, 168, 1, 131);
IPAddress gateway(192, 168, 1, 1);
IPAddress subnet(255, 255, 255, 0);

union
{
  uint8_t frame[16];
  float sensor[4];
  char mess[16];
} sdp, rdp; // send and receive packets

void setup()
{
  Serial.begin(115200);
  Serial.println("\nConnecting to ");
  Serial.println(ssid);
  /* connecting to WiFi */
  WiFi.begin(ssid, password);
  /*wait until ESP32 connect to WiFi*/
  while(WiFi.status() != WL_CONNECTED) { delay(500); Serial.print("."); }
  Serial.println("\nConnected to ");
  WiFi.config(local_IP, gateway, subnet);
  Serial.print("Server IP address: ");
  Serial.println(WiFi.localIP());
  /* start Server */
  server.begin();
  delay(400);
  Serial.println("Waiting for client");
}

void loop()
{
  uint8_t data[30];
  /* listen for client */
  WiFiClient client = server.available();
  if (client)
  {
    Serial.println("new client");
    /* check client is connected */
    while (client.connected())
    {
      if (client.available())
      {
        {
          int len = client.read(rdp.frame, 16);
          if(len==16)
          {
            Serial.print("client sent: ");
            Serial.println(rdp.sensor[0]);
            strcpy(sdp.mess, "ACK data");
            delay(1000);
            client.write(sdp.frame, 16);
          }
        }
      }
    }
  }
}
```

5.2.2.2 Code - Client TCP

```
#include <WiFi.h>
/* change ssid and password according to yours WiFi*/
const char* ssid = "Livebox-08B0"; //Enter SSID
const char* password = "G79ji6dtEptVTPWmZP"; //Enter Password
const char* host = "192.168.1.131";
const int port = 8088;

union
{
  uint8_t frame[16];
```



```

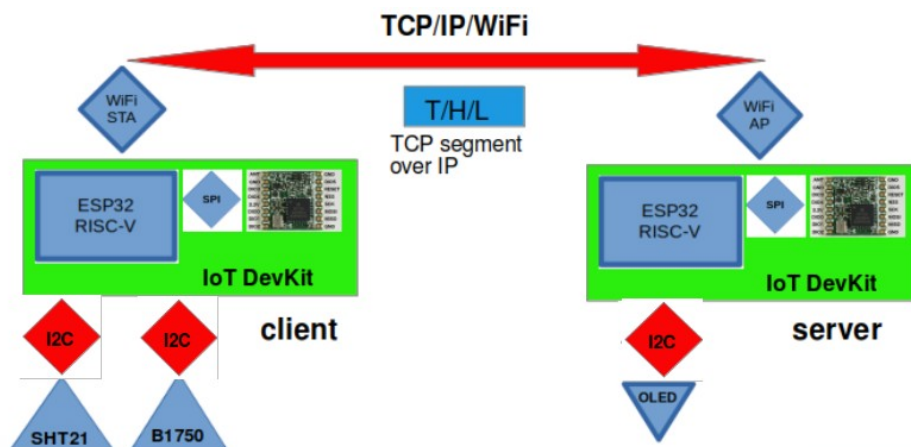
float sensor[4];
char mess[16];
} sdp, rdp; // send and receive packets

void setup()
{
  Serial.begin(115200);
  Serial.print("Connecting to ");
  Serial.println(ssid);
  /* connect to your WiFi */
  WiFi.begin(ssid, password);
  /* wait until ESP32 connect to WiFi*/
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected with IP address: ");
  Serial.println(WiFi.localIP());
}

void loop()
{
  uint8_t data[30];
  int len=0;
  delay(5000);
  Serial.print("\nConnecting to ");
  Serial.println(host);
  /* Use WiFiClient class to create TCP connections */
  WiFiClient client;
  if (!client.connect(host, port))
  {
    Serial.println("connection failed");
    return;
  }
  Serial.println("connection OK - client sends data: ");
  sdp.sensor[0]=0.1; sdp.sensor[1]=0.2; sdp.sensor[2]=0.3; sdp.sensor[3]=0.4;
  client.write(sdp.frame,16);
  delay(2000);
  len = client.read(rdp.frame,16);
  Serial.println(len);
  if(len==16)
  {
    Serial.print("server sent: ");
    Serial.println(rdp.mess);
  }
  client.stop();
}

```

5.2.3 Architecture IoT -TCP avec serveur en mode SoftAP



5.2.3.1 Code - Serveur TCP avec SoftAP

```
#include <WiFi.h>
const char * ssid= "ESP32_AP";
const char * pwd = "smartcomputerlab";

WiFiServer server(8088); // create a server and listen on port 8088

union
{
    uint8_t frame[16];
    float sensor[4];
    char mess[16];
} sdp, rdp; // send and receive packets

void setup()
{
    Serial.begin(115200);
    WiFi.softAP(ssid, pwd);
    Serial.print("\nServer SoftIP: ");
    Serial.println(WiFi.softAPIP());
    /* start Server */
    server.begin();
    delay(400);
    Serial.println("Waiting for client");
}

void loop()
{
    uint8_t data[30];
    /* listen for client */
    WiFiClient client = server.available();
    if (client)
    {
        Serial.println("new client");
        /* check client is connected */
        while (client.connected())
        {
            if (client.available())
            {
                int len = client.read(rdp.frame, 16);
                if(len==16)
                {
                    Serial.print("client sent: ");
                    Serial.println(rdp.sensor[0]);
                    strcpy(sdp.mess, "ACK data");
                    delay(1000);
                    client.write(sdp.frame, 16);
                }
            }
        }
    }
}
```

5.2.3.2 Client TCP pour un serveur avec SoftAP

```
#include <WiFi.h>
/* change ssid and password according to yours WiFi*/
//const char * ssid= "Livebox-08B0";
//const char * pwd = "G79ji6dtEptVTPWmZP";
const char * ssid= "ESP32_AP";
const char * pwd = "smartcomputerlab";
const char* host = "192.168.4.1";
const int port = 8088;

union
{
    uint8_t frame[16];
    float sensor[4];
    char mess[16];
} sdp, rdp; // send and receive packets

void setup()
{
    Serial.begin(115200);
    //Connect to the WiFi network
    WiFi.begin(ssid, pwd);
```

```

Serial.println("");
// Wait for connection
while (WiFi.status() != WL_CONNECTED) {delay(500); Serial.print(".");}
Serial.println("");
Serial.print("Connected to ");
Serial.println(ssid);
Serial.print("IP address: ");
Serial.println(WiFi.localIP());
//This initializes udp and transfer buffer
}

void loop()
{
uint8_t data[30];
int len=0;
  delay(5000);
  Serial.print("\nConnecting to ");
  Serial.println(host);
  /* Use WiFiClient class to create TCP connections */
  WiFiClient client;
  if (!client.connect(host, port))
  {
    Serial.println("connection failed");
    return;
  }
  // This will send the data to the server
  Serial.println("connection OK - client sends data: ");
  sdp.sensor[0]=0.1; sdp.sensor[1]=0.2; sdp.sensor[2]=0.3; sdp.sensor[3]=0.4;
  client.write(sdp.frame,16);
  delay(2000);
  len = client.read(rdp.frame,16);
  Serial.println(len);
  if(len==16)
  {
    Serial.print("server sent: ");
    Serial.println(rdp.mess);
  }
  client.stop();
}

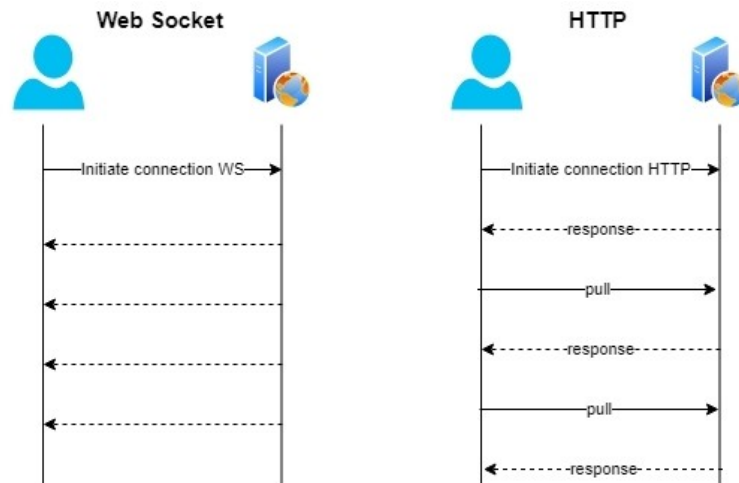
```

A faire

1. Analysez les codes et exécutez-les
2. Ajouter le ou les capteurs côté client
3. Ajoutez l'écran OLED côté serveur

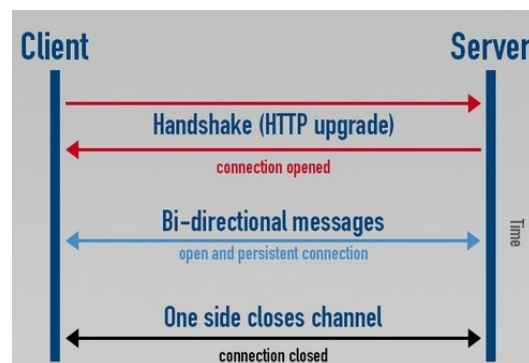
5.3 Web Socket

Web socket est un **protocole de communication sur TCP** qui permet une communication bidirectionnelle entre le client et le serveur.



Comme vous pouvez le voir dans le diagramme ci-dessus, le client initie une connexion via **WebSocket** ou **WebSocket sécurisé** (wss), puis le serveur peut renvoyer des messages au client. Avec le protocole HTTP, nous n'avons pas besoin de rétablir la connexion. Une fois la connexion établie, le serveur peut envoyer autant de requêtes qu'il le souhaite.

Le plus grand avantage de l'utilisation de Web Socket au lieu de HTTP est l'amélioration des performances car nous n'avons pas besoin d'établir une nouvelle connexion. En plus, nous n'avons pas besoin d'introduire une sorte de mécanisme de connexion.



Avantages :

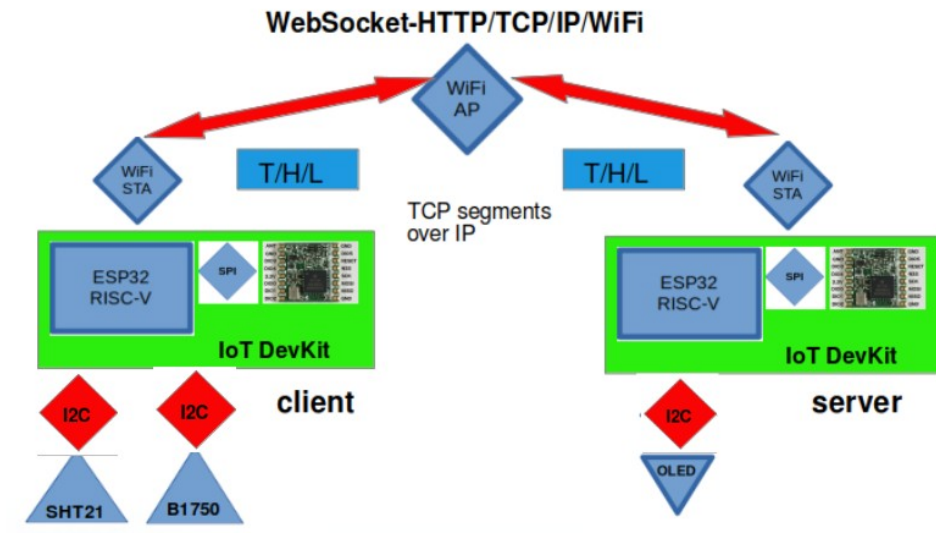
Échange/transfert de données bidirectionnel

En transmettant des données dans les deux sens simultanément via une seule connexion au lieu de deux, vous réduisez le trafic et les retards inutiles sur le réseau.

Plateforme d'événement Publier/S'abonner

Envoyez des messages vers et depuis un serveur et recevez des réponses événementielles sans avoir à interroger le serveur pour obtenir une réponse.

5.3.1 Architecture IoT avec client-serveur sur WebSockets



5.3.1.1 Client

Création d'un client et connexion à un serveur :

```
WebsocketsClient client;  
client.connect("ws://your-server-ip:port/uri");
```

Envoi d'un message :

```
client.send("Hello Server!");
```

En attente de messages :

```
client.onMessage([] (WebsocketsMessage msg) {  
    Serial.println("Got Message: " + msg.data());  
});
```

5.3.1.2 Server

Création d'un serveur et écoute des connexions :

```
WebsocketsServer server;  
server.listen(8080);
```

Accepter les connexions :

```
WebsocketsClient client = server.accept();  
// handle client as described before :)
```

5.3.2.1 Code - Client Websockets minimal

Dans cet exemple :

1. On se connecte à un réseau WiFi
2. On se connecte à un serveur Websockets
3. On envoie un message au serveur websockets ("Hello Server")
4. On imprime tous les messages entrants pendant que la connexion est ouverte

```
#include <ArduinoWebsockets.h>  
#include <WiFi.h>  
const char* ssid = "Livebox-08B0"; // Enter your SSID  
const char* password = "G79ji6dtEptVTPWmZP"; // Enter your Password  
const char* websockets_server_host = "192.168.1.132"; //Enter server adress  
const uint16_t websockets_server_port = 8080; // Enter server port  
  
using namespace websockets;
```

```

WebsocketsClient client;

union
{
  char message[17];
  uint8_t frame[17];
  float sensor[4];
} sdf;

void setup() {
  Serial.begin(115200);
  // Connect to wifi
  WiFi.begin(ssid, password);
  // Wait some time to connect to wifi
  for(int i = 0; i < 10 && WiFi.status() != WL_CONNECTED; i++) {
    Serial.print(".");
    delay(1000);
  }
  Serial.println();
  Serial.print("client connected to WiFi");
  // Connect to server
  client.connect(websockets_server_host, websockets_server_port, "/");
  // Send a message - data
  sdf.sensor[0]=0.1;sdf.sensor[1]=0.2;sdf.sensor[2]=0.3;sdf.sensor[3]=0.4;
  client.send(sdf.message, 17);
}

void loop() {
  client.connect(websockets_server_host, websockets_server_port, "/");
  // Send a message - data
  sdf.sensor[0]=0.1;sdf.sensor[1]=0.2;sdf.sensor[2]=0.3;sdf.sensor[3]=0.4;
  client.send(sdf.message, 17);
  Serial.print("client sent data");
  client.close();
  delay(3000);
}

```

5.3.2.2 Serveur Websockets minimal

Dans cet exemple:

1. On se connecte à un réseau WiFi
2. On démarre un serveur websocket sur le port 80
3. On attend les connexions
4. Une fois qu'un client se connecte, il attend un message du client
5. On envoie éventuellement un message "ACK" au client
6. On ferme la connexion et revient à l'étape 3

```

#include <ArduinoWebsockets.h>
#include <WiFi.h>
const char* ssid = "Livebox-08B0"; // Enter your SSID
const char* password = "G79ji6dtEptVTPWmZP"; // Enter your Password
IPAddress local_IP(192, 168, 1, 132); // set static values on your network
IPAddress gateway(192, 168, 1, 1);
IPAddress subnet(255, 255, 255, 0);
using namespace websockets;
int sport=8080;

WebsocketsServer server;

union
{
  uint8_t frame[17];
  char message[17];
  float sen[4];
} sdf;

void setup() {
  Serial.begin(115200);
  // Connect to wifi
  WiFi.begin(ssid, password);
  // Wait some time to connect to wifi
  for(int i = 0; i < 15 && WiFi.status() != WL_CONNECTED; i++) {
    Serial.print(".");
  }

```

```

        delay(1000);
    }
    Serial.println("\nServer connected");
    WiFi.config(local_IP, gateway, subnet);
    Serial.printf("Server port: %d and IP address: ", sport);
    Serial.println(WiFi.localIP());
    Serial.println("");
    server.listen(sport);
    Serial.print("Is server live? ");
    Serial.println(server.available());
}

void loop() {
    WebsocketsClient client = server.accept();
    if(client.available()) {
        WebsocketsMessage msg = client.readBlocking();
        // log
        Serial.println("\nGot Message: ");
        msg.data().toCharArray(sdf.message, 17);
        Serial.printf("Sensors: %2.2f,%2.2f,%2.2f,%2.2f", sdf.sen[0], sdf.sen[1], sdf.sen[2], sdf.sen[3]);
        client.send("ACK message");
        // close the connection
        client.close();
    }
    delay(1000);
}

```

A faire

1. Analysez les codes et exécutez-les
2. Ajouter le ou les capteurs côté client
3. Ajoutez l'écran OLED côté serveur
4. Utilisez **softAP** pour le serveur et le client se connectant à ce serveur

Laboratoire 6

Wi-Fi bas niveau

6.0 Introduction

Dans ce laboratoire, nous allons approfondir notre pratique des protocoles WiFi en commençant par l'analyse des trames WiFi et du code pour le générateur de **balises**.



Fig 6.1 Trame WiFi avec champs de contrôle et 3 (4) adresses MAC

```
typedef struct {
    unsigned frame_ctrl:16;
    unsigned duration_id:16;
    uint8_t addr1[6]; /* receiver address */
    uint8_t addr2[6]; /* sender address */
    uint8_t addr3[6]; /* filtering address */
    unsigned sequence_ctrl:16;
    uint8_t addr4[6]; /* optional */
} wifi_ieee80211_mac_hdr_t;

typedef struct {
    wifi_ieee80211_mac_hdr_t hdr;
    uint8_t payload[0]; /* network data ended with 4 bytes csum (CRC32) */
} wifi_ieee80211_packet_t;
```

L'image ci-dessus montre le format de la trame WiFi, elle se compose de l'en-tête et de la partie charge utile (**payload**). L'en-tête contient le champ de contrôle, le champ de durée, l'emplacement de quatre adresses MAC et le champ de contrôle de séquence.

Type Value B3..B2	Type Description	Subtype Value B7 .. b4	Subtype Description
00	Management	0000	Association Request
00	Management	0001	Association Response
00	Management	0010	Reassociation Request
00	Management	0011	Reassociation Response
00	Management	0100	Probe Request
00	Management	0101	Probe Response
00	Management	0110	Timing Advertisement
00	Management	0111	Reserved
00	Management	1000	Beacon
00	Management	1001	ATIM
00	Management	1010	Disassociation
00	Management	1011	Authentication
00	Management	1100	Deauthentication
00	Management	1101	Action
00	Management	1110	Action No Ack (NACK)
00	Management	1111	Reserved

Fig 5.2 Trames MAC de gestion WiFi

2.1.1 Générateur des trames WiFi (balises)

L'en-tête complet de la trame MAC est codé comme suit :

```
uint8_t beacon_raw[] = {
    0x80,0x00,          // 0-1: Frame Control: subtype 80 - beacon, type 00 MNGM
    0x00,0x00,          // 2-3: Duration
    0xff,0xff,0xff,0xff,0xff,0xff, // 4-9: Destination address (broadcast)
    0xba,0xde,0xaf,0xfe,0x00,0x06, // 10-15: Source address
    0xba,0xde,0xaf,0xfe,0x00,0x06, // 16-21: BSSID
    0x00,0x00,          // 22-23: Sequence / fragment number
    0x00,0x01,0x02,0x03,0x04, 0x05, 0x06, 0x07, // 24-31: Timestamp
    0x64,0x00,          // 32-33: Beacon interval
    0x31,0x04,          // 34-35: Capability info
    0x00,0x00,          // 36-38: SSID parameter set, 0x00:length:content
    0x01,0x08,0x82,0x84,0x8b,0x96,0x0c,0x12,0x18,0x24, // 39-48: Supported rates
    0x03,0x01,0x01,      // 49-51: DS Parameter set, current channel 1 (= 0x01),
    0x05,0x04,0x01,0x02,0x00,0x00,      // 52-57: Traffic Indication Map
};
```

Les noms du **ssid** proposé sont préparés dans la table **my_ssids[]**. Ils sont attachés à l'en-tête avant d'envoyer la trame sur la ligne.

```
char *my_ssids[] = {
    "01 Hello from SmartComputerLab",
    "02 Hello from SmartComputerLab",
    "03 Hello from SmartComputerLab",
    "04 Hello from SmartComputerLab",
};
```

Les constantes indiquant la position des champs sélectionnés dans la trame brute (**raw**) sont données ci-dessous :

```
#define BEACON_SSID_OFFSET 38
#define SRCADDR_OFFSET 10
#define BSSID_OFFSET 16
#define SEQNUM_OFFSET 22
#define TOTAL_LINES (sizeof(my_ssids) / sizeof(char *))
```

spam_task s'exécute dans la boucle **for(;;)** infinie.

Avant de générer une nouvelle trame, la tâche attend la période
100/TOTAL_LINES/portTICK_PERIOD_MS.

La place initiale de la trame est la table d'octets **beacon_rick[200]** (**uint8_t**). Cette table est remplie avec l'en-tête **beacon_raw** suivi du nom du **ssid**.

```
void spam_task(void *pvParameter) {
    uint8_t line = 0;
    // Keep track of beacon sequence numbers on a per line-basis
    uint16_t seqnum[TOTAL_LINES] = { 0 };
    for (;;) {
        vTaskDelay(100/TOTAL_LINES/portTICK_PERIOD_MS);
        printf("%i %i %s\r\n",strlen(my_ssids[line]),TOTAL_LINES,my_ssids[line]);
        uint8_t beacon_rick[200];
        memcpy(beacon_rick,beacon_raw,BEACON_SSID_OFFSET-1);
        beacon_rick[BEACON_SSID_OFFSET-1]=strlen(my_ssids[line]);
        memcpy(&beacon_rick[BEACON_SSID_OFFSET],my_ssids[line],strlen(my_ssids[line]));
        memcpy(&beacon_rick[BEACON_SSID_OFFSET+strlen(my_ssids[line])],
        &beacon_raw[BEACON_SSID_OFFSET],sizeof(beacon_raw)-BEACON_SSID_OFFSET);
```

Le dernier octet de l'adresse source et du BSSID est le **numéro de ligne** pour émuler les multiples points d'accès de diffusion.

```
        beacon_rick[SRCADDR_OFFSET+5]=line;
        beacon_rick[BSSID_OFFSET+5]=line;
```

Le champ du **numéro de séquence** est calculé comme suit :

```
beacon_rick[SEQNUM_OFFSET] = (seqnum[line] & 0x0f) << 4;
beacon_rick[SEQNUM_OFFSET + 1] = (seqnum[line] & 0xff0) >> 4;
seqnum[line]++;
```

```
if(seqnum[line] > 0xffff) seqnum[line] = 0;
```

Enfin le **frame-beacon** est envoyé par :

```
esp_wifi_80211_tx(WIFI_IF_AP, beacon_raw, sizeof(beacon_raw)+ strlen(my_ssids[line]), false);
```

Le nouveau numéro de ligne (**ssid**) est incrémenté :

```
if (++line >= TOTAL_LINES) line = 0;
```

À un moment donné, l'exécution de la tâche et l'esp32 lui-même passent en état de **veille profonde**. Cette solution permet de réduire la consommation électrique de l'appareil. Il n'y a aucune opération dans la tâche principale `loop()`.

```
if(count<0)
{
    esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);
    esp_deep_sleep_start(); count=30;
}
else count--;
}
```

La fonction `setup()` initialise les fonctions requises (drivers) traitant de l'interface WiFi :

```
void setup(void) {
    Serial.begin(9600);
    nvs_flash_init();
    tcpip_adapter_init();
    wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
    esp_event_loop_init(event_handler, NULL);
    esp_wifi_init(&cfg);
    esp_wifi_set_storage(WIFI_STORAGE_RAM);
```

Nous initialisons un **point d'accès factice** pour spécifier un canal et mettre le matériel WiFi dans un mode où nous pouvons envoyer les trames de fausses balises.

```
esp_wifi_set_mode(WIFI_MODE_AP);
esp_wifi_start();
esp_wifi_set_ps(WIFI_PS_NONE);
```

Enfin nous lançons la `spam_task`.

```
xTaskCreate(&spam_task, "spam_task", 2048, NULL, 5, NULL);
```

```
void loop()
{}
```

6.1.1.1 Code complet

```
#include "freertos/FreeRTOS.h"
#include "esp_wifi.h"
#include "esp_wifi_types.h"
#include "esp_system.h"
#include "esp_event.h"
#include "esp_event_loop.h"
#include "nvs_flash.h"
#include "driver/gpio.h"
#define uS_TO_S_FACTOR 1000000 /* Conversion factor to seconds */
#define TIME_TO_SLEEP 15 /* Time ESP32 will go to sleep (in seconds) */

esp_err_t esp_wifi_80211_tx(wifi_interface_t ifx, const void *buffer, int len, bool en_sys_seq);

uint8_t beacon_raw[] = {
    0x80, 0x00, // 0-1: Frame Control: subtype 80 - beacon, type 00 MNGM
    0x00, 0x00, // 2-3: Duration
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, // 4-9: Destination address (broadcast)
    0xba, 0xde, 0xaf, 0xfe, 0x00, 0x06, // 10-15: Source address
    0xba, 0xde, 0xaf, 0xfe, 0x00, 0x06, // 16-21: BSSID
    0x00, 0x00, // 22-23: Sequence / fragment number
    0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, // 24-31: Timestamp
    0x64, 0x00, // 32-33: Beacon interval
    0x31, 0x04, // 34-35: Capability info
    0x00, 0x00, // 36-38: SSID parameter set, 0x00:length:content
```

```

    0x01, 0x08, 0x82, 0x84, 0x8b, 0x96, 0x0c, 0x12, 0x18, 0x24, // 39-48: Supported rates
    0x03, 0x01, 0x01, // 49-51: DS Parameter set, current channel 1 (= 0x01),
    0x05, 0x04, 0x01, 0x02, 0x00, 0x00, // 52-57: Traffic Indication Map
};

char *my_ssids[] = { // dummy access points
    "01 Hello from SmartComputerLab",
    "02 Hello from SmartComputerLab",
    "03 Hello from SmartComputerLab",
    "04 Hello from SmartComputerLab",
};

#define BEACON_SSID_OFFSET 38
#define SRCADDR_OFFSET 10
#define BSSID_OFFSET 16
#define SEQNUM_OFFSET 22
#define TOTAL_LINES (sizeof(my_ssids) / sizeof(char *))

esp_err_t event_handler(void *ctx, system_event_t *event) { return ESP_OK;}

int count=120;

void spam_task(void *pvParameter) {
    uint8_t line = 0;
    // Keep track of beacon sequence numbers on a per-songline-basis
    uint16_t seqnum[TOTAL_LINES] = { 0 };
    for (;;) {
        vTaskDelay(100 / TOTAL_LINES / portTICK_PERIOD_MS);
        printf("%i %i %s\r\n", strlen(my_ssids[line]), TOTAL_LINES, my_ssids[line]);
        uint8_t beacon_rick[200];
        memcpy(beacon_rick, beacon_raw, BEACON_SSID_OFFSET - 1);
        beacon_rick[BEACON_SSID_OFFSET - 1] = strlen(my_ssids[line]);
        memcpy(&beacon_rick[BEACON_SSID_OFFSET], my_ssids[line], strlen(my_ssids[line]));
        memcpy(&beacon_rick[BEACON_SSID_OFFSET + strlen(my_ssids[line])],
            &beacon_raw[BEACON_SSID_OFFSET], sizeof(beacon_raw) - BEACON_SSID_OFFSET);
        // Last byte of source address/BSSID will be line number
        beacon_rick[SRCADDR_OFFSET + 5] = line;
        beacon_rick[BSSID_OFFSET + 5] = line;
        // Update sequence number
        beacon_rick[SEQNUM_OFFSET] = (seqnum[line] & 0x0f) << 4;
        beacon_rick[SEQNUM_OFFSET + 1] = (seqnum[line] & 0xff0) >> 4;
        seqnum[line]++;
        if (seqnum[line] > 0xffff)
            seqnum[line] = 0;
        esp_wifi_80211_tx(WIFI_IF_AP, beacon_rick, sizeof(beacon_raw) + strlen(my_ssids[line]), false);
        if (++line >= TOTAL_LINES)
            line = 0;
    }
    if(count<0)
    { esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);
      esp_deep_sleep_start(); count=30; }
    else count--;
}

void setup(void) {
    Serial.begin(9600);
    nvs_flash_init();
    tcpip_adapter_init();
    wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
    ESP_ERROR_CHECK(esp_event_loop_init(event_handler, NULL));
    ESP_ERROR_CHECK(esp_wifi_init(&cfg));
    ESP_ERROR_CHECK(esp_wifi_set_storage(WIFI_STORAGE_RAM));
    // Init dummy AP
    ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_AP));
    ESP_ERROR_CHECK(esp_wifi_start());
    ESP_ERROR_CHECK(esp_wifi_set_ps(WIFI_PS_NONE));
    xTaskCreate(&spam_task, "spam_task", 2048, NULL, 5, NULL);
}

void loop() {}

```

A faire:

1. Analysez le code ci-dessus
2. Modifier le(s) nom(s) des points d'accès muets générés par le programme

6.2 Sniffer de WiFi

Dans cette section, nous allons construire un **renifleur WiFi** qui fonctionne en mode **promiscuité**. Le sniffer scanne tous les canaux WiFi (13 en France) à la recherche de trames MAC. Lorsqu'il capture la trame, il analyse le contenu de l'en-tête à la recherche des adresses de l'expéditeur et du destinataire. Ensuite, il gère un compteur qui stocke différentes adresses.



```
typedef struct {
    unsigned frame_ctrl:16;
    unsigned duration_id:16;
    uint8_t addr1[6]; /* receiver address */
    uint8_t addr2[6]; /* sender address */
    uint8_t addr3[6]; /* filtering address */
    unsigned sequence_ctrl:16;
    uint8_t addr4[6]; /* optional */
} wifi_ieee80211_mac_hdr_t;

typedef struct {
    wifi_ieee80211_mac_hdr_t hdr;
    uint8_t payload[0]; /* network data ended with 4 bytes csum (CRC32) */
} wifi_ieee80211_packet_t;
```

L'image ci-dessus montre le format de la trame WiFi, elle se compose de l'en-tête et de la partie charge utile. L'en-tête contient le champ de contrôle, le champ de durée, l'emplacement de quatre adresses MAC et le champ de contrôle de séquence.

6.2.1 Les fonctions d'interface WiFi et les éléments du client renifleur

Les paquets WiFi sont capturés par l'interface WiFi fonctionnant en **mode promiscuité** défini par :

```
esp_wifi_set_promiscuous(true);
```

L'initialisation de l'interface WiFi se fait par les fonctions suivantes :

```
void wifi_sniffer_init(void)
{
    nvs_flash_init();
    tcpip_adapter_init();
    ESP_ERROR_CHECK( esp_event_loop_init(event_handler, NULL) );
    wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
    ESP_ERROR_CHECK( esp_wifi_init(&cfg) );
    ESP_ERROR_CHECK( esp_wifi_set_country(&wifi_country) ); /* set country for channel range [1,13] */
    ESP_ERROR_CHECK( esp_wifi_set_storage(WIFI_STORAGE_RAM) );
    ESP_ERROR_CHECK( esp_wifi_set_mode(WIFI_MODE_NULL) );
    ESP_ERROR_CHECK( esp_wifi_start() );
    esp_wifi_set_promiscuous(true);
    esp_wifi_set_promiscuous_rx_cb(&wifi_sniffer_packet_handler);
}
```

Lorsqu'une trame arrive, la fonction **wifi_sniffer_packet_handler** est invoquée. Notez que dans ce cas, seules les trames de gestion sont sélectionnées.

```
void wifi_sniffer_packet_handler(void* buff, wifi_promiscuous_pkt_type_t type)
{
    if (type != WIFI_PKT_MGMT)
        return;
    const wifi_promiscuous_pkt_t *ppkt = (wifi_promiscuous_pkt_t *)buff;
    const wifi_ieee80211_packet_t *ipkt = (wifi_ieee80211_packet_t *)ppkt->payload;
    const wifi_ieee80211_mac_hdr_t *hdr = &ipkt->hdr;
    ..
}
```

La partie en-tête est reçue dans la partie **ipkt->hdr** du buff(er). Le **canal reniflé** et la puissance du signal reçu sont disponibles dans **ppkt->rx_ctrl.channel** et **ppkt->rx_ctrl.rssi**.

Notre programme analyse l'en-tête reçu et compare l'adresse source (`hdr->addr2`) avec les trames déjà reçues avec la même adresse.
 Seule la nouvelle adresse est conservée et stockée dans la table MAC `uint8_t tmac[512][6]` ; c'est une partie de `udp union` et `pack struct` présentée ci-dessous.
 Les octets supplémentaires [6] et [7] de chaque élément de table sont utilisés pour stocker la valeur RSSI et le numéro de canal correspondants.

```
union
{
  uint8_t frame[4120]; // 512*(7) + 2*6 + 3*4 =
  struct
  {
    int ti; int minRSSI; int maxRSSI;
    uint8_t minmac[6];
    uint8_t maxmac[6];
    uint8_t tmac[512][8]; // -RSSI , channel
  } pack;
} udp; // UDP packet
```

La valeur de `udp.pack.ti` est un compteur qui indique le nombre d'adresses MAC, de RSSI et de numéros de canal différents stockés dans la table `tmac[]`.

L'exécution de l'ensemble du programme est activée par le `wifi_sniffer_init()`; dans la fonction `setup()` et synchronisé par le code suivant dans la tâche `loop()`.

```
vTaskDelay(WIFI_CHANNEL_SWITCH_INTERVAL/portTICK_PERIOD_MS);
wifi_sniffer_set_channel(channel);
channel = (channel % WIFI_CHANNEL_MAX) + 1;
```

Le programme modifie le numéro du canal reniflé et indique l'intervalle de **changement de canal** qui est calculé comme suit :

```
WIFI_CHANNEL_SWITCH_INTERVAL/portTICK_PERIOD_MS
```

Les adresses MAC collectées sont rafraîchies cycliquement pour suivre l'évolution de la présence des appareils dans la zone donnée qui est délimitée par la force du signal (de -40 à -120) fournie dans la phase initiale de l'exécution du programme.

5.2.2 Code complet

```
#include "freertos/FreeRTOS.h"
#include "esp_wifi.h"
#include "esp_wifi_types.h"
#include "esp_system.h"
#include "esp_event.h"
#include "esp_event_loop.h"
#include "nvs_flash.h"
#include "driver/gpio.h"
#define LED_GPIO_PIN 22
#define WIFI_CHANNEL_SWITCH_INTERVAL (500)
#define WIFI_CHANNEL_MAX (13)
uint8_t level = 0, channel = 1;

static wifi_country_t wifi_country = {.cc="FR", .schan = 1, .nchan = 13};

typedef struct {
  unsigned frame_ctrl:16;
  unsigned duration_id:16;
  uint8_t addr1[6]; /* receiver address */
  uint8_t addr2[6]; /* sender address */
  uint8_t addr3[6]; /* filtering address */
  unsigned sequence_ctrl:16;
  uint8_t addr4[6]; /* optional */
} wifi_ieee80211_mac_hdr_t;

typedef struct {
  wifi_ieee80211_mac_hdr_t hdr;
  uint8_t payload[0]; /* network data ended with 4 bytes csum (CRC32) */
} wifi_ieee80211_packet_t;

static esp_err_t event_handler(void *ctx, system_event_t *event);
static void wifi_sniffer_init(void);
```

```

static void wifi_sniffer_set_channel(uint8_t channel);
static const char *wifi_sniffer_packet_type2str(wifi_promiscuous_pkt_type_t type);
static void wifi_sniffer_packet_handler(void *buff, wifi_promiscuous_pkt_type_t type);

esp_err_t event_handler(void *ctx, system_event_t *event)
{
    return ESP_OK;
}

void wifi_sniffer_init(void)
{
    nvs_flash_init();
    tcpip_adapter_init();
    ESP_ERROR_CHECK( esp_event_loop_init(event_handler, NULL) );
    wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
    ESP_ERROR_CHECK( esp_wifi_init(&cfg) );
    ESP_ERROR_CHECK( esp_wifi_set_country(&wifi_country) ); /* set country for channel range [1, 13]
*/
    ESP_ERROR_CHECK( esp_wifi_set_storage(WIFI_STORAGE_RAM) );
    ESP_ERROR_CHECK( esp_wifi_set_mode(WIFI_MODE_NULL) );
    ESP_ERROR_CHECK( esp_wifi_start() );
    esp_wifi_set_promiscuous(true);
    esp_wifi_set_promiscuous_rx_cb(&wifi_sniffer_packet_handler);
}

void wifi_sniffer_set_channel(uint8_t channel)
{
    esp_wifi_set_channel(channel, WIFI_SECOND_CHAN_NONE);
}

const char * wifi_sniffer_packet_type2str(wifi_promiscuous_pkt_type_t type)
{
    switch(type) {
        case WIFI_PKT_MGMT: return "MGMT";
        case WIFI_PKT_DATA: return "DATA";
        default:
            case WIFI_PKT_MISC: return "MISC";
    }
}

uint8_t tmac[100][6]; int ti=0; int match=0;
uint8_t minmac[6],maxmac[6];

bool clearmac()
{
    int i=0;
    for(i=0;i<ti;i++) memset(tmac[i],0x00,6);
    ti=0;
    return 0;
}

bool cmpmac(uint8_t *mac1,uint8_t *mac2)
{
    int i=0;
    for(i=0;i<6;i++) if(mac1[i]!=mac2[i]) return false;
    return true;
}

void wifi_sniffer_packet_handler(void* buff, wifi_promiscuous_pkt_type_t type)
{
    if (type != WIFI_PKT_DATA)
        return;
    const wifi_promiscuous_pkt_t *ppkt = (wifi_promiscuous_pkt_t *)buff;
    const wifi_ieee80211_packet_t *ipkt = (wifi_ieee80211_packet_t *)ppkt->payload;
    const wifi_ieee80211_mac_hdr_t *hdr = &ipkt->hdr;
    printf("PACKET TYPE=%s, CHAN=%02d, RSSI=%02d, "
        " ADDR1=%02x:%02x:%02x:%02x:%02x:%02x, "
        " ADDR2=%02x:%02x:%02x:%02x:%02x:%02x, "
        " ADDR3=%02x:%02x:%02x:%02x:%02x:%02x\n",
        wifi_sniffer_packet_type2str(type),
        ppkt->rx_ctrl.channel,
        ppkt->rx_ctrl.rssi,
        /* ADDR1 */
        hdr->addr1[0],hdr->addr1[1],hdr->addr1[2],
        hdr->addr1[3],hdr->addr1[4],hdr->addr1[5],
        /* ADDR2 */

```

```

    hdr->addr2[0],hdr->addr2[1],hdr->addr2[2],
    hdr->addr2[3],hdr->addr2[4],hdr->addr2[5],
    /* ADDR3 */
    hdr->addr3[0],hdr->addr3[1],hdr->addr3[2],
    hdr->addr3[3],hdr->addr3[4],hdr->addr3[5]
);
//printf(" number addr2 =%d\n",ti);

match=0;
for(int j=0;j<ti;j++)
{
    if(cmpmac((uint8_t *)hdr->addr2,tmac[j])) match=1;
    else continue;
}
if(match==0)
{
    if(ti==99)ti=0;
    memcpy(tmac[ti],(uint8_t *)hdr->addr2,6);
    if(hdr->addr1[0]!= 0xFF) ti++; // we may exclude broadcast frames
}
printf(" addr2 number=%d\n",ti);
}

// the setup function runs once when you press reset or power the board
void setup() {
    // initialize digital pin 5 as an output.
    Serial.begin(115200);
    delay(10);
    wifi_sniffer_init();
    pinMode(LED_GPIO_PIN, OUTPUT);
}

void loop() {
    //Serial.print("inside loop");
    delay(1000); // wait for a second
    if (digitalRead(LED_GPIO_PIN) == LOW)
        digitalWrite(LED_GPIO_PIN, HIGH);
    else
        digitalWrite(LED_GPIO_PIN, LOW);
    vTaskDelay(WIFI_CHANNEL_SWITCH_INTERVAL/portTICK_PERIOD_MS);
    wifi_sniffer_set_channel(channel);
    channel = (channel % WIFI_CHANNEL_MAX) + 1;
}

```

La sortie:

```

..
PACKET TYPE=DATA, CHAN=01, RSSI=-81, ADDR1=01:80:c2:00:00:13, ADDR2=78:81:02:31:08:b0,
ADDR3=78:81:02:31:08:b0
    addr2 number=10
PACKET TYPE=DATA, CHAN=01, RSSI=-44, ADDR1=78:81:02:31:08:b0, ADDR2=c8:c9:a3:d1:96:60,
ADDR3=78:81:02:31:08:b0
    addr2 number=10
PACKET TYPE=DATA, CHAN=01, RSSI=-83, ADDR1=ff:ff:ff:ff:ff:ff, ADDR2=78:81:02:31:08:b0,
ADDR3=c8:c9:a3:d1:96:60
    addr2 number=10
PACKET TYPE=DATA, CHAN=01, RSSI=-45, ADDR1=78:81:02:31:08:b0, ADDR2=c8:c9:a3:d1:96:60,
ADDR3=78:81:02:31:08:b0
    addr2 number=10
...

PACKET TYPE=DATA, CHAN=01, RSSI=-45, ADDR1=78:81:02:31:08:b0, ADDR2=c8:c9:a3:d1:96:60,
ADDR3=78:81:02:31:08:b0
    addr2 number=10

```

A faire

1. Analysez le code
2. Modifier le type de trames à analyser (affichées)
3. Ajoutez un écran OLED pour afficher le nombre de stations actives différentes (addr2 avec des trames DATA)

Laboratoire 7

WiFi directe (ESP-NOW) et WiFi Long Range

7.0 Introduction

ESP-NOW est un protocole développé par Espressif, qui permet à plusieurs appareils de communiquer entre eux sans utiliser de protocoles WiFi de haut niveau. Le protocole est similaire à la connectivité sans fil 2,4 GHz à faible consommation qui est souvent déployée dans les souris sans fil. Ainsi, l'appairage entre les appareils est nécessaire avant leur communication. Une fois l'appairage effectué, la connexion est sécurisée et peer-to-peer, sans qu'aucune poignée de main ne soit nécessaire.

Cela signifie qu'après avoir couplé un appareil avec un autre, la connexion est persistante. En d'autres termes, si soudainement une de vos cartes perd l'alimentation ou se réinitialise, lorsqu'elle redémarre, elle se connectera automatiquement à son pair pour continuer la communication.

ESP-NOW prend en charge les fonctionnalités suivantes :

- Communication **unicast** cryptée et non cryptée ;
- Appareils pairs mixtes chiffrés et non chiffrés ;
- Jusqu'à 250 octets de charge utile peuvent être transportés ;
- Envoi d'une fonction de rappel paramétrable pour informer la couche application de succès ou échec de la transmission.

La technologie ESP-NOW présente également les limitations suivantes :

- Pairs cryptés limités. 10 pairs chiffrés au maximum sont pris en charge en mode **STA** ; 6 au maximum en **SoftAP** ou **SoftAP + STA** ;
- Plusieurs pairs non chiffrés sont pris en charge, cependant, leur nombre total doit être inférieur à 20, y compris les pairs chiffrés ;
- La charge utile est limitée à 250 octets.

En termes simples, **ESP-NOW** est un **protocole de communication rapide** qui peut être utilisé pour échanger de petits messages (jusqu'à 250 octets) entre les cartes ESP32.

ESP-NOW est très polyvalent et vous pouvez avoir une communication **unidirectionnelle ou bidirectionnelle** dans différentes configurations.

Le protocole ESP-NOW nous permet d'envoyer les **trames WiFi identifiées par l'adresse MAC**. Le code suivant montre comment obtenir l'adresse MAC de la carte.

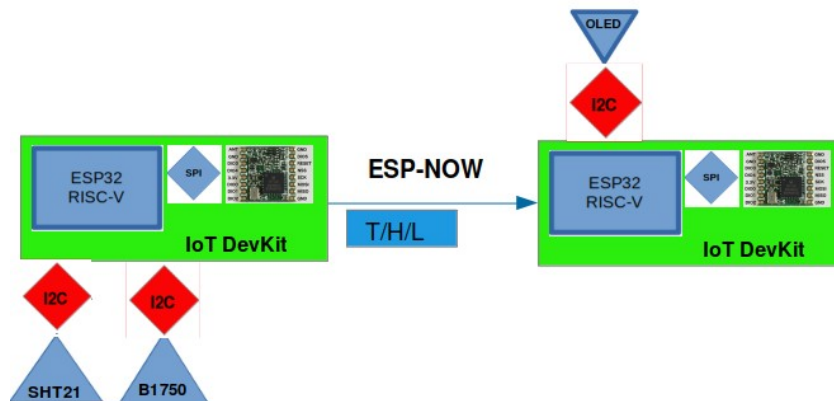
```
#include "WiFi.h"
void setup()
{
  Serial.begin(9600);
  WiFi.mode(WIFI_MODE_STA);
  Serial.println();
  Serial.print("My MAC address is: ");
  Serial.println(WiFi.macAddress());
}

void loop()
{
}

My MAC address is: A0:76:4E:1C:11:88
```


7.1 Exemple simple serveur-client

Notre premier exemple montre comment envoyer et recevoir les paquets **ESP-NOW**. L'expéditeur (Terminal) capture deux valeurs du capteur **SHT21** : la température et l'humidité et les envoie dans un paquet ESP-NOW



7.1.1 L'émetteur

```
#include <esp_now.h>
#include <WiFi.h>

// REPLACE WITH YOUR RECEIVER MAC Address
uint8_t broadcastAddress[] = {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF};
// Structure example to send data
// Must match the receiver structure
union
{
    {
        uint8_t frame[16];
        int sensor[4];
    } pack;
}

esp_now_peer_info_t peerInfo;

// callback when data is sent
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
    Serial.print("\r\nLast Packet Send Status:\t");
    Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" : "Delivery Fail");
}

void setup() {
    // Init Serial Monitor
    Serial.begin(115200);

    // Set device as a Wi-Fi Station
    WiFi.mode(WIFI_STA);
    // Init ESP-NOW
    if (esp_now_init() != ESP_OK) {
        Serial.println("Error initializing ESP-NOW");
        return;
    }
    // Once ESPNow is successfully Init, we will register for Send CB to
    // get the status of Transmitted packet
    esp_now_register_send_cb(OnDataSent);
    // Register peer
    memcpy(peerInfo.peer_addr, broadcastAddress, 6);
    peerInfo.channel = 0;
    peerInfo.encrypt = false;
    // Add peer
    if (esp_now_add_peer(&peerInfo) != ESP_OK) {
        Serial.println("Failed to add peer");
        return;
    }
}

void loop() {
    // Set values to send
    pack.sensor[0] = random(1, 20);
    pack.sensor[1] = random(1, 90);
    // Send message via ESP-NOW
    esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *) pack.frame, 16);
```

```

    if (result == ESP_OK) {
        Serial.println("Sent with success");
    }
    else {
        Serial.println("Error sending the data");
    }
    delay(2000);
}

Sent with success

Last Packet Send Status:      Delivery Success
Sent with success

Last Packet Send Status:      Delivery Success

```

7.1.2 Le récepteur

```

#include <esp_now.h>
#include <WiFi.h>

union
{
    uint8_t  frame[16];
    int  sensor[4];
} pack;

// callback function that will be executed when data is received
void OnDataRecv(const uint8_t *mac, const uint8_t *buff, int len)
{
    memcpy(pack.frame, buff, len);
    Serial.print("Bytes received: ");
    Serial.println(len);
    Serial.print("Int: ");
    Serial.println(pack.sensor[0]);
    Serial.print("Int: ");
    Serial.println(pack.sensor[1]);
    Serial.println();
}

void setup() {
    // Initialize Serial Monitor
    Serial.begin(115200);
    // Set device as a Wi-Fi Station
    WiFi.mode(WIFI_STA);
    // Init ESP-NOW
    if (esp_now_init() != ESP_OK) {
        Serial.println("Error initializing ESP-NOW");
        return;
    }

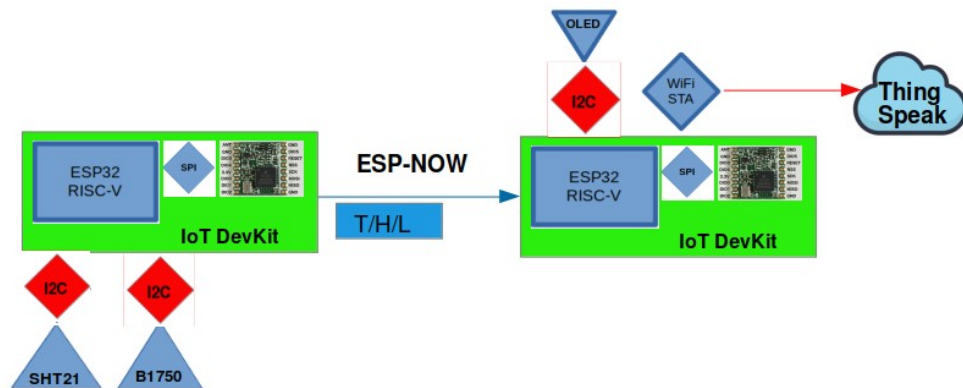
    // Once ESPNow is successfully Init, we will register for recv CB to
    // get recv packer info
    esp_now_register_recv_cb(OnDataRecv);
}

void loop() {
}

```

7.1.3 Création d'un nœud de passerelle vers ThingSpeak

L'exemple suivant présente le code pour créer un nœud récepteur-passerelle avec **deux modes WiFi**, un pour la réception des paquets ESP-NOW (**mode AP**) et un pour la transmission des données reçues via une connexion WiFi à un point d'accès donné (**mode STA**).



Le code terminal émetteur est similaire à l'exemple précédent . Il est complété par une fonction qui détermine le **numéro de canal WiFi** utilisé par la passerelle-récepteur pour communiquer avec le point d'accès.

Attention : Le canal radio utilisé par WiFi et ESP-NOW doit être le même.

```
#include <esp_now.h>
#include <WiFi.h>
#include "ThingSpeak.h"

WiFiClient client;
QueueHandle_t queue;

const char * ssid= "Livebox-08B0";
const char * pwd = "G79ji6dtEptVTPWmZP";
//static const char* ssid = "PhoneAP";
//static const char* password = "smartcomputerlab";

// callback function that will be executed when data is received
void OnDataRecv(const uint8_t *mac, const uint8_t *buff, int len)
{
    union
    {
        {
            uint8_t frame[16];
            int sensor[4];
        } pack;

        memcpy(pack.frame, buff, len);
        Serial.print("Bytes received: ");
        Serial.println(len);
        Serial.print("Int: ");
        Serial.println(pack.sensor[0]);
        Serial.print("Int: ");
        Serial.println(pack.sensor[1]);
        Serial.println();
        xQueueReset(queue);
        xQueueSend(queue, pack.frame, 0);
    }
}

void setup() {
    // Initialize Serial Monitor
    Serial.begin(115200); Serial.println();
    // Set device as a Wi-Fi Station for WiFi and AP for ESP-NOW
    WiFi.mode(WIFI_AP_STA);
    WiFi.begin(ssid, pwd);
    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.print(".");
    }
    Serial.println(); Serial.println("WiFi connected");
    ThingSpeak.begin(client); // Initialize ThingSpeak
```

```

queue = xQueueCreate(10,16); // 10 slots for 4 integer

// Init ESP-NOW
if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
    return;
}
// Once ESPNow is successfully Init, we will register for recv CB to
// get recv packer info
esp_now_register_recv_cb(OnDataRecv);
}

void loop() {
    union
    {
        uint8_t frame[16];
        int sensor[4];
    } rpack;

    xQueueReceive(queue, rpack.frame, portMAX_DELAY);
    ThingSpeak.setField(1, rpack.sensor[0]);
    ThingSpeak.setField(2, rpack.sensor[1]);
    // write to the ThingSpeak channel
    int x = ThingSpeak.writeFields(1626377, "3IN09682SQX3PT4Z" );
    if(x == 200){
        Serial.println("Channel update successful.");
    }
    else{
        Serial.println("Problem updating channel. HTTP error code " + String(x));
    }
    delay(16000);
}

Bytes received: 16
Int: 17
Int: 19

Channel update successful.
Bytes received: 16
Int: 19
Int: 20

Bytes received: 16
Int: 8
Int: 42

Bytes received: 16
Int: 10
Int: 10

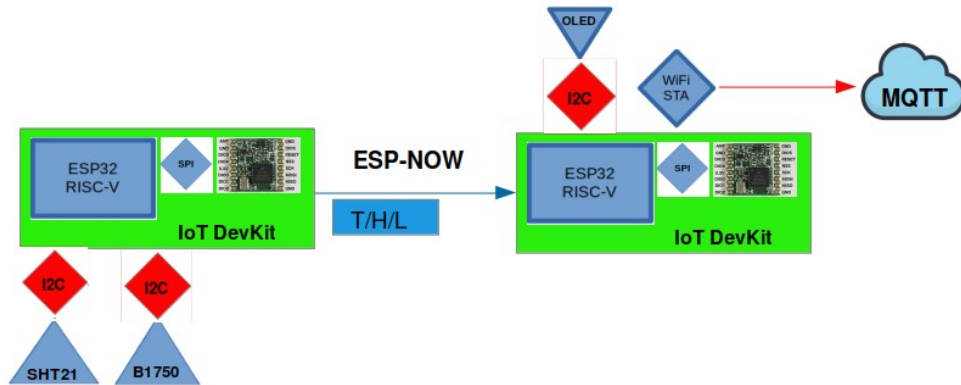
```

A faire

1. Testez les exemples présentés.
2. Ajouter un capteur physique au terminal (nœud expéditeur ESP-NOW)
3. Complétez les applications ci-dessus avec l'écran OLED attaché aux nœuds Gateway et Terminal.

7.1.4 Création d'un nœud de passerelle vers un courtier MQTT

L'exemple de code suivant implémente le nœud récepteur-passerelle (WiFi-MQTT). Les paquets ESP-NOW reçus sont retransmis au serveur MQTT "5.196.95.208" ou `test.mosquitto.org`. (port 1883)



Nous utilisons le même nœud AP pour le nœud récepteur (ThingSpeak à ESP-NOW). Notez que différents points d'accès ne peuvent être utilisés que si le numéro de canal WiFi est le même (dans notre cas, le numéro de canal 6).

```
#include <esp_now.h>
#include <WiFi.h>
#include <MQTT.h>

WiFiClient net;
MQTTClient client;

QueueHandle_t queue;

const char * ssid= "Livebox-08B0";
const char * pwd = "G79ji6dtEptVTPWmZP";
//static const char* ssid = "PhoneAP";
//static const char* password = "smartcomputerlab";

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

void OnDataRecv(const uint8_t *mac, const uint8_t *buff, int len)
{
    union
    {
        {
            uint8_t frame[16];
            int sensor[4];
        } pack;

        memcpy(pack.frame, buff, len);
        Serial.print("Bytes received: ");
        Serial.println(len);
        Serial.print("Int: ");
        Serial.println(pack.sensor[0]);
        Serial.print("Int: ");
        Serial.println(pack.sensor[1]);
        Serial.println();
        xQueueReset(queue);
        xQueueSend(queue, pack.frame, 0);
    }
}

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

void connect() {
    char cbuff[128];
    Serial.print("checking wifi...");
    while (WiFi.status() != WL_CONNECTED) {
        Serial.print(".");
    }
}
```

```

    delay(1000);
}
Serial.print("\nconnecting...");
while (!client.connect("IoT.GW1")) {
    Serial.print("."); delay(1000);
}
Serial.println("\nIoT.GW1 - connected!");
client.subscribe("/esp32_GW1/Test");
delay(1000);
}

void setup() {
    // Initialize Serial Monitor
    Serial.begin(115200); Serial.println();
    // Set device as a Wi-Fi Station for WiFi and AP for ESP-NOW
    WiFi.mode(WIFI_AP_STA);
    WiFi.begin(ssid, pwd);
    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.print(".");
    }
    delay(200);
    Serial.println();
    Serial.println("WiFi connected");
    queue = xQueueCreate(10,16); // 10 slots for 4 integer

    // Init ESP-NOW
    if (esp_now_init() != ESP_OK) {
        Serial.println("Error initializing ESP-NOW");
        return;
    }
    // Once ESPNow is successfully Init, we will register for recv CB to
    // get recv packer info
    esp_now_register_recv_cb(OnDataRecv);
    client.begin(mqttServer, net);
    client.onMessage(messageReceived);
    connect();
}

unsigned long lastMillis = 0;

void loop()
{
    union
    {
        uint8_t frame[32];
        int sensor[4];
    } rpack;

    char buff[64];

    // client.loop();
    delay(10); // <- fixes some issues with WiFi stability
    xQueueReceive(queue, rpack.frame, portMAX_DELAY);
    Serial.print("\nconnecting...");
    if (!client.connected()) { connect(); }
    if (millis() - lastMillis > 5000) { // publish a message every 5 seconds
        lastMillis = millis();
        sprintf(buff, "Temp:%d, Humi:%d\n", rpack.sensor[0], rpack.sensor[1]);
        client.publish("/esp32_GW1/Test", buff);
        Serial.println("in the loop");
        delay(2000);
    }
}

```

The code is tested with the `mosquitto_sub` command on Ubuntu host.

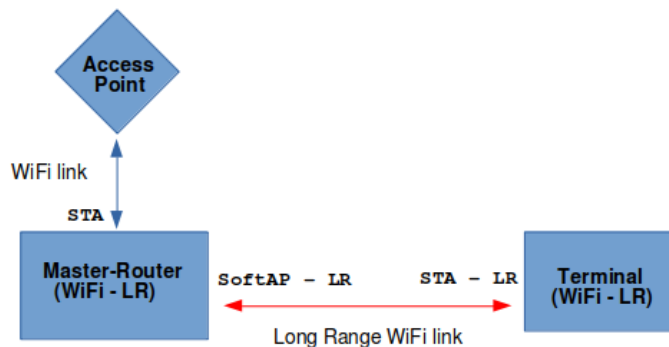
```

bako@bako-SH67H3:~$ mosquitto_sub -h "broker.emqx.io" -t /esp32_GW1/Test
Temp:56, Humi:0
Temp:67, Humi:0
Temp:30, Humi:0
Temp:33, Humi:0
Temp:85, Humi:0

```

7.2 WiFi Long Range

On ne peut pas dire pour le WiFi qu'il a une bonne portée, ou que la configuration de la couverture est facile. Avec ESP32C3 existe une extension intéressante, - une extension propriétaire à faible débit de données et à longue portée du protocole.



Ajouté discrètement aux cadres de développement officiels, le support du mode propriétaire **802.11 LR** dans l'**ESP-IDF** est devenu effectif en janvier 2017. Le mode **802.11 LR** est un mode personnalisé breveté qui peut atteindre une portée de vue de 1 km tant que la station et le **SoftAP** sont connectés à un appareil ESP32. Bien que ce type de couverture, et bien plus encore, puisse être réalisé avec des antennes directionnelles et d'autres piratages, il s'agit d'une réalisation étonnante pour une carte prête à l'emploi.

Le **scénario de communication** WiFi-LR est le suivant :

L'un des DevKits est configuré comme **maître** en **mode STA et AP**. Il se connecte d'abord au routeur traditionnel en **mode STA** et obtient une adresse IP.

Il crée ensuite un point d'accès -**SoftAP** en **mode 802.11 Long Range**, avec un **ssid** nommé "**LongRangeAP**".

Ce point d'accès n'est pas visible par les appareils standard car ils ne peuvent pas puisent détecter le protocole WiFi LR.

L'adresse IP du point d'accès dans ce mode est **192.168.4.1**. Le maître envoie cycliquement la **commande 'b'** via **UDP**, à l'adresse de diffusion **192.168.4.255**.

L'autre DevKit est configuré en **esclave**, en **mode STA**. Il se connecte au **ssid "LongRangeAP"** et obtient son IP, probablement **192.168.4.2**. A chaque fois que la commande '**b**' arrive, la RGB LED connectée au **GPIO7** change d'état. S'il n'y a pas de réception, il n'y a pas de changement.

7.2.1 Code du maître

```
#include <Arduino.h>
#include <WiFi.h>
#include <WiFiUdp.h>
#include <esp_wifi.h>
const char* ssid = "LongRange"; //AP ssid
const char* password = "smartcomputerlab"; //AP password
const char* ssidRouter = "Livebox-08B0"; //STA router ssid
const char* passwordRouter = "G79ji6dtEptVTPWmZP"; //STA router password

WiFiUDP udp;

void setup() {
  Serial.begin(115200);
  Serial.println("Master");
  //first, we start STA mode and connect to router
  WiFi.mode(WIFI_AP_STA);
  WiFi.begin(ssidRouter, passwordRouter);
  //Wifi connection
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }
}
```

```

Serial.println("Router WiFi connected");
Serial.print("IP address: ");
Serial.println(WiFi.localIP());
//second, we start AP mode with LR protocol
//This AP ssid is not visible with our regular devices
WiFi.mode( WIFI_AP );//for AP mode
//here config LR mode
int a= esp_wifi_set_protocol( WIFI_IF_AP, WIFI_PROTOCOL_LR );
if (a==0)
{
    Serial.println(" ");
    Serial.print("Return = ");
    Serial.print(a);
    Serial.println(" , Mode LR OK!");
}
else//if some error in LR config
{
    Serial.println(" ");
    Serial.print("Return = ");
    Serial.print(a);
    Serial.println(" , Error in Mode LR!");
}
WiFi.softAP(ssid, password);
Serial.println( WiFi.softAPIP() );
Serial.println("#");//for debug
delay( 5000 );
udp.begin( 8888 );
}

void loop()
{
    udp.beginPacket( { 192, 168, 4, 255 }, 8888 );//send a broadcast message
    udp.write( 'b' );//the payload

    if ( !udp.endPacket() ){
        Serial.println("NOT SEND!");
        delay(100);
        ESP.restart(); // When the connection is bad, the TCP stack refuses to work
    }
    else{
        Serial.println("SEND IT!!");
    }
    delay( 1000 );//wait a second for the next message
}

```

Résultat d'exécution :

```

Master
.Router WiFi connected
IP address: 192.168.1.62

Return = 0 , Mode LR OK!
192.168.4.1
#
SEND IT!!
SEND IT!!
SEND IT!!
SEND IT!!
SEND IT!!

```

7.2.1 Code de l'esclave

```

#include <Arduino.h>
#include <WiFi.h>
#include <WiFiUdp.h>
#include <esp_wifi.h>

const char* ssid = "LongRange";//AP ssid
const char* password = "smartcomputerlab";//AP password
const char* ssidRouter = "Livebox-08B0";//STA router ssid
const char* passwordRouter = "G79ji6dtEptVTPWmZP";//STA router password

WiFiUDP udp;

```



```

const char *toStr( wl_status_t status ) {
    switch( status ) {
        case WL_NO_SHIELD: return "No shield";
        case WL_IDLE_STATUS: return "Idle status";
        case WL_NO_SSID_AVAIL: return "No SSID avail";
        case WL_SCAN_COMPLETED: return "Scan completed";
        case WL_CONNECTED: return "Connected";
        case WL_CONNECT_FAILED: return "Failed";
        case WL_CONNECTION_LOST: return "Connection lost";
        case WL_DISCONNECTED: return "Disconnected";
    }
    return "Unknown";
}

void setup() {
    Serial.begin(115200);
    Serial.println( "Slave" );
    //first, we start STA mode and connect to router
    WiFi.mode( WIFI_AP_STA );
    WiFi.begin(ssidRouter,passwordRouter);
    //Wifi connection
    while (WiFi.status() != WL_CONNECTED)
    {
        delay(500);
        Serial.print(".");
    }
    Serial.println("Router WiFi connected");
    Serial.print("IP address: ");
    Serial.println(WiFi.localIP());
    //We start STA mode with LR protocol
    //This ssid is not visible with our regular devices
    WiFi.mode( WIFI_STA );//for STA mode
    //if mode LR config OK
    int a= esp_wifi_set_protocol( WIFI_IF_STA, WIFI_PROTOCOL_LR );
    if (a==0)
    {
        Serial.println(" ");
        Serial.print("Return = ");
        Serial.print(a);
        Serial.println(" , Mode LR OK!");
    }
    else//if some error in LR config
    {
        Serial.println(" ");
        Serial.print("Return = ");
        Serial.print(a);
        Serial.println(" , Error in Mode LR!");
    }

    WiFi.begin(ssid, password);//this ssid is not visible
    //Wifi connection, we connect to master
    while (WiFi.status() != WL_CONNECTED)
    {
        delay(500);
        Serial.print(".");
    }
    Serial.println("WiFi connected");
    Serial.print("IP address: ");
    Serial.println(WiFi.localIP());
    delay(5000);
    udp.begin( 8888 );
}

void loop() {
    //problems with connection
    if ( WiFi.status() != WL_CONNECTED )
    {
        Serial.println( "|" );
        int tries = 0;
        WiFi.begin( ssid, password );
        while( WiFi.status() != WL_CONNECTED ) {
            tries++;
            if ( tries == 5 )
                return;
            Serial.println( toStr( WiFi.status() ) );
            delay( 1000 );
        }
    }
}

```

```

    }
    Serial.print( "Connected " );
    Serial.println( WiFi.localIP() );
}
//if connection OK, execute command 'b' from master
int size = udp.parsePacket();
if ( size == 0 )
    return;
char c = udp.read();
if ( c == 'b' ){
    Serial.println("RECEIVED!");
    Serial.println(millis());
}
udp.flush();
}

```

A faire :

1. Tester le codes de deux **nœuds**
2. Ajouter un capteur coté maître
3. Ajouter un écran OLED coté esclave

Discussion :

Comment transmettre les données reçues vers un point d'accès standard – un bridge UART vers une autre carte ESP32 ?

Laboratoire 8

Programmation OTA – Over The Air

8.0 Introduction

La **programmation OTA** permet télécharger un nouveau programme sur ESP32 **via Wi-Fi** au lieu de forcer l'utilisateur à connecter l'ESP32 à un ordinateur via USB .

La fonctionnalité OTA est extrêmement utile s'il n'y a pas d'accès physique au module ESP. Cela réduit le temps passé à mettre à jour chaque module ESP pendant la maintenance. Une caractéristique importante d'OTA est qu'un emplacement central unique peut envoyer une mise à jour à plusieurs ESP partageant le même réseau.

Le seul inconvénient est que vous devez ajouter du code supplémentaire pour OTA à chaque programme que vous téléchargez, afin que vous puissiez utiliser OTA dans la prochaine mise à jour.

8.0.1 Flash memory partitions

Avant de passer à la programmation, nous allons étudier les partitions de la mémoire flash d'un ESP32. La mémoire flash est divisée en plusieurs partitions logiques pour stocker divers composants. La manière typique de le faire est illustrée dans la figure suivante.

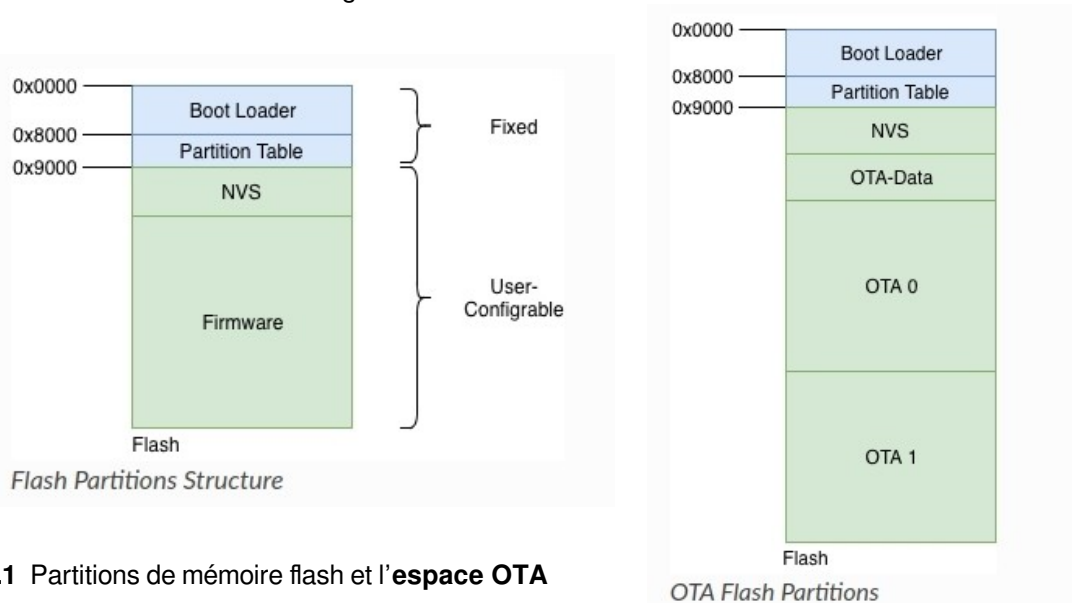


Figure 8.1 Partitions de mémoire flash et l'espace OTA

Comme le montre la figure ci-dessus, la structure est statique jusqu'à l'adresse flash **0x9000** .

La première partie du flash contient le code de démarrage, qui est immédiatement suivi de la table de partition.

La table de partition définit comment le reste de l'espace flash doit être interprété. Typiquement, une installation aura au moins 1 partition **NVS** (Non Volatile Storage - **identifiants Wi-Fi**, ..) et une partition pour le code utilisateur

8.0.2 Mécanisme OTA

Pour les mises à niveau du code, on utilise un schéma de **partition active-passive**. Comme illustré dans la figure ci-dessus deux partitions flash sont réservées au composant '**firmware**', La partition **OTA-Data** se souvient laquelle d'entre elles est la partition active.

Les changements d'état typiques qui se produisent dans le flux de travail de mise à niveau du code OTA sont illustrés à la figure 8.2.

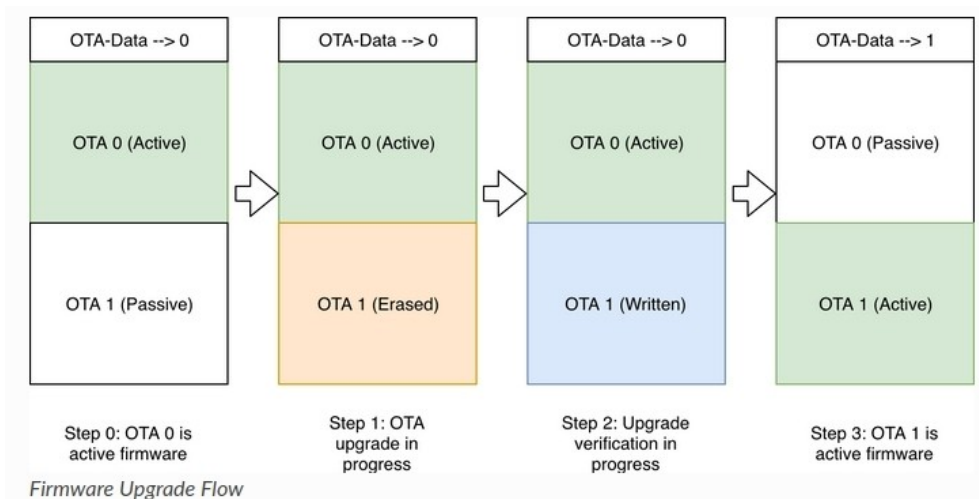


Figure 8.2 Flux de la mise à niveau du code dans la mémoire flash

1. **Étape 0** : OTA 0 est le **code actif**. La partition **OTA-Data** stocke ces informations comme on peut le voir.
2. **Étape 1** : Le processus de mise à niveau du code commence. La partition passive est identifiée, effacée et un nouveau code est en cours d'écriture sur la **partition OTA 1**.
3. **Étape 2** : La mise à niveau du code est entièrement écrite et la vérification est en cours.
4. **Étape 3** : La mise à niveau du code est réussie, la partition **OTA-Data** est mise à jour pour montrer que **OTA 1** est maintenant la **partition active**. Au prochain démarrage, le code démarrera à partir de cette partition.

8.1 Implémentation de l'OTA de base sur la carte ESP32C3

Il existe **trois façons** d'implémenter la fonctionnalité **OTA** dans ESP32.

1. **OTA de base** - Les mises à jour en direct sont envoyées via **Arduino IDE** ou **PlatformIO**.
2. **Web Updater OTA** - Les mises à jour en direct sont fournies via un **navigateur Web**.
3. La bibliothèque **WebOTA** permet également d'envoyer le sketch Arduino compilé (**.bin**) directement via l'**interface WEB**.

Chaque méthode a ses propres avantages. Vous pouvez les mettre en œuvre selon les besoins de votre projet.

8.1.1 Implémentation - Basic OTA

Pour commencer, téléchargez le **micrologiciel OTA de base** via un port série. Il s'agit d'une étape obligatoire pour pouvoir effectuer de futures mises à jour - téléchargements via WiFi. Dans la phase suivante, vous pouvez télécharger de nouveaux programmes sur ESP32 à partir d'Arduino IDE **via air - WiFi** avec une adresse IP.

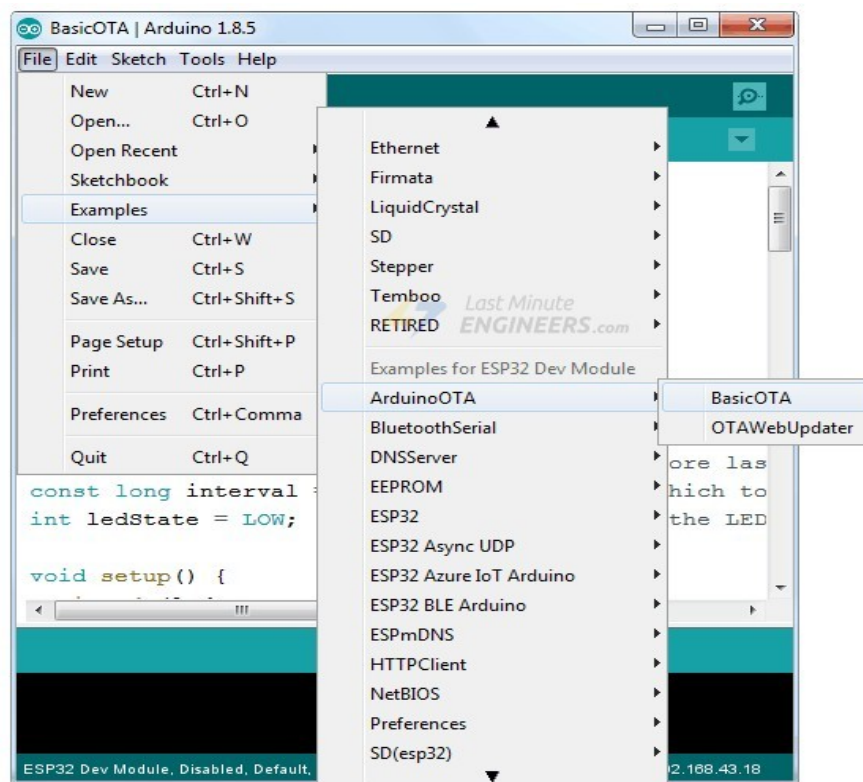


Figure 8.3 Sélection de code OTA de base

Avant de télécharger le code, vous devez fournir quelques modifications pour le faire fonctionner. Vous devez modifier les deux variables suivantes avec vos informations d'identification réseau, afin qu'ESP32 puisse établir une connexion avec le réseau existant.

```
const char* ssid = ".....";  
const char* password = ".....";
```

Une fois que vous avez terminé, téléchargez le code suivant via un câble USB.

```
#include <WiFi.h>  
#include <ESPmDNS.h>  
#include <WiFiUdp.h>  
#include <ArduinoOTA.h>  
const char* ssid = "PhoneAP";  
const char* password = "smartcomputerlab";
```

```

void setup() {
  Serial.begin(9600);
  Serial.println("Booting");
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  while (WiFi.waitForConnectResult() != WL_CONNECTED) {
    Serial.println("Connection Failed! Rebooting...");
    delay(5000);
    ESP.restart();
  }
  ArduinoOTA.onStart([]() {
    String type;
    if (ArduinoOTA.getCommand() == U_FLASH)
      type = "sketch";
    else // U_SPIFFS
      type = "filesystem";
    Serial.println("Start updating " + type);
  })
  .onEnd([]() {
    Serial.println("\nEnd");
  })
  .onProgress([](unsigned int progress, unsigned int total) {
    Serial.printf("Progress: %u%%\r", (progress / (total / 100)));
  })
  .onError([](ota_error_t error) {
    Serial.printf("Error[%u]: ", error);
    if (error == OTA_AUTH_ERROR) Serial.println("Auth Failed");
    else if (error == OTA_BEGIN_ERROR) Serial.println("Begin Failed");
    else if (error == OTA_CONNECT_ERROR) Serial.println("Connect Failed");
    else if (error == OTA_RECEIVE_ERROR) Serial.println("Receive Failed");
    else if (error == OTA_END_ERROR) Serial.println("End Failed");
  });
  ArduinoOTA.begin();
  Serial.println("Ready");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
}

void loop() {
  ArduinoOTA.handle();
}

```

Affichage sur le terminal :

```

Booting
Ready
IP address: 192.168.1.65

```

Ouvrez maintenant le moniteur série à un débit en bauds de 115200. Si tout va bien, l'adresse IP dynamique obtenue de votre routeur s'affichera. Écrivez le.

8.1.2 Télécharger un nouveau code via WiFi

Maintenant, téléchargeons un nouveau programme.

Il est nécessaire d'ajouter le code pour OTA dans chaque croquis que vous téléchargez. Sinon, vous perdrez la capacité OTA et ne pourrez plus effectuer de futurs téléchargements en direct. Il est donc recommandé de modifier le code ci-dessus pour inclure votre nouveau code.

À titre d'exemple, nous inclurons une simple esquisse RGB LED - rouge dans le code OTA de base. N'oubliez pas de modifier les variables **ssid** et **password** avec vos identifiants réseau.

```

#include <ESPmDNS.h>
#include <WiFiUdp.h>
#include <ArduinoOTA.h>
#include <Adafruit_NeoPixel.h>
#define NUMPIXELS 1
#define PIN 7 // build-in LED - 7
Adafruit_NeoPixel pixels(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);
//const char* ssid = "PhoneAP";
//const char* password = "smartcomputerlab";
const char* ssid = "Livebox-08B0"; //STA router ssid
const char* password = "G79ji6dtEptVTPWmZP"; //STA router password
unsigned long previousMillis = 0; // will store last time LED was updated
const long interval = 1000; // interval at which to blink (milliseconds)
int ledState = LOW; // ledState used to set the LED

```

```

void setup() {
  Serial.begin(115200);
  Serial.println("Booting");
  pixels.begin();
  for(int i=0;i<12;i++)pixels.setPixelColor(i, pixels.Color(0, 150, 0));
  pixels.show();
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  while (WiFi.waitForConnectResult() != WL_CONNECTED) {
    Serial.println("Connection Failed! Rebooting...");
    delay(5000); ESP.restart();
  }
  ArduinoOTA.onStart([]() {
    String type;
    if (ArduinoOTA.getCommand() == U_FLASH)
      type = "sketch";
    else // U_SPIFFS
      type = "filesystem";
    // NOTE: if updating SPIFFS this would be the place to unmount SPIFFS using SPIFFS.end()
    Serial.println("Start updating " + type);
  })
  .onEnd([]() {
    Serial.println("\nEnd");
  })
  .onProgress([](unsigned int progress, unsigned int total) {
    Serial.printf("Progress: %u%%\r", (progress / (total / 100)));
  })
  .onError([](ota_error_t error) {
    Serial.printf("Error[%u]: ", error);
    if (error == OTA_AUTH_ERROR) Serial.println("Auth Failed");
    else if (error == OTA_BEGIN_ERROR) Serial.println("Begin Failed");
    else if (error == OTA_CONNECT_ERROR) Serial.println("Connect Failed");
    else if (error == OTA_RECEIVE_ERROR) Serial.println("Receive Failed");
    else if (error == OTA_END_ERROR) Serial.println("End Failed");
  });
  ArduinoOTA.begin();
  Serial.println("Ready");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
}

void loop() {
  ArduinoOTA.handle();
  //loop to blink without delay
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillis >= interval) {
    // save the last time you blinked the LED
    previousMillis = currentMillis;
    // if the LED is off turn it on and vice-versa:
    ledState = not(ledState);
    // set the LED with the ledState of the variable:
    if (ledState== LOW)
    {
      pixels.setPixelColor(0, pixels.Color(0, 150, 0)); pixels.show();
    }
    else
    {
      pixels.setPixelColor(0, pixels.Color(0, 0, 150)); pixels.show();
    }
  }
}

```

Attention !

Dans le programme ci-dessus, nous n'avons pas utilisé `delay()` pour faire clignoter la LED, car ESP32 met votre **programme en pause pendant `delay()`**. Si la prochaine requête OTA est générée alors qu'on attend `timeout()`, votre programme manquera cette requête.

Pour réaliser le délai, nous avons utilisé la minuterie : `currentMillis = millis()`

Une fois que vous avez copié l'esquisse ci-dessus sur votre IDE Arduino, accédez à l'option **Outils-> Port** et vous devriez obtenir la sortie suivante : `esp32-xxxxxx` à `esp_ip_address` (at LOLIN C3 MINI) pour votre carte

Si vous ne le trouvez pas, vous devrez peut-être redémarrer votre IDE.

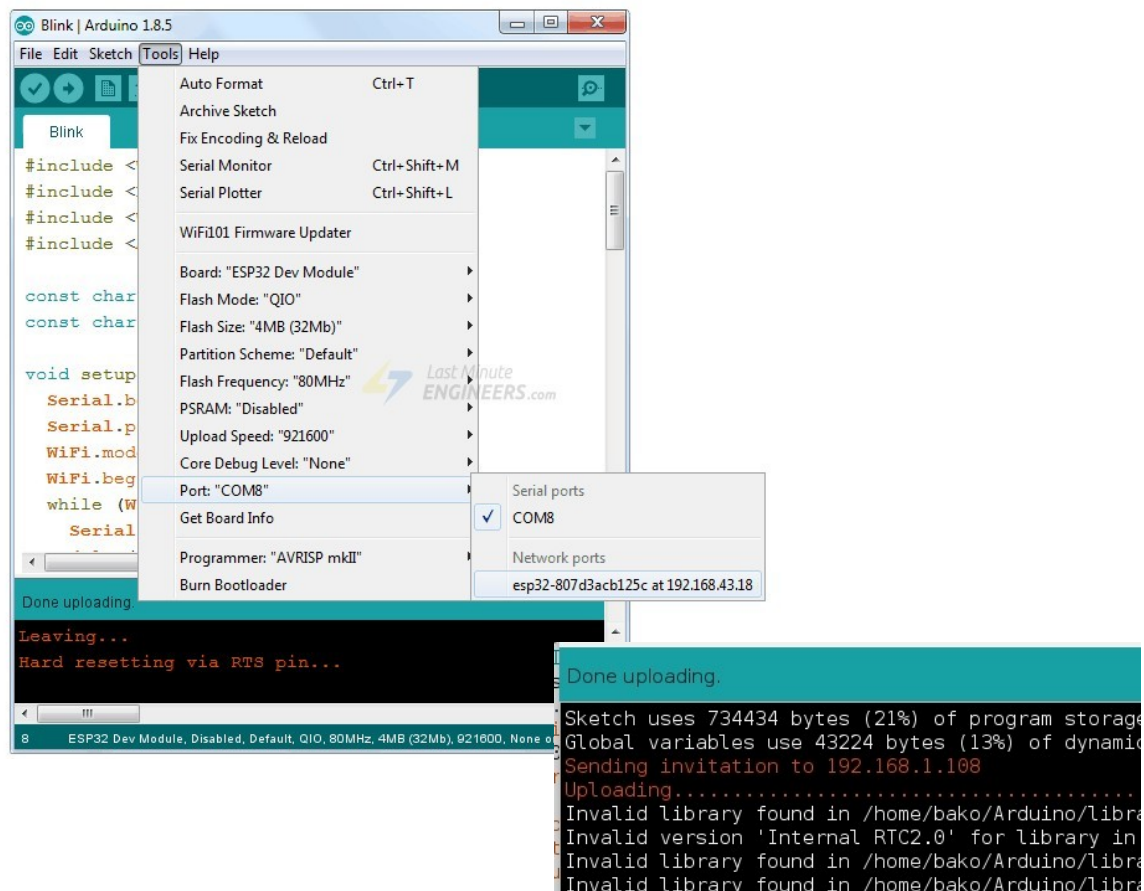


Figure 8.4 Sélection du port de transfert WiFi et de l'adresse IP

Sélectionnez le port et cliquez sur le bouton Télécharger. Dans quelques secondes, le nouveau programme sera téléchargé. Et vous devriez voir la RGB LED intégrée clignoter (**rouge - bleu**).

Pour **vérifier le processus**, vous pouvez modifier le programme en modifiant (par exemple) la valeur de :

```
int long const = 5000;
```

et téléchargez-le à nouveau via WiFi pour voir le résultat.

A faire :

1. Testez les programmes ci-dessus.
2. Modifier le code en ajoutant un affichage sur l'écran OLED
3. Modifiez le code en ajoutant la lecture du capteur et l'affichage sur l'écran OLED.

8.2 OTA sur ESP32 avec serveur WEB

Comme dans la solution précédente, la première étape consiste à télécharger le code contenant la routine OTA via USB. Il s'agit d'une étape obligatoire pour pouvoir effectuer de futures mises à jour/téléchargements via WiFi.

Le nouveau programme OTA crée un **serveur Web en mode STA**, accessible via un navigateur Web. Une fois que vous êtes connecté au serveur Web, vous pouvez télécharger de nouveaux programmes avec la routine OTA.

Vous pouvez maintenant télécharger de nouveaux programmes sur l'ESP32 en générant et en téléchargeant le fichier `.bin` compilé dans l'environnement Arduino, via un serveur Web.

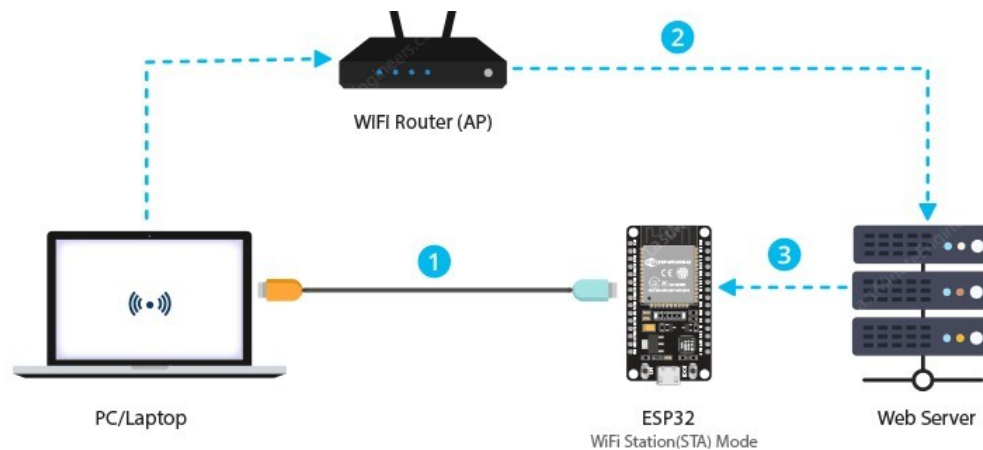


Figure 8.5 Transfert du code binaire (fichier `.bin`) par le serveur WEB vers la carte ESP32C3

Le module complémentaire ESP32 pour l'IDE Arduino est livré avec une bibliothèque OTA et un exemple `OTAWebUpdater`. Vous pouvez y accéder via **Fichier > Exemples > ArduinoOTA > OTAWebUpdater** ou via [github](#).

Pour commencer, connectez votre ESP32 à votre ordinateur et téléchargez le code ci-dessous. Comme d'habitude, vous devez proposer les identifiants WiFi de votre point d'accès, afin que ESP32 puisse établir une connexion avec le réseau existant.

8.2.1 The starting program with Webserver

```
#include <WiFi.h>
#include <WiFiClient.h>
#include <WebServer.h>
#include <ESPmDNS.h>
#include <Update.h>

const char* host = "esp32";
const char* ssid = "Livebox-08B0";
const char* password = "G79ji6dtEptVTPWmZP";

WebServer server(80);

// Login page
const char* loginIndex =
  "<form name='loginForm'>"
    "<table width='20%' bgcolor='A09F9F' align='center'>"
      "<tr>"
        "<td colspan=2>"
          "<center><font size=4><b>ESP32 Login Page</b></font></center>"
          "<br>"
        "</td>"
        "<br>"
        "<br>"
      "</tr>"
      "<td>Username:</td>"
      "<td><input type='text' size=25 name='userid'><br></td>"
    "</tr>"
  "<br>"
```

```

        "<br>"
        "<tr>"
            "<td>Password:</td>"
            "<td><input type='Password' size=25 name='pwd'><br></td>"
            "<br>"
            "<br>"
        "</tr>"
        "<tr>"
            "<td><input type='submit' onclick='check(this.form)' value='Login'></td>"
        "</tr>"
    "</table>"
"</form>"
"<script>"
    "function check(form)"
    "{"
    "if(form.userid.value=='admin' && form.pwd.value=='admin')"

```

```

if (!MDNS.begin(host)) { //http://esp32.local
  Serial.println("Error setting up MDNS responder!");
  while (1) {
    delay(1000);
  }
}
Serial.println("mDNS responder started");
/*return index page which is stored in serverIndex */
server.on("/", HTTP_GET, []() {
  server.sendHeader("Connection", "close");
  server.send(200, "text/html", loginIndex);
});
server.on("/serverIndex", HTTP_GET, []() {
  server.sendHeader("Connection", "close");
  server.send(200, "text/html", serverIndex);
});
/*handling uploading firmware file */
server.on("/update", HTTP_POST, []() {
  server.sendHeader("Connection", "close");
  server.send(200, "text/plain", (Update.hasError()) ? "FAIL" : "OK");
  ESP.restart();
}, []() {
  HTTPUpload& upload = server.upload();
  if (upload.status == UPLOAD_FILE_START) {
    Serial.printf("Update: %s\n", upload.filename.c_str());
    if (!Update.begin(UPDATE_SIZE_UNKNOWN)) { //start with max available size
      Update.printError(Serial);
    }
  } else if (upload.status == UPLOAD_FILE_WRITE) {
    /* flashing firmware to ESP*/
    if (Update.write(upload.buf, upload.currentSize) != upload.currentSize) {
      Update.printError(Serial);
    }
  } else if (upload.status == UPLOAD_FILE_END) {
    if (Update.end(true)) { //true to set the size to the current progress
      Serial.printf("Update Success: %u\nRebooting...\n", upload.totalSize);
    } else {
      Update.printError(Serial);
    }
  }
});
server.begin();
}

void loop(void) {
  server.handleClient(); delay(1);
}

```

8.2.2 L'accès au serveur WEB

Le programme **OTAWebUpdater** crée un serveur web en mode **STA** accessible via un navigateur web et qui permet de télécharger de nouveaux programmes sur votre ESP32 via WiFi.

Pour accéder au serveur Web, ouvrez le moniteur série à un débit de 115200 bauds. Si tout va bien, l'adresse IP dynamique obtenue de votre routeur (carte) s'affichera. Ensuite, chargez un navigateur et pointez-le vers l'adresse IP affichée sur le moniteur série. L'ESP32 doit servir de page Web demandant des informations de connexion (par défaut : **admin** et **admin**).

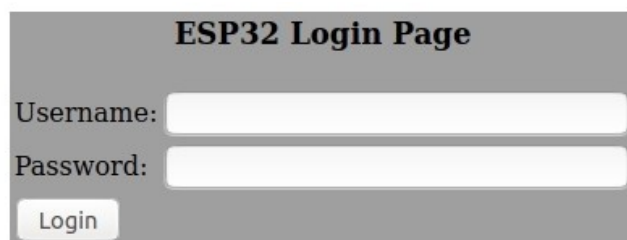


Figure 8.6 Page initiale du serveur WEB sur la carte ESP32

Si vous souhaitez modifier l'ID utilisateur et le mot de passe, modifiez le code ci-dessous dans votre programme.

```
"if (form.userid.value == 'admin' && form.pwd.value == 'admin')"
```

Une fois connecté au serveur, vous serez redirigé vers la page `/serverIndex`.

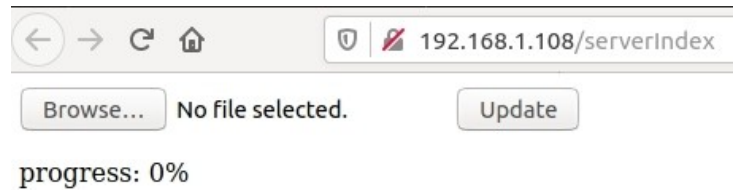


Figure 8.7 Page de recherche et de chargement d'un fichier `.bin`

Cette page vous permet de télécharger de nouveaux programmes sur votre ESP32 via WiFi. Ce nouveau programme en cours de téléchargement doit être au format binaire `.bin`.

8.2.3 Télécharger un nouveau programme via WiFi (`.bin`)

```
#include <WiFi.h>
#include <WiFiClient.h>
#include <WebServer.h>
#include <ESPmDNS.h>
#include <Update.h>
const char* host = "esp32";
const char* ssid = "Livebox-08B0";
const char* password = "G79ji6dtEptVTPWmZP";
//variables for blinking an LED with Millis
const int led = 25; // ESP32 Pin to which onboard LED is connected
unsigned long previousMillis = 0; // will store last time LED was updated
const long interval = 1000; // interval at which to blink (milliseconds)
int ledState = LOW; // ledState used to set the LED
WebServer server(80);
/* Style */
String style =
"<style>#file-input,input{width:100%;height:44px;border-radius:4px;margin:10px auto;font-size:15px}"
"input{background:#f1f1f1;border:0;padding:0 15px}body{background:#3498db;font-family:sans-"
"serif;font-size:14px;color:#777}"
"#file-input{padding:0;border:1px solid #ddd;line-height:44px;text-"
"align:left;display:block;cursor:pointer}"
"#bar,#prgbar{background-color:#f1f1f1;border-radius:10px}#bar{background-"
"color:#3498db;width:0%;height:10px}"
"form{background:#fff,max-width:258px;margin:75px auto;padding:30px;border-radius:5px;text-"
"align:center}"
".btn{background:#3498db;color:#fff;cursor:pointer}</style>";
/* Login page */
String loginIndex =
"<form name=loginForm>"
"<h1>ESP32 Login</h1>"
"<input name=userid placeholder='User ID'> "
"<input name=pwd placeholder=Password type=Password> "
"<input type=submit onclick=check(this.form) class=btn value=Login></form>"
"<script>"
"function check(form) {"
"if(form.userid.value=='admin' && form.pwd.value=='admin'){"
"{window.open('/serverIndex')}"
"else{"
"{alert('Error Password or Username')}"
"}"
"</script>" + style;

String serverIndex =
"<script src='https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js'></script>"
"<form method='POST' action='#' enctype='multipart/form-data' id='upload_form'>"
"<input type='file' name='update' id='file' onchange='sub(this)' style=display:none>"
"<label id='file-input' for='file'> Choose file...</label>"
"<input type='submit' class=btn value='Update'>"
"<br><br>"
"<div id='prg'></div>"
"<br><div id='prgbar'><div id='bar'></div></div><br></form>"
"<script>"
"function sub(obj){"
"var fileName = obj.value.split('\\\\\\\\');"
"document.getElementById('file-input').innerHTML = ' ' + fileName[fileName.length-1];"
"};"
"$('form').submit(function(e){"
```

```

"e.preventDefault();"
"var form = $('#upload_form')[0];"
"var data = new FormData(form);"
"$ajax({"
"url: '/update',"
"type: 'POST',"
"data: data,"
"contentType: false,"
"processData: false,"
"xhr: function() {"
"var xhr = new window.XMLHttpRequest();"
"xhr.upload.addEventListener('progress', function(evt) {"
"if (evt.lengthComputable) {"
"var per = evt.loaded / evt.total;"
"$('#prg').html('progress: ' + Math.round(per*100) + '%');"
"$('#bar').css('width', Math.round(per*100) + '%');"
"}"
"}, false);"
"return xhr;"
"},"
"success: function(d, s) {"
"console.log('success!') "
"},"
"error: function (a, b, c) {"
"}"
});"
});"
"</script>" + style;

void setup(void) {
  Serial.begin(9600);pinMode(led, OUTPUT);
  WiFi.begin(ssid, password);Serial.println("");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);Serial.print(".");
  }
  Serial.println("");Serial.print("Connected to ");
  Serial.println(ssid);Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
  /*use mdns for host name resolution*/
  if (!MDNS.begin(host)) { //http://esp32.local
    Serial.println("Error setting up MDNS responder!");
    while (1) { delay(1000);}
  }
  Serial.println("mDNS responder started");
  /*return index page which is stored in serverIndex */
  server.on("/", HTTP_GET, []() {
    server.sendHeader("Connection", "close");
    server.send(200, "text/html", loginIndex);
  });
  server.on("/serverIndex", HTTP_GET, []() {
    server.sendHeader("Connection", "close");
    server.send(200, "text/html", serverIndex);
  });
  /*handling uploading firmware file */
  server.on("/update", HTTP_POST, []() {
    server.sendHeader("Connection", "close");
    server.send(200, "text/plain", (Update.hasError()) ? "FAIL" : "OK");
    ESP.restart();
  }, []() {
    HTTPUpload& upload = server.upload();
    if (upload.status == UPLOAD_FILE_START) {
      Serial.printf("Update: %s\n", upload.filename.c_str());
      if (!Update.begin(UPDATE_SIZE_UNKNOWN)) { //start with max available size
        Update.printError(Serial);
      }
    } else if (upload.status == UPLOAD_FILE_WRITE) {
      /* flashing firmware to ESP*/
      if (Update.write(upload.buf, upload.currentSize) != upload.currentSize) {
        Update.printError(Serial);
      }
    } else if (upload.status == UPLOAD_FILE_END) {
      if (Update.end(true)) { //true to set the size to the current progress
        Serial.printf("Update Success: %u\nRebooting...\n", upload.totalSize);
      } else {
        Update.printError(Serial);
      }
    }
  }
}

```

```

    }
  });
  server.begin();
}

void loop(void) {
  server.handleClient(); delay(1);
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;
    ledState = not(ledState);
    digitalWrite(led, ledState);
  }
}
}

```

8.2.4 Générer un fichier .bin dans l'IDE Arduino

Afin de télécharger un nouveau sketch sur l'ESP32, nous devons d'abord générer le binaire compilé .bin de votre programme. Pour ce faire, sélectionnez **Sketch > Export** le binaire compilé

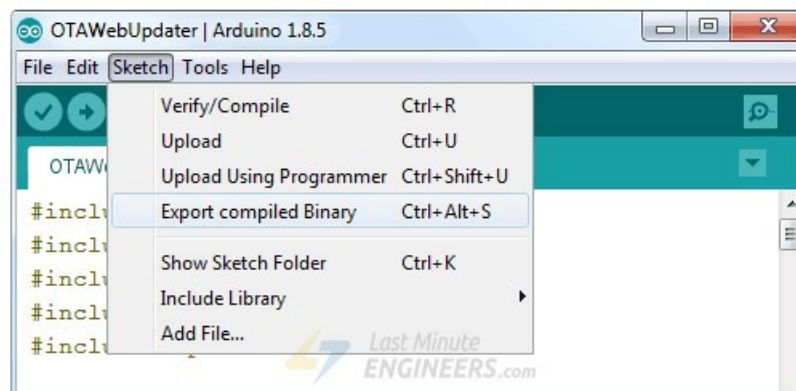


Figure 1.8 Génération de code binaire pour transfert vers la carte ESP32

8.2.5 Télécharger un nouveau sketch en direct sur l'ESP32

Une fois le fichier .bin généré, vous êtes maintenant prêt à télécharger le nouveau code sur l'ESP32 via WiFi. Ouvrez la page **/serverIndex** dans votre navigateur. Cliquez sur **Choose File...** Sélectionnez le fichier .bin généré (dans le même répertoire que votre croquis .ino), puis cliquez sur **Update** à jour.

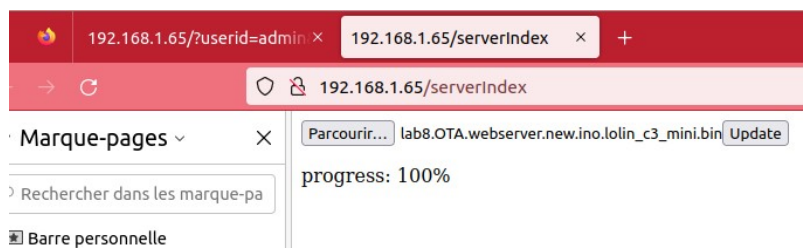


Figure 8.9 Transfert du code binaire vers la carte ESP32

Dans quelques secondes, le nouveau programme sera téléchargé. Et vous devriez voir la LED RGB intégrée changer le couleur.

A faire

1. Testez les programmes ci-dessus.
2. Modifier le code en ajoutant un affichage sur l'écran OLED
3. Modifiez le code en ajoutant la lecture du capteur et l'affichage sur l'écran OLED.

8.3 Implémenter OTA avec la bibliothèque WebOTA

La solution précédente nécessite l'insertion du code HTML dans la page WEB créée pour le transfert du code `.bin` vers la carte ESP32. La bibliothèque **WebOTA** promet de simplifier cette préparation.

Vous devez utiliser le **port 8080** et le chemin par défaut `/webota`, appelez simplement `handle_webota`) à partir de votre boucle principale.

Si vous souhaitez modifier les valeurs par défaut, vous devez ajouter un appel supplémentaire dans votre configuration. Vous devez également configurer quelques variables globales pour spécifier vos paramètres réseau.

Le seul inconvénient est que les déclarations avec un long `delay()` dans votre boucle peuvent empêcher certaines choses de fonctionner correctement et ne sont pas une bonne idée. Si vous en avez, vous pouvez remplacer tous vos appels `delay()` par `webota_delay()`, ce qui empêchera le système d'ignorer les demandes de mise à jour.

Le code initial doit être chargé par l'environnement Arduino. Pour effectuer une mise à jour, accédez simplement à la carte ESP32 avec un navigateur Web et utilisez le numéro de port et le chemin corrects. De là, vous pouvez télécharger une nouvelle **image binaire (.bin)** créée dans l'IDE Arduino.

Attention :

Le code à charger n'est pas authentifié. Cela signifie que n'importe qui peut télécharger du code sur votre ESP. Cela peut être acceptable sur un réseau privé, mais sur Internet, c'est certainement problématique.

8.3.1 Code initial

```
#include <WebOTA.h>
#define LED_PIN 25
const char* host      = "ESP-OTA"; // Used for MDNS resolution
const char* ssid = "Livebox-08B0";
const char* password = "G79ji6dtEptVTPWmZP";

void setup() {
    Serial.begin(9600);
    pinMode(LED_PIN, OUTPUT);
    init_wifi(ssid, password, host);
    // Defaults to 8080 and "/webota" , webota.init(80, "/update");
}

void loop() {
    int md = 5000;
    digitalWrite(LED_PIN, HIGH);
    webota.delay(md);
    digitalWrite(LED_PIN, LOW);
    webota.delay(md);
    webota.handle();
}
```

La sortie sur le terminal :

```
Connecting to Wifi.
Connected to 'Livebox-08B0'

IP address   : 192.168.1.54
MAC address  : 7C:9E:BD:46:3A:7C
mDNS started : ESPOTA.local
WebOTA url   : http://ESPOTA.local:8080/webota
```

L'affichage est à la hauteur de l'URL **WebOTA**. Le serveur est accessible avec;

192.168.1.54:8080/webota



Il vous demande de fournir l'emplacement de votre code Arduino compilé dans **.bin**.



Figure 8.10 La page initiale de chargement du code binaire depuis le serveur WEB

A faire

1. Testez l'exemple ci-dessus.
2. Modifier le code en ajoutant un affichage sur l'écran OLED
3. Modifiez le code en ajoutant la lecture du capteur et l'affichage sur l'écran OLED.

Table of Contents

0. Introduction.....	2
0.1 ESP32 Soc – une unité avancée pour les architectures IoT.....	3
0.2 Carte LOLIN PICO.....	3
0.3 Pomme-Pi C3 (IoT DevKit) une plate-forme de développement IoT.....	4
0.4 Architectures IoT – présentation par composants.....	5
0.4 L'installation de l'Arduino IDE.....	7
0.4.1 Installation des nouvelles cartes ESP32.....	7
Laboratoire 1.....	8
1.1 Premier exemple – l'affichage des données.....	8
A faire:.....	8
1.1.1 Test – scan du bus I2C.....	9
1.2 Capture et affichage des valeurs.....	9
1.2.1 Capture de la température/humidité par SHT21.....	9
Code :.....	10
A faire:.....	10
1.2.2 Capture de la luminosité par BH1750.....	11
A faire:.....	11
1.2.3 Capture de la pression/température avec capteur BMP180.....	11
A faire :.....	12
1.2.4 Capture les coordonnées GPS.....	12
A faire :.....	13
1.2.5 Capture de présence avec un capteur PIR SR602.....	13
1.2.5.1 Le code.....	13
A faire :.....	14
Laboratoire 2.....	14
Communication en WiFi et serveurs MQTT et ThingSpeak.....	14
2.1 Introduction.....	14
2.1.1 Un programme de test – scrutation du réseau WiFi.....	14
2.2.1 Mode STA – lecture d'une page WEB.....	16
A faire :.....	17
2.2.2 Mode softAP avec un serveur WEB.....	17
A faire :.....	18
2.2.3 Mode WiFi – STA et WiFiManager.....	18
Le code :.....	18
2.2.4 Envoi et réception des messages MQTT.....	19
2.2.4.1 Le code :.....	19
A faire :.....	20
2.2.5 Envoi des données sur ThingSpeak avec la connexion par WiFiManager.....	20
2.2.5.1 Voici le code.....	21
2.2.6 Réception des données de ThingSpeak.....	22
A faire :.....	23
2.2.7 Obtenir la data/l'heure du serveur NTP.....	23
A faire :.....	24
Laboratoire 3.....	25
Communication longue distance avec LoRa (<i>Long Range</i>).....	25
3.1 Introduction.....	25
3.1.1 Modulation LoRa.....	25
3.1.1.1 Fréquence LoRa en France.....	25
3.1.1.2 Facteur d'étalement en puissance de 2.....	25
3.1.1.3 Bande passante.....	25
3.1.2 Paquets LoRa.....	25
3.2 Premier exemple – émetteur et récepteur des paquets LoRa.....	27
3.2.1 Sender - code complet.....	28
3.2.2 Récepteur - code complet.....	30
A faire :.....	31
3.3 onReceive() – récepteur des paquets LoRa avec une interruption.....	31
3.3.1 Code - récepteur avec un Callback.....	32
A faire :.....	33
Laboratoire 4.....	34

Développement des simples passerelles IoT (LoRa-WiFi).....	34
4.1 Passerelle LoRa-WiFi (MQTT).....	34
A faire :	36
4.2 Passerelle LoRa-WiFi (ThingSpeak).....	37
4.2.1 Le principe de fonctionnement.....	37
4.1.2 Les éléments du code.....	37
4.2.3 Code complet pour une passerelle des paquets en format de base.....	38
A faire :	40
Laboratoire 5.....	41
Protocoles UDP, TCP et Web Sockets.....	41
5.1 Sockets UDP - envoi et réception de datagrammes.....	41
5.1.1 Architecture IoT : Client-Serveur (UDP-STA).....	41
5.1.1.1 Code - Client – émetteur UDP.....	42
5.1.1.2 Code - Serveur – récepteur UDP synchrone.....	43
5.1.2 Architecture IoT avec un serveur UDP synchrone et SoftAP	44
5.1.2.1 Code - Serveur UDP avec un softAP	44
5.1.2.2 Code - Client UDP synchrone pour le serveur avec SoftAP	45
A faire.....	45
5.2.1 Sockets TCP – établissement de connexions et envoi/réception de données (segments).....	46
5.2.2 Architecture IoT avec client-serveur TCP en mode STA.....	46
5.2.2.1 Code - Serveur TCP en mode STA.....	47
5.2.2.2 Code - Client TCP	47
5.2.3 Architecture IoT -TCP avec serveur en mode SoftAP	48
5.2.3.1 Code - Serveur TCP avec SoftAP.....	49
5.2.3.2 Client TCP pour un serveur avec SoftAP	49
A faire.....	50
5.3 Web Socket.....	51
5.3.1 Architecture IoT avec client-serveur sur WebSockets.....	52
5.3.1.1 Client.....	52
5.3.1.2 Server.....	52
5.3.2.1 Code - Client Websockets minimal.....	52
5.3.2.2 Serveur Websockets minimal.....	53
Laboratoire 6.....	55
6.0 Introduction.....	55
2.1.1 Générateur des trames WiFi (balises).....	56
6.1.1.1 Code complet.....	57
A faire:.....	58
6.2 Sniffer de WiFi.....	59
6.2.1 Les fonctions d'interface WiFi et les éléments du client renifleur	59
5.2.2 Code complet.....	60
A faire.....	62
Laboratoire 7.....	63
WiFi directe (ESP-NOW) et WiFi Long Range.....	63
7.0 Introduction.....	63
7.1 Exemple simple serveur-client.....	64
7.1.1 L'émetteur.....	64
7.1.2 Le récepteur.....	65
7.1.3 Création d'un nœud de passerelle vers ThingSpeak.....	66
A faire.....	67
7.1.4 Création d'un nœud de passerelle vers un courtier MQTT	68
7.2 WiFi Long Range.....	70
7.2.1 Code du maître.....	70
7.2.1 Code de l'esclave.....	71
A faire :	73
Discussion :	73
Laboratoire 8.....	74
Programmation OTA – Over The Air.....	74
8.0 Introduction.....	74
8.0.1 Flash memory partitions.....	74
8.0.2 Mécanisme OTA.....	74
8.1 Implémentation de l'OTA de base sur la carte ESP32C3.....	76
8.1.1 Implémentation - Basic OTA.....	76
8.1.2 Télécharger un nouveau code via WiFi.....	77

A faire :.....	79
8.2 OTA sur ESP32 avec serveur WEB.....	80
8.2.1 The starting program with Webserver.....	80
8.2.2 L'accès au serveur WEB.....	82
8.2.3 Télécharger un nouveau programme via WiFi(.bin).....	83
8.2.4 Générer un fichier .bin dans l'IDE Arduino.....	85
8.2.5 Télécharger un nouveau sketch en direct sur l'ESP32.....	85
A faire.....	85
8.3 Implémenter OTA avec la bibliothèque WebOTA.....	86
8.3.1 Code initial.....	86
A faire.....	87