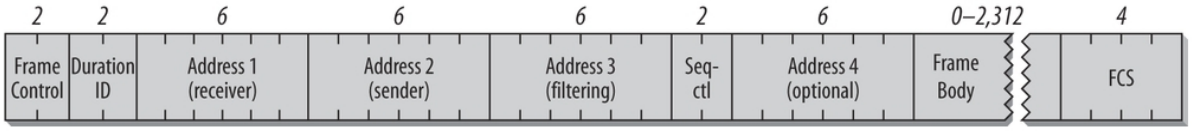
Lab 18

WiFi sniffer and beacon sender

Introduction

In this Lab we are going to analyze the basic WiFi frames and develop a simple sniffer that sends the synthetic report to the *Thingspeak* server. The connection with the Internet may be provided indirectly or directly via **Eport** Ethernet module within the **IoT DevKit** or indirectly via a **LoRa-WiFi** gateway.
The beacon sender is an independent tool to test the WiFi sniffer. This part of the Lab shows how to build the basic raw WiFi frames.
In the developed examples we are using **feeRTOS** functions to control the execution flow of the application.

18.1 WiFi frame




```
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;
```

The above picture shows the format of WiFi frame , it consists of the **header** and the **payload** part.
The **header** contains the **control** field, the **duration** field, the place for **four MAC addresses** , and the **sequence control** field.

18.2 WiFi interface functions and sniffer client elements

The WiFi packets are captured by the WiFi interface operating in **promiscuous** mode set by:

```
esp_wifi_set_promiscuous(true);.
```

The initialization of the WiFi interface is done by the following functions:

```
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);
```

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

When a frame arrives the wifi_sniffer_packet_handler function is evoked.

```
void wifi_sniffer_packet_handler(void* buff,wifi_promiscuous_pkt_type_t type)
{
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;
..
```

The header part is received in the **`ipkt->hdr`** part of the **`buff`**(er). The sniffed **channel** and the received **signal strength** are available in **`ppkt->rx_ctrl.channel`** and **`ppkt->rx_ctrl.rssi`** .
Our program analysis the received header and compares the source address (**`hdr->addr2`**) with the already received frames with the same address.
Only the new address is retained and stored in the MAC table **`uint8_t tmac[512][6];`** that is a part of the **`udp union`** and **`pack struct`** presented below.
The additional bytes **`[6]`** and **`[7]`** of each table element are used to store the corresponding **RSSI value** and **channel number**.

```
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
```

The value of **`udp.pack.ti`** is a counter that indicates the number of different MAC addresses, RSSI, and channel numbers stored in the **`tmac[]`** table.

The execution of the whole program is activated by the **`wifi_sniffer_init();`** in the **`setup()`** function and synchronized by the following code in the **`loop()`** task.

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

The program modifies the **number** of the sniffed channel and indicates the **channel switch interval that is calculated as:**

```
WIFI_CHANNEL_SWITCH_INTERVAL/portTICK_PERIOD_MS
```

The collected MAC addresses are refreshed cyclically to follow the evolution of the presence of the devices in the given area that is delimited by the signal strength (from **-40** to **-120**) provided in the initial phase of the execution of the program.

```
int setRSSI()
{
int did=-120;int  buttonState;char dbuf[32];
u8x8.clear(); u8x8.drawString(0, 0,"Setting RSSI");
while(1)
  {
  buttonState = digitalRead(buttonPin);
  if(buttonState)
    {
    did=did+10; if(did==-30) did=-120;
    sprintf(dbuf,"RSSI=%3.3d",did);u8x8.drawString(0, 2,dbuf);
    }
  else
    {
    u8x8.drawString(0, 4,"RSSI");
    sprintf(dbuf,"set to %3.3d",did);u8x8.drawString(0, 5,dbuf);
    return did;
    }
  delay(3000);
  }
}
```

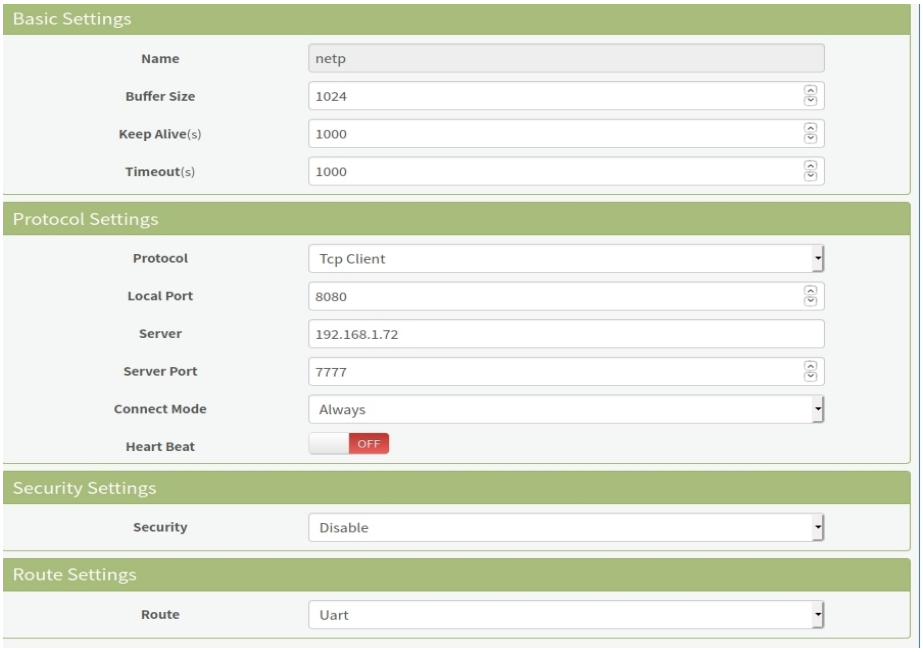
After each cycle the collected data is sent to the external server via the **Eport – Ethernet** module integrated in IoT DevKit. This is done simple by the serial link with the following code:

```
  uart.write(udp.frame,24+(udp.pack.ti*8));
```

The serial link si activated as `SoftwareSerial uart;` and configured with

```
uart.begin(57600, SWSERIAL_8N1, 12, 13); // RxD, TxD
```

Below the configuration window of the Eport module. This page is displayed when the host computer connects to its internal web server IP address, for example, 192.168.1.62. The initial page asks for the user name (admin), and password (admin).



18.2.1 TCP server

At the receiving end we have a simple TCP server code that uses traditional TCP sockets. Its reception buffer and the main loop look like this:

```
union
  {
  uint8_t frame[3096];
  struct
    {
    int ti; int minRSSI; int maxRSSI;
    uint8_t minmac[6];
    uint8_t maxmac[6];
    uint8_t tmac[512][8];  // ti in 512
    } pack;
  } udp;     // UDP packet

..
while(1)
  {
  puts("Waiting for incoming connections...");
  new_socket = accept(socket_desc, (struct sockaddr *)&client, (socklen_t*)&c);
  if (new_socket<0) { puts("accept failed"); continue; }
  else  puts("Connection accepted");
  n=read(new_socket , udp.frame , BUFSIZE);
  printf("Got %d bytes\n",n); for(i=0;i<n;i++) printf("%2.2x",udp.frame[i]);
  printf("\n\n");
  printmac();sleep(2);
  close(new_socket);
  sleep(2);
  }
.tmac[j][5]);
   }
}
```

The execution at the server side gives the following results.

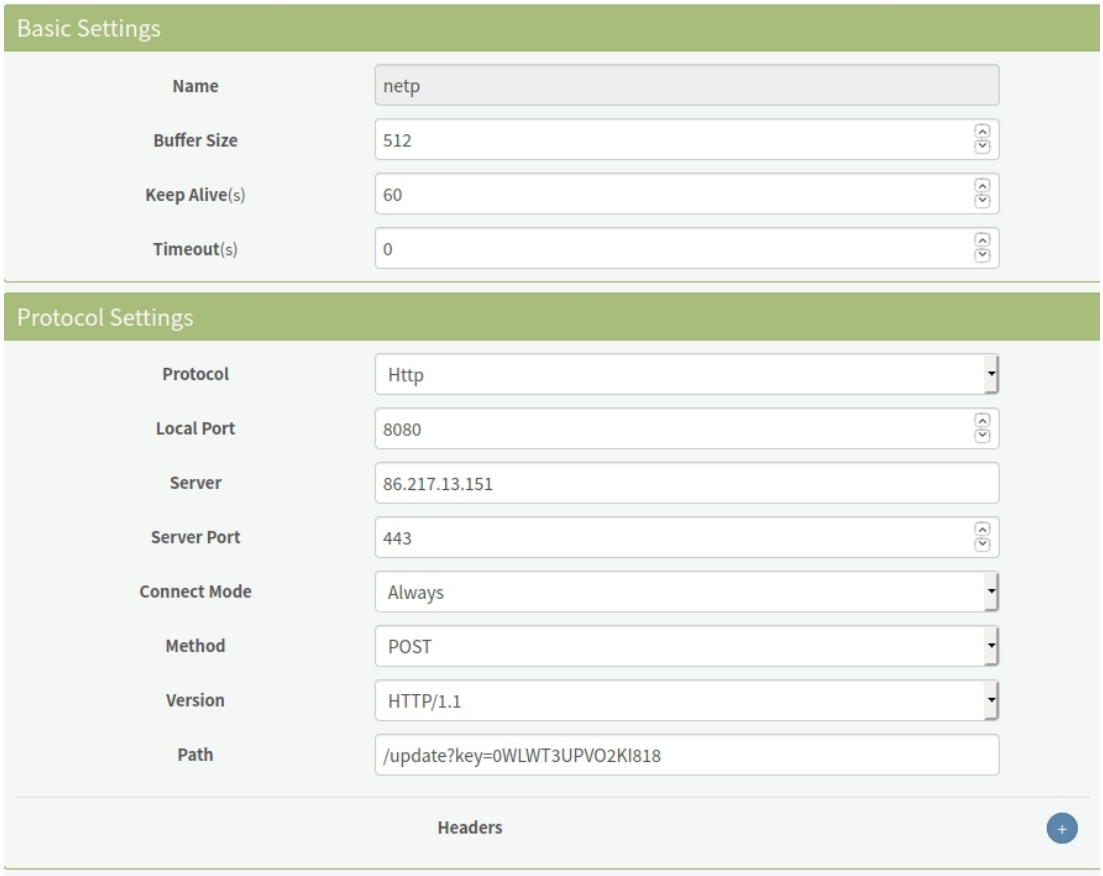
```
N=18,minRSSI=-100,maxRSSI=-36,minMAC=4e:f0:e2:4f:8f:08,maxMAC=da:a1:19:21:3c:9a
MAC=78:81:02:59:3d:36
MAC=57:01:44:fe:3b:1d
MAC=3c:c0:5e:01:62:a1
MAC=d7:4f:e0:b5:5b:01
MAC=62:a1:d7:4f:e0:b7
MAC=5b:01:e0:a1:d7:4f
MAC=e0:b4:5a:01:78:81
MAC=02:31:08:b0:46:06
MAC=78:81:02:31:08:b1
MAC=47:06:f0:82:61:e9
MAC=f9:36:5f:06:d0:ae
MAC=ec:bf:3a:82:3d:07
MAC=14:0c:76:b2:49:6f
MAC=59:0b:14:0c:76:b2
MAC=49:71:5a:0b:4e:f0
MAC=e2:4f:8f:08:5d:0b
MAC=4e:f0:e2:4f:8f:09
MAC=5c:0b:4e:f0:e2:4f
Waiting for incoming connections...
Connection accepted
```

18.3 WiFi sniffer and sender to Thingspeak server

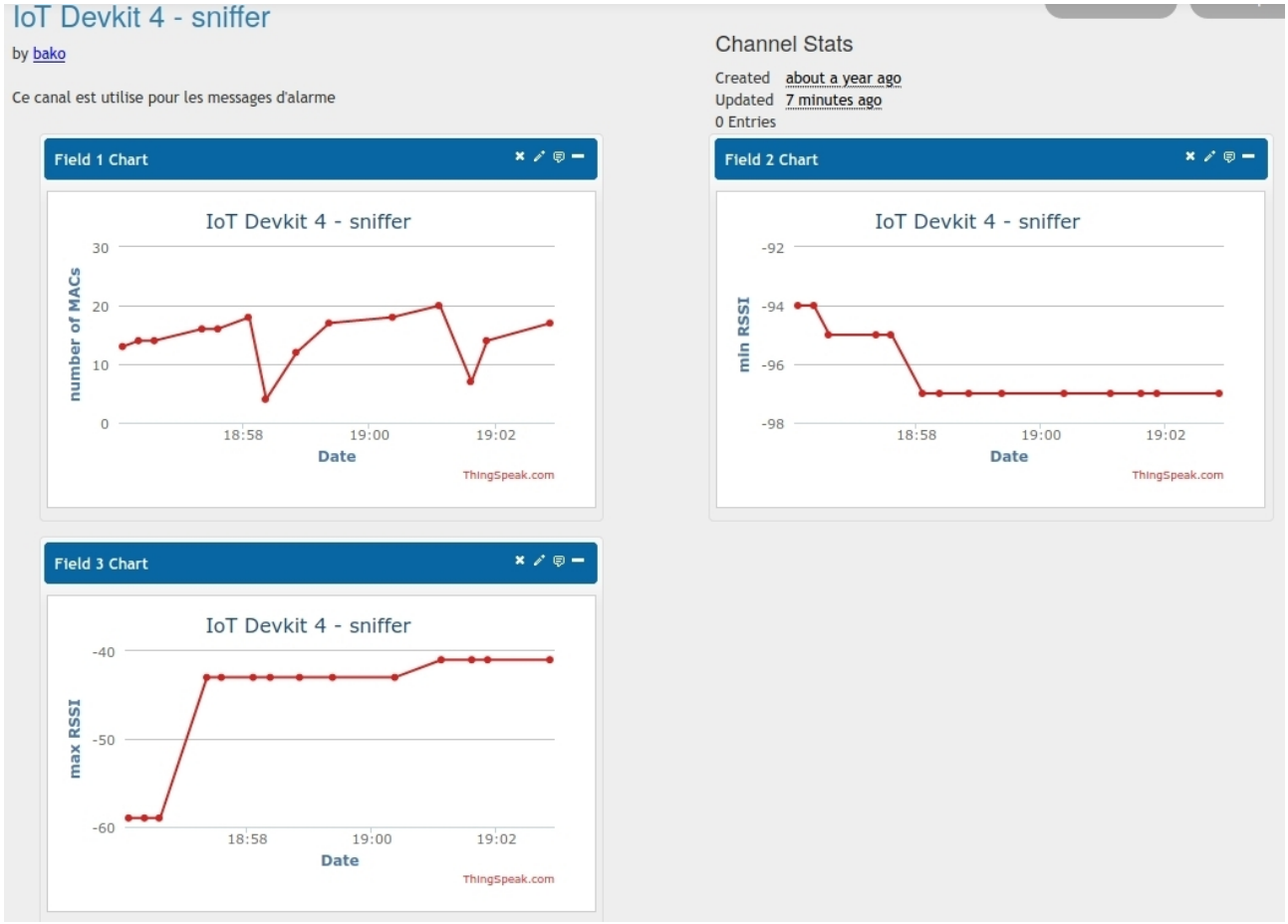
The following example shows how to send the received and selected data directly to the Thingspeak server via the Eport module. This operation is done cyclically in the Eport task.

```
void Eport_Task( void * parameter ){
while(1)
  {
  char dbuff[128],cbuff[128]; int rlen=0, i=0;
  sprintf(dbuff,"&field1=%d&field2=%d&field3=
%d",udp.pack.ti,udp.pack.minRSSI,udp.pack.maxRSSI);
  rlen=strlen(dbuff);
  uart.write((uint8_t*)dbuff,rlen+1);
  uart.flush();
  i=0;
  while (uart.available() > 0)
  {
    cbuff[i]=(char)uart.read();
    Serial.print(cbuff[i]);i++;
  }
  uart.flush();
  delay(4000);
  }
}
```

The Eport module is configured to send HTTP request directly to the Thingspeak server:



The following figure shows the resulting diagram at the Thingspeak server:


IoT Devkit 4 - sniffer
by bako
Ce canal est utilise pour les messages d'alarme
Channel Stats
Created about a year ago
Updated 7 minutes ago
0 Entries
Field 1 Chart
IoT Devkit 4 - sniffer
number of MACs
30
20
10
0
18:58
19:00
19:02
Date
ThingSpeak.com
Field 2 Chart
IoT Devkit 4 - sniffer
min RSSI
-92
-94
-96
-98
18:58
19:00
19:02
Date
ThingSpeak.com
Field 3 Chart
IoT Devkit 4 - sniffer
max RSSI
-40
-50
-60
18:58
19:00
19:02
Date
ThingSpeak.com

18.4 WiFi sniffer with LoRa sender-client and LoRa-WiFi gateway

Our MCU board (ESP32 – Heltec – WiFi-LoRa) provides us with the Lora module. In the following example we use this module to send the data from the sniffer client to the server that relays the data to the Thingspeak server.
Comparing with the previous sniffer client example we activate the LoRa module and introduce the **LoRa** task instead of the Eport task.

```
#include <SPI.h>              // include libraries
#include <LoRa.h>

union {
  uint8_t eeprom[20];
  struct {
    long frequency;
    int sf;         // spreading factor
    long sbw;       // signal bandwidth
    int denominator;   // coding rate 4/denominator
    int sw;    // synchronization word
  } lora;
} rpar;

int LoRaPar()
{
  rpar.lora.frequency=868E6;rpar.lora.sf=8;
  rpar.lora.sbw=125E3;rpar.lora.denominator=8;
  rpar.lora.sw=0xF3;
}

void LoRa_Task( void * parameter ){
while(1)
  {
  char dbuff[128],cbuff[128]; int rlen=0, i=0;
  sdf.pack.nmac=udp.pack.ti; sdf.pack.minrssi= udp.pack.minRSSI;
sdf.pack.maxrssi=udp.pack.maxRSSI;
  LoRa.beginPacket();
  sdf.pack.head[0]=0xFF;
  sdf.pack.head[1]=0x00;
  sdf.pack.head[2]=0x00;
  sdf.pack.head[3]=0x00;
  LoRa.write(sdf.frame,24);
  LoRa.endPacket();
  delay(10000);
  }
}
..
setup()
{
..
SPI.begin(5,19,27,18);  // SCK,MISO,MOSI,SS
LoRa.setPins(18,14,26);  // SS,RST,DI0
LoRaPar();
  if (!LoRa.begin(rpar.lora.frequency)) {
    Serial.println("Starting LoRa failed!");
    u8x8.drawString(0,1,"LoRa failed");
    while (1);
  }
```

```
  LoRa.setSpreadingFactor(rpar.lora.sf); // 8 by default, 11-12
  LoRa.setSignalBandwidth(rpar.lora.sbw);
  LoRa.setCodingRate4(rpar.lora.denominator);      // 5 is default

  TaskHandle_t myTask;
  xTaskCreate(
              LoRa_Task,        /* Task function. */
              "LoRa_Task",      /* String with name of task. */
              10000,              /* Stack size in words. */
              NULL,               /* Parameter passed as input of the task */
              12,                 /* Priority of the task. */
              &myTask);           /* Task handle. */

  Serial.print("Setup: created Task priority = ");
  Serial.println(uxTaskPriorityGet(myTask));
```

The complete code is given in the Annex part of this Lab.

The receiver and gateway device also operates on ESP32 board with LoRa modem. The characteristic part of the code is the WiFi task activated as follows in the **`setup()`** function.

```
void setup()
{
..
 TaskHandle_t myTask;

  xTaskCreate(
              WiFi_Task,        /* Task function. */
              "WiFi_Task",      /* String with name of task. */
              10000,              /* Stack size in words. */
              NULL,               /* Parameter passed as input of the task */
              12,                 /* Priority of the task. */
              &myTask);           /* Task handle. */

  Serial.print("Setup: created Task priority = ");
  Serial.println(uxTaskPriorityGet(myTask));
  LoRa.receive();
}
```

The code of the task is as follows:

```
void WiFi_Task( void * parameter ){
while(1)
  {
  Serial.println("In the task");
  if(WiFi.status() != WL_CONNECTED){
      WiFi.disconnect(true);
      wifiMulti.run();
      }
  Serial.println("WiFi connected");
  ThingSpeak.begin(client); // connexion (TCP) du client au serveur
  delay(1000);
  Serial.println("ThingSpeak begin");
  Serial.println("Fields update");
  if(rdf.pack.nmac>0 && rdf.pack.nmac<1000 && rdf.pack.minrssi> -150 &&
rdf.pack.minrssi< 0 && rdf.pack.maxrssi> -150 && rdf.pack.maxrssi< 0)
    {
    ThingSpeak.setField(1, (int)rdf.pack.nmac);
```

```
    ThingSpeak.setField(2, (int)rdf.pack.minrssi);
    ThingSpeak.setField(3, (int)rdf.pack.maxrssi);
    ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
    }
  delay(15000);
  }
}
```

18.5 WiFi sniffer with WiFi gateway

This example shows how to combine two ESP32 based boards to implement the WiFi sniffer and WiFI station sending the data to Thingspeak server.
The boards, HELTEC ESP32-WiFi-Lora and ESP32-Wemos-D1 board are connected via an UART bus.

18.5.1 WiFi sniffer with UART bus to second ESP32 board

The following sniffer code captures the WiFi frames and analyzes the addresses of both the **Management** and the **Data** frames.
The **Management** frames are often sent as beacons with the **randomly generated MAC** addresses. A single device can generate a number of different MAC addresses. In order to evaluate the real number of communicating devices we also capture the **Data** frames with more stable destination addresses (**`Addr1`**).

```
#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"
#include <SoftwareSerial.h>
SoftwareSerial uart;
#include <time.h>
#include <U8x8lib.h>
U8X8_SSD1306_128X64_NONAME_SW_I2C u8x8(15, 4, 16);
#define WIFI_CHANNEL_SWITCH_INTERVAL  (800)  //(500)
#define WIFI_CHANNEL_MAX               (13)
uint8_t level = 0, channel = 1;
static wifi_country_t wifi_country =  {.cc="EU", .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_event_loop_init(event_handler, NULL);
  wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
  esp_wifi_init(&cfg);
  esp_wifi_set_country(&wifi_country); // channel range [1, 13]
  esp_wifi_set_storage(WIFI_STORAGE_RAM);
  esp_wifi_set_mode(WIFI_MODE_NULL);
  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";
  case WIFI_PKT_MISC: return "MISC";
  default:  return "MISC";
  }
}

int match_a1=0, match_a2=0;

union
  {
  uint8_t frame[4120];
  struct
    {
    int ti; int minRSSI; int maxRSSI;
    uint8_t minmac[6];
    uint8_t maxmac[6];
    uint8_t tmac[512][8];  // -RSSI , channel
    } pack;
  } udp;     // source address : addr2

  union
  {
  uint8_t frame[2072];
  struct
    {
    int ti; int minRSSI; int maxRSSI;
    uint8_t minmac[6];
    uint8_t maxmac[6];
    uint8_t tmac[256][8];  // -RSSI , channel, 32 destination
    } pack;
  } dest;     // destination addresses  addr1
```

```
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;
}

int limRSSI=-100;
char dbuff[1024];char rep[512];
int val=99;

long dflat=0,dflong=0;

void wifi_sniffer_packet_handler(void* buff, wifi_promiscuous_pkt_type_t type)
{
  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;

  if((-ppkt->rx_ctrl.rssi) <32) return;
  if(ppkt->rx_ctrl.rssi < limRSSI) return;
  if(ppkt->rx_ctrl.rssi < udp.pack.minRSSI) { udp.pack.minRSSI=(int)ppkt-
>rx_ctrl.rssi; memcpy(udp.pack.minmac,hdr->addr2,6); }
  if(ppkt->rx_ctrl.rssi > udp.pack.maxRSSI) { udp.pack.maxRSSI=(int)ppkt-
>rx_ctrl.rssi; memcpy(udp.pack.maxmac,hdr->addr2,6); }
  if(type== WIFI_PKT_DATA)
    {
    if(ppkt->rx_ctrl.rssi < dest.pack.minRSSI) { dest.pack.minRSSI=(int)ppkt-
>rx_ctrl.rssi; memcpy(dest.pack.minmac,hdr->addr1,6); }
    if(ppkt->rx_ctrl.rssi > dest.pack.maxRSSI) { dest.pack.maxRSSI=(int)ppkt-
>rx_ctrl.rssi; memcpy(dest.pack.maxmac,hdr->addr1,6); }
    }
  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,
    hdr->addr1[0],hdr->addr1[1],hdr->addr1[2],
    hdr->addr1[3],hdr->addr1[4],hdr->addr1[5],
    hdr->addr2[0],hdr->addr2[1],hdr->addr2[2],
    hdr->addr2[3],hdr->addr2[4],hdr->addr2[5],
    hdr->addr3[0],hdr->addr3[1],hdr->addr3[2],
    hdr->addr3[3],hdr->addr3[4],hdr->addr3[5]
    );
 match_a2=0; match_a1=0;
 for(int j=0;j<udp.pack.ti;j++)
   {
    if(cmpmac((uint8_t *)hdr->addr2,udp.pack.tmac[j])) { match_a2=1;}
    else continue;
   }
 for(int j=0;j<dest.pack.ti;j++)
   {
    if(cmpmac((uint8_t *)hdr->addr1,dest.pack.tmac[j])) { match_a1=1;}
    else continue;
   }
```

```
 if(match_a2==0)
   {
    if(udp.pack.ti==511) { udp.pack.ti=0;}
    memcpy(udp.pack.tmac[udp.pack.ti],hdr->addr2,6);
    udp.pack.tmac[udp.pack.ti][6] = (int8_t)-ppkt->rx_ctrl.rssi;
    udp.pack.tmac[udp.pack.ti][7] = (int8_t)ppkt->rx_ctrl.channel;
    udp.pack.ti++;
   }
 if(match_a1==0)
   {
    if(dest.pack.ti==255) { dest.pack.ti=0;}
    memcpy(dest.pack.tmac[dest.pack.ti],hdr->addr1,6);
    dest.pack.tmac[dest.pack.ti][6] = (int8_t)-ppkt->rx_ctrl.rssi;
    dest.pack.tmac[dest.pack.ti][7] = (int8_t)ppkt->rx_ctrl.channel;
    dest.pack.ti++;
   }
 if(match_a1==0 || match_a1==0)
   {
    Serial.println(udp.pack.ti);
    Serial.println(dest.pack.ti-1);
    u8x8.clear();
    u8x8.drawString(0,0,"WiFi Sniffer");
    sprintf(dbuff,"nmac=%d",udp.pack.ti);
    u8x8.drawString(0,1,dbuff);
    sprintf(dbuff,"dmac=%d",dest.pack.ti-1);
    u8x8.drawString(0,2,dbuff);
    sprintf(dbuff,"minRSSI=%3.3d",udp.pack.minRSSI);
    u8x8.drawString(0,3,dbuff);
    sprintf(dbuff,"maxRSSI=%3.3d",udp.pack.maxRSSI);
    u8x8.drawString(0,4,dbuff);
    sprintf(dbuff,"MAC:%2.2x%2.2x%2.2x%2.2x%2.2x%2.2x",
    udp.pack.maxmac[0],udp.pack.maxmac[1],
    udp.pack.maxmac[2],udp.pack.maxmac[3],
    udp.pack.maxmac[4],udp.pack.maxmac[5]);
    u8x8.drawString(0,5,dbuff);
    sprintf(dbuff,"long:%2.6f",dflong);
    u8x8.drawString(0,6,dbuff);
    sprintf(dbuff,"lat:%2.6f",dflat);
    u8x8.drawString(0,7,dbuff);
    }
  delay(40);
}

const int buttonPin=0;

int setRSSI()
{
int did=-120;
int  buttonState;
u8x8.clear();
char dbuf[32];
u8x8.drawString(0, 0,"Setting RSSI");
while(1)
  {
  buttonState = digitalRead(buttonPin);
  if(buttonState) { did=did+10; if(did==-30) did=-120;
                    Serial.println(did);
```

```
                    sprintf(dbuf,"RSSI=%3.3d",did);
                    u8x8.drawString(0, 2,dbuf);
                  }
  else
     {
      Serial.println(did);
      u8x8.drawString(0, 4,"RSSI");
      sprintf(dbuf,"set to %3.3d",did);
             u8x8.drawString(0, 5,dbuf);
      return did;
      }
  delay(3000);
  }
}

union
{
  uint8_t frame[28];
  struct
    {
      uint8_t head[4];
      int nmac; int dmac;
      int minrssi;
      int maxrssi;
      long longitude;
      long latitude;
    } pack;
} sdf;

void UART_Task( void * parameter ){
while(1)
  {
  sdf.pack.nmac=(int)udp.pack.ti; sdf.pack.dmac=(int)dest.pack.ti;
  sdf.pack.minrssi= udp.pack.minRSSI; sdf.pack.maxrssi=udp.pack.maxRSSI;
  sdf.pack.head[0]=0xFF;
  sdf.pack.head[1]=0x00;
  sdf.pack.head[2]=0x00;
  sdf.pack.head[3]=0x00;
  uart.write(sdf.frame,28);
  //uart.write("hello");
  delay(10000);
  }
}

int cycle=0;

void setup() {
  Serial.begin(9600);
  uart.begin(9600, SWSERIAL_8N1, 12, 13); // RxD, TxD
  u8x8.begin();  // initialize OLED
  u8x8.setFont(u8x8_font_chroma48medium8_r);
  u8x8.clear();
  u8x8.drawString(0,0,"WiFi sniffer");
  limRSSI=setRSSI();
  udp.pack.minRSSI=0; udp.pack.maxRSSI=-100;
  delay(1000);
  wifi_sniffer_init();
  TaskHandle_t myTask;
```

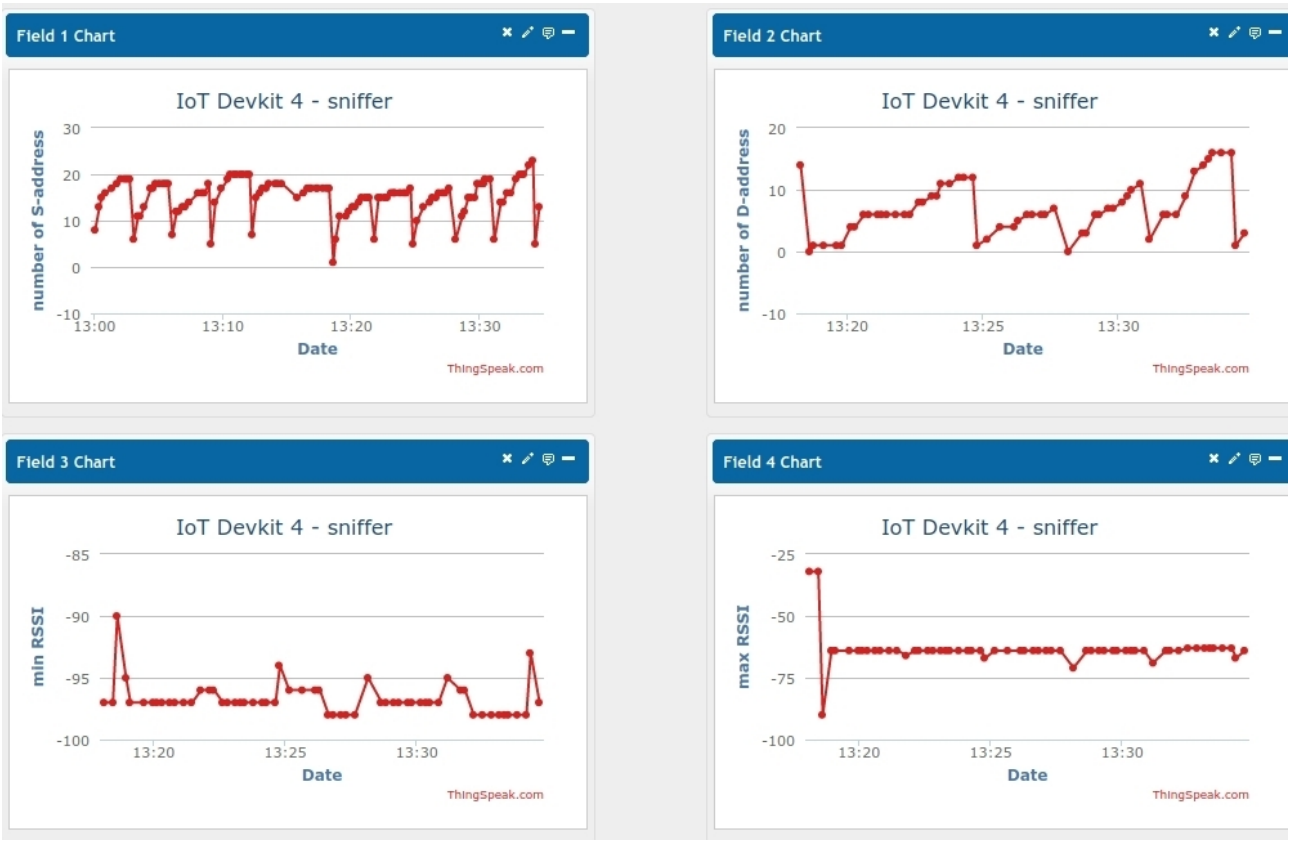
```
  xTaskCreate(
              UART_Task,        /* Task function. */
              "UART_Task",      /* String with name of task. */
              10000,              /* Stack size in words. */
              NULL,               /* Parameter passed as input of the task */
              12,                 /* Priority of the task. */
              &myTask);           /* Task handle. */
  Serial.print("Setup: created Task priority = ");
  Serial.println(uxTaskPriorityGet(myTask));
}

void loop() {
if(channel==0x01)
  {
  for(int j=0; j<udp.pack.ti;j++)
    {
    sprintf(dbuff,"MAC:%2.2x%2.2x%2.2x%2.2x%2.2x%2.2x  RSSI=-%d  CHAN=%d",
    udp.pack.tmac[j][0],udp.pack.tmac[j][1], udp.pack.tmac[j][2],
udp.pack.tmac[j][3],
    udp.pack.tmac[j][4],udp.pack.tmac[j][5],udp.pack.tmac[j][6],
udp.pack.tmac[j][7]);
    Serial.println(dbuff);
   }
  for(int j=0; j<dest.pack.ti;j++)
    {
    sprintf(dbuff,"MAC:%2.2x%2.2x%2.2x%2.2x%2.2x%2.2x  RSSI=-%d  CHAN=%d",
    dest.pack.tmac[j][0],dest.pack.tmac[j][1], dest.pack.tmac[j][2],
dest.pack.tmac[j][3],
    dest.pack.tmac[j][4],dest.pack.tmac[j][5],dest.pack.tmac[j][6],
dest.pack.tmac[j][7]);
    Serial.println(dbuff);
   }
  cycle++;
  if(cycle>6)
   {
   udp.pack.ti=0;    dest.pack.ti=0;
   udp.pack.minRSSI=0; udp.pack.maxRSSI=-100;
   cycle=0;
   }
 }
  delay(1000);
  vTaskDelay(WIFI_CHANNEL_SWITCH_INTERVAL/portTICK_PERIOD_MS);
  wifi_sniffer_set_channel(channel);
  channel = (channel % WIFI_CHANNEL_MAX) + 1;
}
```

18.5.2 ESP32 board with input UART and WiFi connection

The following code implements the reception of UART frames and transmission of the data to the Thingspeak server. The code runs on an **MH ET LIVE ESP32MiniKit** board with additional OLED screen.

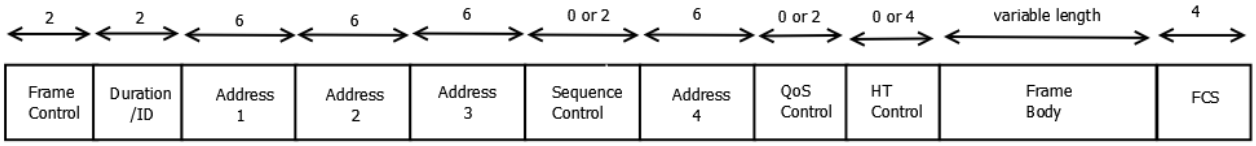
```
// compile with MH ET LIVE ESP32 MiniKit
#include "ThingSpeak.h"
#include <WiFi.h>
#include <SoftwareSerial.h>
#include <Wire.h>
#include "SSD1306.h" // alias for `#include "SSD1306Wire.h"`
SSD1306  display(0x3c, 21, 22); // SDA - 21, SCL - 22
#include <SoftwareSerial.h>
SoftwareSerial uart;
char *ssid = "PhoneAP";          //  your network SSID (name)
char * pass = "smartcomputerlab";   // your network passw
WiFiClient  client;
unsigned long myChannelNumber = 112;
const char *myWriteAPIKey="0WLWT3UPVO2KI818" ;

union
{
  uint8_t frame[28];
  struct
    {
      uint8_t head[4];
      int nmac; int dmac;
      int minrssi;
      int maxrssi;
      long longitude;
      long latitude;
    } pack;
} rdf;

char dbuff[128];

void setup() {
char dbuff[128];
Serial.begin(9600);
display.init();
//display.flipScreenVertically();
display.setFont(ArialMT_Plain_10);  // _16, _24
display.setTextAlignment(TEXT_ALIGN_LEFT);
display.drawString(0, 4, "UART-WiFi relay");
display.display();
WiFi.disconnect(true);
WiFi.mode(WIFI_STA);
WiFi.begin(ssid, pass);
while (WiFi.status() != WL_CONNECTED) {
  delay(500);Serial.print(".");
  }
IPAddress ip = WiFi.localIP();
Serial.print("IP Address: ");
Serial.println(ip);
sprintf(dbuff,"IP:%3.3d.%3.3d.%3.3d.%3.3d",ip[0],ip[1],ip[2],ip[3]);
display.drawString(0, 16, dbuff);display.display();
ThingSpeak.begin(client);
uart.begin(9600, SWSERIAL_8N1, 16, 17); // RxD, TxD
}
```

```
int i=0,j=0,n=0;

void loop()
{
char dbuff[128];
i=0;
while(uart.available() > 0)
  {
  rdf.frame[i]=uart.read();i++; n=i;
  }
if(i==28)
  {

  if(WiFi.status() != WL_CONNECTED) {  delay(500);  }
  Serial.println("WiFi connected");
  ThingSpeak.begin(client); // connexion (TCP) du client au serveur
  delay(1000);
  Serial.println("ThingSpeak begin");
  Serial.println(rdf.pack.nmac);Serial.println(rdf.pack.minrssi);
  Serial.println(rdf.pack.maxrssi);
  Serial.println("Fields update");
  if(rdf.pack.nmac>0 && rdf.pack.nmac<1000 && rdf.pack.minrssi> -150 &&
rdf.pack.minrssi< 0 && rdf.pack.maxrssi> -150 && rdf.pack.maxrssi< 0)
    {
    ThingSpeak.setField(1, (int)rdf.pack.nmac);
    ThingSpeak.setField(2, (int)rdf.pack.dmac-1);
    ThingSpeak.setField(3, (int)rdf.pack.minrssi);
    ThingSpeak.setField(4, (int)rdf.pack.maxrssi);
    ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
    display.clear();display.display();
    display.setFont(ArialMT_Plain_16);memset(dbuff,0x00,128);
    sprintf(dbuff,"SMAC:%3.3d",(int)rdf.pack.nmac);display.drawString(0, 0,
dbuff);
    display.setFont(ArialMT_Plain_16);memset(dbuff,0x00,128);
    sprintf(dbuff,"DMAC:%3.3d",(int)rdf.pack.dmac);display.drawString(0, 20,
dbuff);
    display.setFont(ArialMT_Plain_10);memset(dbuff,0x00,128);
    sprintf(dbuff,"min/max:%3.3d/%3.3d",(int)rdf.pack.minrssi,
(int)rdf.pack.maxrssi);display.drawString(0, 40, dbuff);
    display.display();
    delay(5000);uart.flush();  // must be shorter than the sender cycle !
    }
  }
delay(200);
}
```


Field 1 Chart
IoT Devkit 4 - sniffer
number of S-address
30
20
10
0
-10
13:00
13:10
13:20
13:30
Date
ThingSpeak.com
Field 2 Chart
IoT Devkit 4 - sniffer
number of D-address
20
10
0
-10
13:20
13:25
13:30
Date
ThingSpeak.com
Field 3 Chart
IoT Devkit 4 - sniffer
min RSSI
-85
-90
-95
-100
13:20
13:25
13:30
Date
ThingSpeak.com
Field 4 Chart
IoT Devkit 4 - sniffer
max RSSI
-25
-50
-75
-100
13:20
13:25
13:30
Date
ThingSpeak.com

18.6 Building WiFi frame (beacon) generator

In this part of the Lab we are going to see how to build a raw WiFi frame and send it over. This example may be used to test the previous sniffer applications.
In the frame/beacon generator application, the library elements are the same as in the sniffer code.




```
#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"
```

The essential function specification is **`esp_wifi_80211_tx()`** function that sends a raw WiFi **frame** that may carry a **beacon**.

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

A raw frame with a beacon is prepared in the following **byte** table. Note that the first byte is the **sub-type/type** code with **0x80** value that corresponds to the beacon.

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

The complete header of the MAC frame is coded as follows:

```
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
};
```

The **names** of the proposed **ssid** are prepared in `my_ssid[]` table. They are attached to the header before sending the frame on the line.

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

The constant indicating the position of the selected fields in the raw frame are give below:

```
#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` is running in the infinite `for()` loop. Before generating a new frame it waits for the `100/TOTAL_LINES/portTICK_PERIOD_MS` period. The initial place of the frame is `beacon_rick[200]` table of bytes `(uint8_t)`. This table is filled in with `beacon_raw` header followed by the name of **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);
```

The last byte of source address and BSSID is the line number to emulate the multiple broadcasting Access Points.

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

The sequnce number field is calculated as follows:

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

Finally the frame-beacon is sent by:

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

The new line number (ssid) is incremented :

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

At some point, the task execution and the esp32 itself go into **deep sleep** state. This solution allows for a lower power consumption of the device. There are no operations in the main **loop() task.**

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

The **setup()** function initializes the required functions (drivers) dealing with the WiFi interface:

```
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, NUL);
  esp_wifi_init(&cfg);
  esp_wifi_set_storage(WIFI_STORAGE_RAM);
```

We initialize a **dummy AP** to specify a channel and get WiFi hardware into a mode where we can send the actual fake beacon frames.

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

Finally we launch the **spam_task**.

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

void loop()
{
}
```

18.7 Annex – complete codes

18.7.1 Sniffer Client-Server (TCP)

The following application (**18.5.1**) is built from two parts:

- the sniffer TCP client with Eport module (**18.5.1.1**)
- the server and relay node to the Thingspeak server (**18.5.1.2**) on Ubuntu host

18.7.1.1 The complete code of the sniffer as TCP client

```
#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"
#include <SoftwareSerial.h>
#include <time.h>
#include <U8x8lib.h>  // bibliothèque à charger a partir de
// the OLED used
U8X8_SSD1306_128X64_NONAME_SW_I2C u8x8(15, 4, 16);
SoftwareSerial uart;
float dflong,dflat;

#define WIFI_CHANNEL_SWITCH_INTERVAL  (800)  //(500)
#define WIFI_CHANNEL_MAX               (13)
uint8_t level = 0, channel = 1;
static wifi_country_t wifi_country =  {.cc="EU", .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";
  }
}

 int match=0;

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

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;
}

bool cpymac(uint8_t *mac, int ind)
{
  int i=0;
  for(i=0;i<6;i++) udp.pack.tmac[ind][i]=mac[i];
}

int limRSSI=-100;
char dbuff[1024];char rep[512];
```

```
int val=99;

void wifi_sniffer_packet_handler(void* buff, wifi_promiscuous_pkt_type_t type)
{
bool newData = false;
//  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;

  if(ppkt->rx_ctrl.rssi < limRSSI) return;
  if(ppkt->rx_ctrl.rssi < udp.pack.minRSSI) { udp.pack.minRSSI=(int)ppkt-
>rx_ctrl.rssi; memcpy(udp.pack.minmac,hdr->addr2,6); }
  if(ppkt->rx_ctrl.rssi > udp.pack.maxRSSI) { udp.pack.maxRSSI=(int)ppkt-
>rx_ctrl.rssi; memcpy(udp.pack.maxmac,hdr->addr2,6); }
  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]
  );
 match=0;
 for(int j=0;j<udp.pack.ti;j++)
   {
    if(cmpmac((uint8_t *)hdr->addr2,udp.pack.tmac[j])) { match=1;}
    else continue;
   }
 if(match==0)
   {
    if(udp.pack.ti==511) udp.pack.ti=0;
    memcpy(udp.pack.tmac[udp.pack.ti],hdr->addr2,6);
    udp.pack.tmac[udp.pack.ti][6] = (int8_t)-ppkt->rx_ctrl.rssi;
    udp.pack.tmac[udp.pack.ti][7] = (int8_t)ppkt->rx_ctrl.channel;
    udp.pack.ti++;
    Serial.println(udp.pack.ti);
    u8x8.clear();
    u8x8.drawString(0,0,"WiFi Sniffer");
    sprintf(dbuff,"number=%d",udp.pack.ti);
    u8x8.drawString(0,1,dbuff);
    sprintf(dbuff,"minRSSI=%3.3d",udp.pack.minRSSI);
    u8x8.drawString(0,2,dbuff);
    sprintf(dbuff,"maxRSSI=%3.3d",udp.pack.maxRSSI);
    u8x8.drawString(0,3,dbuff);
    sprintf(dbuff,"MAC:%2.2x%2.2x%2.2x%2.2x%2.2x%2.2x",
    udp.pack.maxmac[0],udp.pack.maxmac[1],
```

```
    udp.pack.maxmac[2],udp.pack.maxmac[3],
    udp.pack.maxmac[4],udp.pack.maxmac[5]);
    u8x8.drawString(0,4,dbuff);
    sprintf(dbuff,"long:%2.6f",dflong);
    u8x8.drawString(0,5,dbuff);
    sprintf(dbuff,"lat:%2.6f",dflat);
    u8x8.drawString(0,6,dbuff);
    }
  delay(40);
}

int setRSSI()
{
int did=-120;
int  buttonState;
u8x8.clear();
char dbuf[32];
u8x8.drawString(0, 0,"Setting RSSI");
while(1)
  {
  buttonState = digitalRead(buttonPin);
  if(buttonState) { did=did+10; if(did==-30) did=-120;
                    Serial.println(did);
                    sprintf(dbuf,"RSSI=%3.3d",did);
                    u8x8.drawString(0, 2,dbuf);
                  }
  else
     {
      Serial.println(did);
      u8x8.drawString(0, 4,"RSSI");
      sprintf(dbuf,"set to %3.3d",did);
             u8x8.drawString(0, 5,dbuf);
      return did;
      }
  delay(3000);18.4.1 Sniffer Client-Server (TCP)
  }
}

int cycle=0;

void setup() {
  Serial.begin(9600);
  uart.begin(57600, SWSERIAL_8N1, 12, 13); // RxD, TxD
  u8x8.begin();  // initialize OLED
  u8x8.setFont(u8x8_font_chroma48medium8_r);
  u8x8.clear();
  u8x8.drawString(0,0,"SMTR sniffer");
  limRSSI=setRSSI(); udp.pack.minRSSI=0; udp.pack.maxRSSI=-100;
  delay(1000);
  wifi_sniffer_init();
  pinMode(LED_GPIO_PIN, OUTPUT);
}

void loop() {
  uart.flush();
 if(channel==0x01)
 {
 for(int j=0; j<udp.pack.ti;j++)
```

```
   {
   sprintf(dbuff,"MAC:%2.2x%2.2x%2.2x%2.2x%2.2x%2.2x  RSSI=-%d  CHAN=%d",
   udp.pack.tmac[j][0],udp.pack.tmac[j][1],udp.pack.tmac[j][2],
   udp.pack.tmac[j][3],
   udp.pack.tmac[j][4],udp.pack.tmac[j][5],udp.pack.tmac[j][6],
   udp.pack.tmac[j][7]);
   Serial.println(dbuff);
   }
  uart.write(udp.frame,24+(udp.pack.ti*6));  // 25 simple, + 8*number
  cycle++;
  if(cycle>6)
   {
   udp.pack.ti=0; cycle=0;
   }
 }
  delay(1000); // wait for a second
  vTaskDelay(WIFI_CHANNEL_SWITCH_INTERVAL/portTICK_PERIOD_MS);
  wifi_sniffer_set_channel(channel);
  channel = (channel % WIFI_CHANNEL_MAX) + 1;
}
```

18.7.1.2 The complete code of the TCP server (Ubuntu host)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#define BUFSIZE  3096

union
  {
  uint8_t frame[3096];
  struct
    {
    int ti; int minRSSI; int maxRSSI;
    uint8_t minmac[6];
    uint8_t maxmac[6];
    uint8_t tmac[512][8];  // ti in 512
    } pack;
  } udp;     // UDP packet

int printmac()
{
int i=0,j=0;
char dbuff[4120];
printf("N=%d,minRSSI=%d,maxRSSI=%d,minMAC=%02x:%02x:%02x:%02x:%02x:%02x,maxMAC=
%02x:%02x:%02x:%02x:%02x:%02x\n",udp.pack.ti,udp.pack.minRSSI,udp.pack.maxRSSI,
udp.pack.minmac[0],udp.pack.minmac[1],udp.pack.minmac[2],udp.pack.minmac[3],udp.
pack.minmac[4],udp.pack.minmac[5],
udp.pack.maxmac[0],udp.pack.maxmac[1],udp.pack.maxmac[2],udp.pack.maxmac[3],udp.
pack.maxmac[4],udp.pack.maxmac[5]);
for(j=0; j<udp.pack.ti;j++)
   {
   printf("MAC=%2.2x:%2.2x:%2.2x:%2.2x:%2.2x:%2.2x\n",udp.pack.tmac[j][0],
   udp.pack.tmac[j][1], udp.pack.tmac[j][2],udp.pack.tmac[j][3],
```

```
   udp.pack.tmac[j][4],udp.pack.tmac[j][5]);
   }
}

int main(int count , char *a[])
{
int socket_desc , new_socket , c, n, i;
struct sockaddr_in server , client;
socket_desc = socket(AF_INET , SOCK_STREAM , 0);
if (socket_desc == -1)
  { printf("Could not create socket"); return 1; }
server.sin_family = AF_INET;
server.sin_addr.s_addr = INADDR_ANY;
server.sin_port = htons(7777);
if( bind(socket_desc,(struct sockaddr *)&server , sizeof(server)) < 0)
  { puts("bind failed"); return 1; }
puts("bind done");
listen(socket_desc , 5);
c = sizeof(struct sockaddr_in);
while(1)
  {
  puts("Waiting for incoming connections...");
  new_socket = accept(socket_desc, (struct sockaddr *)&client, (socklen_t*)&c);
  if (new_socket<0) { puts("accept failed"); continue; }
  else  puts("Connection accepted");
  n=read(new_socket , udp.frame , BUFSIZE);
  printf("Got %d bytes\n",n); for(i=0;i<n;i++) printf("%2.2x",udp.frame[i]);
  printf("\n\n");
  printmac();sleep(2);
  close(new_socket);
  sleep(2);
  }
return 0;
}
```

Execution of the server side:

```
N=18,minRSSI=-100,maxRSSI=-36,minMAC=4e:f0:e2:4f:8f:08,maxMAC=da:a1:19:21:3c:9a
MAC=78:81:02:59:3d:36
MAC=57:01:44:fe:3b:1d
MAC=3c:c0:5e:01:62:a1
MAC=d7:4f:e0:b5:5b:01
MAC=62:a1:d7:4f:e0:b7
MAC=5b:01:e0:a1:d7:4f
MAC=e0:b4:5a:01:78:81
MAC=02:31:08:b0:46:06
MAC=78:81:02:31:08:b1
MAC=47:06:f0:82:61:e9
MAC=f9:36:5f:06:d0:ae
MAC=ec:bf:3a:82:3d:07
MAC=14:0c:76:b2:49:6f
MAC=59:0b:14:0c:76:b2
MAC=49:71:5a:0b:4e:f0
MAC=e2:4f:8f:08:5d:0b
MAC=4e:f0:e2:4f:8f:09
MAC=5c:0b:4e:f0:e2:4f
Waiting for incoming connections...
Connection accepted
```

18.7.2 Sniffer Client (HTTP)

The following code is the HTTP version that allows us to send the synthetic data directly to the Thingspeak server. Note the introduction of Eport_task that prepares the the data for the Thingspeak server.

```
#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"

#include <SoftwareSerial.h>
SoftwareSerial uart;
#include <time.h>
float dflong,dflat;
#include <U8x8lib.h>
U8X8_SSD1306_128X64_NONAME_SW_I2C u8x8(15, 4, 16);

#define LED_GPIO_PIN                    25
#define WIFI_CHANNEL_SWITCH_INTERVAL  (800)  //(500)
#define WIFI_CHANNEL_MAX               (13)

uint8_t level = 0, channel = 1;

static wifi_country_t wifi_country =  {.cc="EU", .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_event_loop_init(event_handler, NULL);
  wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
  esp_wifi_init(&cfg);
  esp_wifi_set_country(&wifi_country);
  esp_wifi_set_storage(WIFI_STORAGE_RAM);
  esp_wifi_set_mode(WIFI_MODE_NULL);
  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";
  }
}

int match=0;

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

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;
}

bool cpymac(uint8_t *mac, int ind)
{
  int i=0;
  for(i=0;i<6;i++) udp.pack.tmac[ind][i]=mac[i];
}

int limRSSI=-100;
char dbuff[1024];char rep[512];
int val=99;
```

```
void wifi_sniffer_packet_handler(void* buff, wifi_promiscuous_pkt_type_t type)
{
bool newData = false;
//  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;

  if(ppkt->rx_ctrl.rssi < limRSSI) return;
  if(ppkt->rx_ctrl.rssi < udp.pack.minRSSI) { udp.pack.minRSSI=(int)ppkt-
>rx_ctrl.rssi; memcpy(udp.pack.minmac,hdr->addr2,6); }
  if(ppkt->rx_ctrl.rssi > udp.pack.maxRSSI) { udp.pack.maxRSSI=(int)ppkt-
>rx_ctrl.rssi; memcpy(udp.pack.maxmac,hdr->addr2,6); }

  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]
  );
 match=0;
 for(int j=0;j<udp.pack.ti;j++)
   {
    if(cmpmac((uint8_t *)hdr->addr2,udp.pack.tmac[j])) { match=1;}  // cmpmac -
true if equal
    else continue;
   }
 if(match==0)
   {
    if(udp.pack.ti==511) udp.pack.ti=0;
    memcpy(udp.pack.tmac[udp.pack.ti],hdr->addr2,6);
    udp.pack.tmac[udp.pack.ti][6] = (int8_t)-ppkt->rx_ctrl.rssi;
    udp.pack.tmac[udp.pack.ti][7] = (int8_t)ppkt->rx_ctrl.channel;
    udp.pack.ti++;
    Serial.println(udp.pack.ti);
    u8x8.clear();
    u8x8.drawString(0,0,"WiFi Sniffer");
    sprintf(dbuff,"number=%d",udp.pack.ti);
    u8x8.drawString(0,1,dbuff);
    sprintf(dbuff,"minRSSI=%3.3d",udp.pack.minRSSI);
    u8x8.drawString(0,2,dbuff);
    sprintf(dbuff,"maxRSSI=%3.3d",udp.pack.maxRSSI);
    u8x8.drawString(0,3,dbuff);
    sprintf(dbuff,"MAC:%2.2x%2.2x%2.2x%2.2x%2.2x%2.2x",
    udp.pack.maxmac[0],udp.pack.maxmac[1],
```

```
    udp.pack.maxmac[2],udp.pack.maxmac[3],
    udp.pack.maxmac[4],udp.pack.maxmac[5]);
    u8x8.drawString(0,4,dbuff);
    sprintf(dbuff,"long:%2.6f",dflong);
    u8x8.drawString(0,5,dbuff);
    sprintf(dbuff,"lat:%2.6f",dflat);
    u8x8.drawString(0,6,dbuff);
    }
  delay(40);
}

const int buttonPin=0;

int setRSSI()
{
int did=-120;
int  buttonState;char dbuf[32];
u8x8.clear();
u8x8.drawString(0, 0,"Setting RSSI");
while(1)
  {
  buttonState = digitalRead(buttonPin);
  if(buttonState) { did=did+10; if(did==-30) did=-120;
                    Serial.println(did);
                    sprintf(dbuf,"RSSI=%3.3d",did);
                    u8x8.drawString(0, 2,dbuf);
                  }
  else
     {
      Serial.println(did);
      u8x8.drawString(0, 4,"RSSI");
      sprintf(dbuf,"set to %3.3d",did);
             u8x8.drawString(0, 5,dbuf);
      return did;
      }
  delay(3000);
  }
}

void Eport_Task( void * parameter ){
while(1)
  {
  char dbuff[128],cbuff[128]; int rlen=0, i=0;
  sprintf(dbuff,"&field1=%d&field2=%d&field3=%d", udp.pack.ti,
udp.pack.minRSSI,udp.pack.maxRSSI);
  rlen=strlen(dbuff);
  uart.write((uint8_t*)dbuff,rlen+1);
  uart.flush();
  i=0;
  while (uart.available() > 0)
  {
    cbuff[i]=(char)uart.read();
    Serial.print(cbuff[i]);i++;
  }
  uart.flush();
  delay(4000);
  }
}
```

```
int cycle=0;
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin 5 as an output.
  Serial.begin(9600);
  uart.begin(115200, SWSERIAL_8N1, 12, 13); // RxD, TxD
  u8x8.begin();  // initialize OLED
  u8x8.setFont(u8x8_font_chroma48medium8_r);
  u8x8.clear();
  u8x8.drawString(0,0,"SMTR sniffer");
  limRSSI=setRSSI(); udp.pack.minRSSI=0; udp.pack.maxRSSI=-100;
  delay(1000);
  wifi_sniffer_init();
  TaskHandle_t myTask;
  xTaskCreate(
              Eport_Task,        /* Task function. */
              "Eport_Task",      /* String with name of task. */
              10000,              /* Stack size in words. */
              NULL,               /* Parameter passed as input of the task */
              12,                 /* Priority of the task. */
              &myTask);           /* Task handle. */
  Serial.print("Setup: created Task priority = ");
  Serial.println(uxTaskPriorityGet(myTask));
}

void loop() {
if(channel==0x01)
  {
  for(int j=0; j<udp.pack.ti;j++)
    {
    sprintf(dbuff,"MAC:%2.2x%2.2x%2.2x%2.2x%2.2x%2.2x  RSSI=-%d  CHAN=%d",
    udp.pack.tmac[j][0],udp.pack.tmac[j][1], udp.pack.tmac[j]
[2],udp.pack.tmac[j][3],
    udp.pack.tmac[j][4],udp.pack.tmac[j][5],udp.pack.tmac[j][6],udp.pack.tmac[j]
[7]);
    Serial.println(dbuff);
   }
  cycle++;
  if(cycle>6)
   {
   udp.pack.ti=0; cycle=0;
   }
 }
  delay(1000); // wait for a second

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

18.7.3 Sniffer Client-Serve (gateway) with LoRa link

The following application consists of two parts:

- the sniffer with LoRa sender (**18.4.3.1**)
- the receiver and LoRa-Wifi gateway (**18.4.3.2**)

18.7.3.1 WiFi sniffer with LoRa sender

```
#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"
#include <SPI.h>              // include libraries
#include <LoRa.h>
#include <time.h>

#include <U8x8lib.h>
U8X8_SSD1306_128X64_NONAME_SW_I2C u8x8(15, 4, 16);

#define LED_GPIO_PIN                    25
#define WIFI_CHANNEL_SWITCH_INTERVAL  (800)  //(500)
#define WIFI_CHANNEL_MAX               (13)
uint8_t level = 0, channel = 1;

static wifi_country_t wifi_country =  {.cc="EU", .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_event_loop_init(event_handler, NULL);
  wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
  esp_wifi_init(&cfg);
  esp_wifi_set_country(&wifi_country);
  esp_wifi_set_storage(WIFI_STORAGE_RAM);
  esp_wifi_set_mode(WIFI_MODE_NULL);
  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";
  }
}

union {
  uint8_t eeprom[20];
  struct {
    long frequency;
    int sf;        // spreading factor
    long sbw;      // signal bandwidth
    int denominator;   // coding rate 4/denominator
    int sw;    // synchronization word
  } lora;
} rpar;

int LoRaPar()
{
  rpar.lora.frequency=868E6;rpar.lora.sf=8;
  rpar.lora.sbw=125E3;rpar.lora.denominator=8;
  rpar.lora.sw=0xF3;
}

int match=0;

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

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;
}

bool cpymac(uint8_t *mac, int ind)
{
  int i=0;
  for(i=0;i<6;i++) udp.pack.tmac[ind][i]=mac[i];
}

int limRSSI=-100;
char dbuff[1024];char rep[512];
int val=99;


void wifi_sniffer_packet_handler(void* buff, wifi_promiscuous_pkt_type_t type)
{
bool newData = false;
//  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;
  if(ppkt->rx_ctrl.rssi < limRSSI) return;
  if(ppkt->rx_ctrl.rssi < udp.pack.minRSSI) { udp.pack.minRSSI=(int)ppkt-
>rx_ctrl.rssi; memcpy(udp.pack.minmac,hdr->addr2,6); }
  if(ppkt->rx_ctrl.rssi > udp.pack.maxRSSI) { udp.pack.maxRSSI=(int)ppkt-
>rx_ctrl.rssi; memcpy(udp.pack.maxmac,hdr->addr2,6); }

  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]
  );

 match=0;
 for(int j=0;j<udp.pack.ti;j++)
   {
```

```
    if(cmpmac((uint8_t *)hdr->addr2,udp.pack.tmac[j])) { match=1;}  // cmpmac -
true if equal
    else continue;
   }
 if(match==0)
   {
    if(udp.pack.ti==511) udp.pack.ti=0;
    memcpy(udp.pack.tmac[udp.pack.ti],hdr->addr2,6);
    udp.pack.tmac[udp.pack.ti][6] = (int8_t)-ppkt->rx_ctrl.rssi;
    udp.pack.tmac[udp.pack.ti][7] = (int8_t)ppkt->rx_ctrl.channel;
    udp.pack.ti++;
    Serial.println(udp.pack.ti);
    u8x8.clear();
    u8x8.drawString(0,0,"WiFi Sniffer");
    sprintf(dbuff,"number=%d",udp.pack.ti);
    u8x8.drawString(0,1,dbuff);
    sprintf(dbuff,"minRSSI=%3.3d",udp.pack.minRSSI);
    u8x8.drawString(0,2,dbuff);
    sprintf(dbuff,"maxRSSI=%3.3d",udp.pack.maxRSSI);
    u8x8.drawString(0,3,dbuff);
    sprintf(dbuff,"MAC:%2.2x%2.2x%2.2x%2.2x%2.2x%2.2x",
    udp.pack.maxmac[0],udp.pack.maxmac[1],
    udp.pack.maxmac[2],udp.pack.maxmac[3],
    udp.pack.maxmac[4],udp.pack.maxmac[5]);
    u8x8.drawString(0,4,dbuff);
    sprintf(dbuff,"long:%2.6f",dflong);
    u8x8.drawString(0,5,dbuff);
    sprintf(dbuff,"lat:%2.6f",dflat);
    u8x8.drawString(0,6,dbuff);
    }
  delay(40);
}

const int buttonPin=0;

int setRSSI()
{
int did=-120;
int  buttonState;
u8x8.clear();
char dbuf[32];
u8x8.drawString(0, 0,"Setting RSSI");
while(1)
  {
  buttonState = digitalRead(buttonPin);
  if(buttonState) { did=did+10; if(did==-30) did=-120;
                    Serial.println(did);
                    sprintf(dbuf,"RSSI=%3.3d",did);
                    u8x8.drawString(0, 2,dbuf);
                  }
  else
     {
      Serial.println(did);
      u8x8.drawString(0, 4,"RSSI");
      sprintf(dbuf,"set to %3.3d",did);
             u8x8.drawString(0, 5,dbuf);
      return did;
      }
```

```
  delay(3000);
  }
}

union
{
  uint8_t frame[24];
  struct
    {
      uint8_t head[4];
      int nmac;
      int minrssi;
      int maxrssi;
      long longitude;
      long latitude;
    } pack;
} sdf;

void LoRa_Task( void * parameter ){
while(1)
  {
  char dbuff[128],cbuff[128]; int rlen=0, i=0;
  sdf.pack.nmac=udp.pack.ti; sdf.pack.minrssi= udp.pack.minRSSI;
sdf.pack.maxrssi=udp.pack.maxRSSI;
  LoRa.beginPacket();
  sdf.pack.head[0]=0xFF;
  sdf.pack.head[1]=0x00;
  sdf.pack.head[2]=0x00;
  sdf.pack.head[3]=0x00;
  LoRa.write(sdf.frame,24);
  LoRa.endPacket();
  delay(10000);
  }
}

int cycle=0;

void setup() {
  // initialize digital pin 5 as an output.
  Serial.begin(9600);
  uart.begin(115200, SWSERIAL_8N1, 12, 13); // RxD, TxD
  u8x8.begin();  // initialize OLED
  u8x8.setFont(u8x8_font_chroma48medium8_r);
  u8x8.clear();
  SPI.begin(5,19,27,18);  // SCK,MISO,MOSI,SS
  LoRa.setPins(18,14,26);  // SS,RST,DI0
  LoRaPar();

  if (!LoRa.begin(rpar.lora.frequency)) {
    Serial.println("Starting LoRa failed!");
    u8x8.drawString(0,1,"LoRa failed");
    while (1);
  }
  //LoRa.setTxPower(20, PA_OUTPUT_PA_BOOST_PIN);
  LoRa.setSpreadingFactor(rpar.lora.sf); // 8 by default, 11-12
  LoRa.setSignalBandwidth(rpar.lora.sbw);
  LoRa.setCodingRate4(rpar.lora.denominator);      // 5 is default
```

```
  u8x8.drawString(0,0,"SMTR sniffer");
  limRSSI=setRSSI(); udp.pack.minRSSI=0; udp.pack.maxRSSI=-100;
  delay(1000);
  wifi_sniffer_init();
  TaskHandle_t myTask;
  xTaskCreate(
              LoRa_Task,        /* Task function. */
              "LoRa_Task",      /* String with name of task. */
              10000,              /* Stack size in words. */
              NULL,               /* Parameter passed as input of the task */
              12,                 /* Priority of the task. */
              &myTask);           /* Task handle. */
  Serial.print("Setup: created Task priority = ");
  Serial.println(uxTaskPriorityGet(myTask));
}

void loop() {
if(channel==0x01)
  {
  for(int j=0; j<udp.pack.ti;j++)
    {
    sprintf(dbuff,"MAC:%2.2x%2.2x%2.2x%2.2x%2.2x%2.2x  RSSI=-%d  CHAN=%d",
    udp.pack.tmac[j][0],udp.pack.tmac[j][1], udp.pack.tmac[j]
[2],udp.pack.tmac[j][3],
    udp.pack.tmac[j][4],udp.pack.tmac[j][5],udp.pack.tmac[j][6],udp.pack.tmac[j]
[7]);
    Serial.println(dbuff);
   }
  cycle++;
  if(cycle>6)
   {
   udp.pack.ti=0;
   cycle=0;
   }
 }
  delay(1000); // wait for a second

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

18.7.3.2 LoRa receiver and gateway to Thingspeak server

```
#include <SPI.h>              // include libraries
#include <LoRa.h>
#include "WiFiMulti.h"
#include "ThingSpeak.h"
#include <U8x8lib.h>
U8X8_SSD1306_128X64_NONAME_SW_I2C u8x8(15, 4, 16); // data, clock, reset
const int Button = 17;

bool MOTION_DETECTED = false;

union {
  uint8_t eeprom[20];
  struct {
```

```
    long frequency;
    int sf;        // spreading factor
    long sbw;      // signal bandwidth
    int denominator;   // coding rate 4/denominator
    int sw;    // synchronization word
  } lora;
} rpar;

WiFiMulti wifiMulti;

String ssid1 =     "Livebox-08B0";
String password1 = "G79ji6dtEptVTPWmZP";
String ssid2 = "PhoneAP";          //  your network SSID (name)
String password2 = "smartcomputerlab";   // your network passw

unsigned long myChannelNumber = 112;
const char * myWriteAPIKey="0WLWT3UPVO2KI818" ;
WiFiClient  client;

union
{
  uint8_t frame[24];
  struct
    {
      uint8_t head[4];
      int nmac;
      int minrssi;
      int maxrssi;
      long longitude;
      long latitude;
    } pack;
} rdf;

uint8_t id=0x00;  // Monitor ID is zero

int LoRaPar()
{
  rpar.lora.frequency=868E6;rpar.lora.sf=8;
  rpar.lora.sbw=125E3;rpar.lora.denominator=8;
  rpar.lora.sw=0xF3;
}

void WiFi_Task( void * parameter ){
while(1)
  {
  Serial.println("In the task");
  if(WiFi.status() != WL_CONNECTED){
      WiFi.disconnect(true);
      wifiMulti.run();
      }
  Serial.println("WiFi connected");
  ThingSpeak.begin(client); // connexion (TCP) du client au serveur
  delay(1000);
  Serial.println("ThingSpeak begin");
  Serial.println("Fields update");
  if(rdf.pack.nmac>0 && rdf.pack.nmac<1000 && rdf.pack.minrssi> -150 &&
rdf.pack.minrssi< 0 && rdf.pack.maxrssi> -150 && rdf.pack.maxrssi< 0)
    {
```

```
    ThingSpeak.setField(1, (int)rdf.pack.nmac);
    ThingSpeak.setField(2, (int)rdf.pack.minrssi);
    ThingSpeak.setField(3, (int)rdf.pack.maxrssi);
    ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
    }
  delay(15000);
  }
}

void setup() {
char dbuff[40];
  Serial.begin(9600);
  u8x8.begin();  // initialize u8x8
  u8x8.setFont(u8x8_font_chroma48medium8_r);
  u8x8.clear();
  u8x8.drawString(0,0,"Monitor node");
  WiFi.mode(WIFI_STA);
  wifiMulti.addAP(ssid1.c_str(), password1.c_str());
  wifiMulti.addAP(ssid2.c_str(), password2.c_str());
  wifiMulti.run();
  if(WiFi.status() != WL_CONNECTED){
      WiFi.disconnect(true);
      wifiMulti.run();
      }
  Serial.println("WiFi connected");
  u8x8.drawString(0,1,"WiFi connected");
  ThingSpeak.begin(client); // connexion (TCP) du client au serveur
  delay(1000);
  Serial.println("ThingSpeak begin");
  Serial.println("Start LoRa Monitor");
  SPI.begin(5,19,27,18);  // SCK,MISO,MOSI,SS
  LoRa.setPins(18,14,26);  // SS,RST,DI0
  LoRaPar();
  if (!LoRa.begin(rpar.lora.frequency)) {
    Serial.println("Starting LoRa failed!");
    u8x8.drawString(0,1,"LoRa failed");
    while (1);
  }
  //LoRa.setTxPower(20, PA_OUTPUT_PA_BOOST_PIN);
  LoRa.setSpreadingFactor(rpar.lora.sf); // 8 by default, 11-12
  LoRa.setSignalBandwidth(rpar.lora.sbw);
  LoRa.setCodingRate4(rpar.lora.denominator);      // 5 is default
 // LoRa.setSyncWord(sw);          // 0xF3
  Serial.println("LoRa OK");u8x8.clear();
  u8x8.drawString(0,0,"Monitor node");
  u8x8.drawString(0,1,"LoRa started:");
  sprintf(dbuff,"freq:%d KHz",rpar.lora.frequency/1000);
  u8x8.drawString(0,2,dbuff);
  sprintf(dbuff,"bw:%d KHz",rpar.lora.sbw/1000);
  u8x8.drawString(0,3,dbuff);
  sprintf(dbuff,"sf:%d",rpar.lora.sf);
  u8x8.drawString(0,4,dbuff);
  sprintf(dbuff,"cr:4/%d",rpar.lora.denominator);
  u8x8.drawString(0,5,dbuff);
  delay(1000);

  TaskHandle_t myTask;
  xTaskCreate(
```

```
              WiFi_Task,        /* Task function. */
              "WiFi_Task",      /* String with name of task. */
              10000,              /* Stack size in words. */
              NULL,               /* Parameter passed as input of the task */
              12,                 /* Priority of the task. */
              &myTask);           /* Task handle. */
  Serial.print("Setup: created Task priority = ");
  Serial.println(uxTaskPriorityGet(myTask));
  LoRa.receive();
}

int sdata, ldata[4];
char dbuff[64];

int packetSize=0,i=0;

void loop()
{
packetSize = LoRa.parsePacket();
if(packetSize==24)
  {
  i=0;
  while (LoRa.available()) { rdf.frame[i] = LoRa.read(); i++; }
  if(rdf.pack.head[0]==0xFF)
    {
    u8x8.clear();
    u8x8.drawString(0,0,"new frame");
    sprintf(dbuff,"NMAC:%d",rdf.pack.nmac);
    u8x8.drawString(0,1,dbuff); Serial.println(dbuff);
    sprintf(dbuff,"minRSSI:%d",rdf.pack.minrssi); u8x8.drawString(0,2,dbuff);
    sprintf(dbuff,"maxRSSI:%d",rdf.pack.maxrssi);Serial.println(dbuff);
    u8x8.drawString(0,3,dbuff);
    }
  }
}
```

18.7.4 WiFi frame (beacon) generator

```
#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[] = {
  "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] > 0xfff)
      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()
{
}
```

Table of Contents

Introduction....1
18.1 WiFi frame....1
18.2 WiFi interface functions and sniffer client elements....1
18.2.1 TCP server....4
18.3 WiFi sniffer and sender to Thingspeak server....5
18.4 WiFi sniffer with LoRa sender-client and LoRa-WiFi gateway....7
18.5 WiFi sniffer with WiFi gateway....10
18.5.1 WiFi sniffer with UART bus to second ESP32 board....10
18.5.2 ESP32 board with input UART and WiFi connection....16
18.6 Building WiFi frame (beacon) generator....19
18.7 Annex – complete codes....22
18.7.1 Sniffer Client-Server (TCP)....22
18.7.1.1 The complete code of the sniffer as TCP client....22
18.7.1.2 The complete code of the TCP server (Ubuntu host)....26
18.7.2 Sniffer Client (HTTP)....28
18.7.3 Sniffer Client-Serve (gateway) with LoRa link....33
18.7.3.1 WiFi sniffer with LoRa sender....33
18.7.3.2 LoRa receiver and gateway to Thingspeak server....38
18.7.4 WiFi frame (beacon) generator....42