CONTADOR DE PERSONAS EN MODO PROMISCUO

Grupo H: @usbt0p, @LucachuTW.
Grupo Q: @santipvz, @SantiagoRR2004.

ÍNDICE

I. Introducción

En estas páginas desarrollaremos el informe acerca del proyecto para el conteo de personas. El objetivo del proyecto es el de obtener una estimación del número de personas en un lugar determinado, utilizando las direcciones MAC únicas asociadas a cada dispositivo que se envían junto a los paquetes de wifi, y pueden ser detectadas y recogidas por un dispositivo en modo promiscuo.

Uno de los objetivos del proyecto es familiarizarse con el concepto de IoT y su flujo de trabajo, por lo que se utilizará una arquitectura de sistema de 3 capas en la que se incorporará un servidor de MQTT y dispositivos sensores con chips ESP32.

Las capas de la arquitectura del proyecto son una capa de percepción o de dispositivos (la capa más baja), una capa de procesamiento y gestión de datos o capa de middleware y una capa de aplicación y servicios. Cada una de ellas utilizará su propia plataforma y tiene un proceso de desarrollo único, y en las siguientes páginas desarrollaremos en profundidad este proceso, así como los pasos tomados para completar el desarrollo.

II. Dispositivo de Percepción

Para la capa de percepción, se hizo uso de la placa de desarrollo IoT portátil M5StickC PLUS, basado en el chip ESP32, un microcontrolador de bajo costo y bajo consumo de energía muy utilizado para proyectos y prototipos de IoT que dispone, entre otras capacidades, de red Wifi de 2,4 GHz que se utiliza para la conexión con el servidor MQTT.

El M5StickC PLUS se programa utilizando Arduino, un framework de desarrollo que utiliza un lenguaje de programación basado en C++, aunque en nuestro caso, utilizamos C++ directamente, a través de la plataforma de desarrollo PlatformIO como extensión de Visual Studio Code, que integra control de versiones, debuggeado, y facilita el uso de bibliotecas, ya que proporciona una base de datos centralizada para su búsqueda, instalación y actualización.

Para comprender el proyecto, especialmente la capa de percepción y la aplicación del M5StickC PLUS, es importante comprender el funcionamiento del modo promiscuo.

El modo promiscuo es una capacidad de una interfaz de red que le permite capturar y analizar todo el tráfico de datos que se transmite en la red, incluso si los datos no están destinados específicamente a esa interfaz en particular.
En nuestro contexto el M5StickC PLUS se activa en modo promiscuo, detectando todos los dispositivos que estén activamente comunicándose a través de WiFi y dentro de su rango de detección.

Al establecer conexión en modo promiscuo, el dispositivo no tiene la capacidad de enviar señales de wifi, únicamente puede recibirlas. Es importante tener en cuenta esto a la hora de programar la aplicación. También tuvimos que tener en cuenta que, en general, cualquier dispositivo se conecta por defecto a un canal WiFi y se mantiene conectado a este canal mientras la conexión no se resetee. Este también es el caso para el M5StickC, por lo que se debe programar de forma que escuche en modo promiscuo todos los 13 canales, uno de cada vez.

Ya que la capa de percepción se centra sobre todo en la *adquisición* de señales y no en su procesamiento, cabe profundizar en la señal con la que trabajaremos en este proyecto. Se trata de paquetes WiFi, o como se conoce por su nombre técnico: IEEE 802.11.
Los paquetes que siguen el estándar IEEE 802.11 tienen numerosas regulaciones y peculiaridades, pero para nuestro caso nos interesan especialmente sus canales y frecuencias, y su contenido, comúnmente conocido como *frames*.

Sobre los canales, debemos tener en cuenta que estos paquetes se envían en el espectro de 2.400–2.500 GHz o en el de 4.915–5.825 GHz (comúnmente conocidos como 2.4G y 5G). El chip ESP23 del M5StickPlus recibe únicamente en la banda de 2.4G, por lo que esta será la que usaremos. A su vez, esta banda está subdividida en 14 canales, de los cuales, legalmente sólo podemos utilizar los 13 primeros.
El canal 14, conocido como la banda ISM, por sus siglas de *Industrial, Scientific and Medical*, se reserva al uso de la industria, medicina, investigación científica, ejército y diversas comunicaciones vía satélite. Por ello, usarla puede generar problemas en el tráfico aéreo, radares, comunicaciones satelitales y equipos de vigilancia, y hay acuerdo internacional de no usarla comercialmente (excepto en Japón, donde es legal a intensidades bajas, pero a no ser que pretendamos flashear un firmware Japonés, vamos listos).

Por otro lado, los *frames* son los bloques en los que se divide la información a enviar, y están generalmente compuestos de tres elementos:

1. una **cabecera** (*header*): que contiene generalmente la información necesaria para trasladar el paquete desde el emisor hasta el receptor,

2. el **área de datos** (*payload*): que contiene los datos que se desean trasladar,

3. y la **cola** (*trailer*): que comúnmente incluye código de detección de errores.

La estructura exacta, a nivel de bits, de los *frames* que siguen el protocolo IEEE 802.11 es la siguiente.

Field	Frame control	Duration, id.	Address 1	Address 2	Address 3	Sequence control	Address 4	QoS control	HT control	Frame body	Frame check sequence
Length (Bytes)	2	2	6	6	6	0, or 2	6	0, or 4	0, or 4	*Variable*	4

Cada una de las secciones cumple una función propia, pero nos interesa el header, que contiene los *adress*. Estos *adress* son las MAC's (*Media Acess Control)*, que son identificadores únicos que el dispositivo que envía o recibe señales WiFi incluye en los paquetes. Los tres primeros *adresses* son los siguientes:

1. Dirección MAC de destino (*adress 1*): Esta es la primera dirección MAC que aparece en el paquete. Indica la MAC del dispositivo receptor al que se dirige el paquete.
2. Dirección MAC de origen (*adress 2*): Indica la MAC del dispositivo que envía el paquete.
3. Dirección MAC BSSID "*Basic Service Set Identifier*"(*adress 3*): En el caso de los paquetes asociados a un punto de acceso Wi-Fi, la dirección MAC BSSID es la siguiente en aparecer. Identifica de manera única el punto de acceso al que están asociados el emisor y el receptor.

De entre todas estas MAC's la que nos interesa es la segunda: conociendo el dispositivo de origen es como podremos determinar cuántos dispositivos se encuentran en nuestro rango de detección.

Ahora que ya conocemos como se estructura la señal, sus limitaciones, y las partes que nos interesan, podemos pasar a la aplicación de procesado de datos. Más adelante, en la explicación del código, veremos como obtener el canal y la RSSI.

III. Preprocesado de datos

Siguiendo los detalles mencionados en el punto anterior, y teniendo en cuenta la plantilla de código con la que contábamos, daremos una explicación de cómo funciona el código y del proceso que seguimos para ampliar la plantilla y añadir funcionalidades.

A continuación se muestra una lista con las funciones creadas en el código y una descripción breve de cada una. A partir de estas descripciones, y con un vistazo al código, explicaremos su funcionamiento.

```
esp_err_t esp_wifi_init(wifi_init_config_t *config); // Inicializa el controlador WiFi del ESP32.
esp_err_t esp_wifi_set_country(const wifi_country_t *country); // Establece el país para configurar el WiFi.
esp_err_t esp_wifi_set_storage(wifi_storage_t storage_type); // Configura el almacenamiento de WiFi en RAM.
esp_err_t esp_wifi_set_mode(wifi_mode_t mode); // Establece el modo WiFi en modo nulo.
esp_err_t esp_wifi_start(void); // Inicia el WiFi.
esp_err_t esp_wifi_set_promiscuous(bool enable); // Habilita el modo promiscuo del WiFi para capturar paquetes.
esp_err_t esp_wifi_set_promiscuous_rx_cb(wifi_promiscuous_cb_t cb); // Establece el controlador de paquetes WiFi promiscuos.
esp_err_t tcpip_adapter_init(void); // Inicializa el adaptador TCP/IP.
static void wifi_sniffer_init(void); // Inicializa el sniffer de WiFi.
esp_err_t wifi_sniffer_set_channel(int channel); // Establece el canal de WiFi para la captura de paquetes.
const char* wifi_sniffer_packet_type2str(wifi_promiscuous_pkt_type_t type); // Convierte el tipo de paquete WiFi a una cadena de texto.
void handleMqttMessage(char* topic, byte* payload, unsigned int length); // Controlador de mensajes MQTT recibidos.
esp_err_t event_handler(void* ctx, system_event_t* event); // Controlador de eventos del sistema.
void start_Scan_Wifi(void); // Inicia el escaneo de redes WiFi.
void end_Scan_Wifi(void); // Finaliza el escaneo de redes WiFi.
void setup_wifi(void); // Configura la conexión WiFi.
void MQTT_Server(void); // Configura y establece la conexión MQTT.
void reboot(void); // Reinicia el dispositivo ESP32.
void checkButton(void); // Verifica si se ha presionado el botón de reinicio.
void toggleLcd(void); // Activa o desactiva la retroiluminación de la pantalla LCD.
void setup(void); // Configuración inicial del dispositivo.
void loop(void); // Bucle de ejecución, funciona constantemente
```

Buena parte de estas funciones estaban ya creadas en la plantilla inicial que se nos proporcionó.
Para entender el funcionamiento del programa, podemos observar el flujo de ejecución que sigue. Para esto, analizamos primero la función `setup()` y la función `loop()`.

```
void setup()
{
  M5.begin();                             // Initialize the M5StickC Plus
  esp_task_wdt_init(WDT_TIMEOUT, true); // Initialize the Watchdog Timer
```

```
  M5.Axp.ScreenBreath(9);                 // Set to 0 to turn off the LCD
  M5.Lcd.setRotation(1);                  // Set the LCD rotation
  M5.Lcd.fillScreen(BLACK);               // Fill the screen with black color
  M5.Lcd.setTextColor(RED);              // Set the text color to red
  M5.Lcd.setTextSize(1);                  // Set the text size

  pinMode(BUTTON_PIN, INPUT_PULLUP); // Initialize reboot button

  wifi_sniffer_init(); // Initialize the WiFi sniffer

  M5.Lcd.setCursor(0, 128);
  M5.Lcd.printf("Current Channel: %d", channel); // Display current channel

  //We create the remote_control topic
  sprintf(mqttRebootTopic, "aps2023/Proyecto7/remote_control/%llu", chipId);
}
```

Esta función es relativamente autoexplicativa. Inicializamos el microcontrolador junto con los elementos del LCD, inicializamos el watchdog y guardamos al final una string con el tópico que usaremos para el MQTT.
Cabe pararse a explicar la función `wifi_sniffer_init()`.

```
// Function to initialize WiFi sniffer
void wifi_sniffer_init(void)
{
  tcpip_adapter_init();                                  // Initialize the TCP/IP adapter
  ESP_ERROR_CHECK(esp_event_loop_init(event_handler, NULL)); // Initialize the event loop
  wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
  ESP_ERROR_CHECK(esp_wifi_init(&cfg));                  // Initialize the WiFi driver
  ESP_ERROR_CHECK(esp_wifi_set_country(&wifi_country));  // Set the WiFi country to Spain
  ESP_ERROR_CHECK(esp_wifi_set_storage(WIFI_STORAGE_RAM));   // Set WiFi storage to RAM
  ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_NULL));       // Set WiFi mode to NULL mode
  ESP_ERROR_CHECK(esp_wifi_start());                     // Start WiFi
  esp_wifi_set_promiscuous(true);                        // Enable promiscuous mode
  esp_wifi_set_promiscuous_rx_cb(wifi_sniffer_packet_handler); // Set the callback for
WiFi packet handling
}
```

La mayoría de estas funciones provienen de la biblioteca `"esp_wifi.h"`, y configuran el sensor WiFi del ESP32 para su correcto funcionamiento. Nos interesan
`tcpip_adapter_init()`
`esp_wifi_set_promiscuous(true)`
`esp_wifi_set_promiscuous_rx_cb(wifi_sniffer_packet_handler)`
ya que o las veremos más adelante en el código o tienen cierta importancia. La segunda hace lo que claramente indica su nombre.
La primera función configura el protocolo TCP/IP, un conjunto de protocolos que permite la comunicación entre dispositivos en una red, utilizando el protocolo IP para el direccionamiento y la transferencia de datos.
Es necesaria para la comunicación en redes ya que comunica el hardware de red y el software de aplicación; define parámetros como la dirección IP del dispositivo, la máscara de subred, la puerta de enlace predeterminada, los servidores DNS, entre otros. Estos

parámetros son esenciales para que el dispositivo pueda comunicarse con otros dispositivos en la red y acceder a recursos de red como servidores MQTT, servicios en la nube, bases de datos, etc.

La tercera función registra otra función de devolución de llamada (*callback*) que se invocará cada vez que se reciba un paquete en el modo promiscuo (este funcionamiento recuerda al de un evento). La función de devolución de llamada, `wifi_sniffer_packet_handler`, será llamada con cada paquete recibido, permitiéndonos procesar y analizar los paquetes capturados según sea necesario. Más adelante veremos como modificamos la función `wifi_sniffer_packet_handler` original para gestionar los paquetes de acuerdo a nuestras necesidades.

Una vez realizado el *setup*, pasemos a explicar la función `loop()`.

```
void loop()
{
  esp_task_wdt_reset(); // Reset the Watchdog Timer
  checkButton();
  M5.update();

  if (M5.BtnB.wasPressed())
  {
      toggleLcd(); // Call the toggleLcd() function when the secondary button
is pressed
  }

  vTaskDelay(WIFI_CHANNEL_SWITCH_INTERVAL / portTICK_PERIOD_MS);
  channel++;

  if (channel > WIFI_CHANNEL_MAX)
  {
      // Serialize the JSON array to a string
      end_Scan_Wifi(); // Disable promiscuous mode;

      if (setup_wifi())
      {
      MQTT_Server();
      }

      M5.Lcd.fillScreen(BLACK);
      start_Scan_Wifi();

      channel = 1;
  }
  wifi_sniffer_set_channel(channel);
}
```

Esta función se ejecuta hasta el fin de los tiempos o hasta que el *watchdog* haga timeout, así se asegura que ante un mal funcionamiento, el programa se resetee en lugar de dejar de funcionar por completo.
En las primeras líneas hacemos comprobaciones de los botones del M5, y con la función `vTaskDelay()` aseguramos que el programa se mantenga cierto tiempo en cada canal para recibir todos los paquetes posibles.

Después de esta función, colocamos un condicional que se ejecuta cuando se ha pasado por todos los canales. Al ejecutarse, el M5 deja de escuchar en modo promiscuo ya que como habíamos establecido previamente, al establecer conexión en modo promiscuo, el dispositivo no tiene la capacidad de enviar señales de wifi, únicamente puede recibirlas.
Un nuevo condicional dentro del anterior ejecuta `setupWiFi()`, una función booleana que intenta conectarse a la red configurada en el archivo `"config.h"`.

Si lo consigue, ejecuta la función de conexión al servidor MQTT, que publica en el tópico correspondiente los datos obtenidos con `wifi_sniffer_packet_handler`.
Por tanto, lo único que nos queda por explicar es esta función, ya que es la que realiza todo el procesado de los paquetes. Esta función es extensa y es la que más modificamos con respecto a la plantilla original.

```
// Callback function for WiFi packet handling
void wifi_sniffer_packet_handler(void *buff, wifi_promiscuous_pkt_type_t type)
{

  const wifi_promiscuous_pkt_t *ppkt = (wifi_promiscuous_pkt_t *)buff;
// Get the WiFi packet
  const wifi_ieee80211_packet_t *ipkt = (wifi_ieee80211_packet_t *)ppkt->payload;
// Get the IEEE80211 packet

  const wifi_ieee80211_mac_hdr_t *hdr = &ipkt->hdr;

  char mac1[18];
  sprintf(mac1, "%02X:%02X:%02X:%02X:%02X:%02X", hdr->addr1[0], hdr->addr1[1],
hdr->addr1[2], hdr->addr1[3], hdr->addr1[4], hdr->addr1[5]);

  char mac2[18];
  sprintf(mac2, "%02X:%02X:%02X:%02X:%02X:%02X", hdr->addr2[0], hdr->addr2[1],
hdr->addr2[2], hdr->addr2[3], hdr->addr2[4], hdr->addr2[5]);
...
```

En primer lugar, declaramos `ppkt`, un puntero a una estructura `wifi_promiscuous_pkt_t` que se utiliza para acceder a los datos del paquete capturado, y `ipkt`, un puntero a una estructura `wifi_ieee80211_packet_t` que se utiliza para acceder a los datos del paquete IEEE80211 contenido dentro del paquete WiFi.
El primero es el paquete que procesa el ESP32, y por tanto el segundo es un subconjunto del primero. `ppkt` contiene, un header con datos como el canal y la RSSI, y un payload con el paquete IEEE80211; `ipkt` es este payload.
En `*hdr` extraemos el header del paquete ya que como vimos en la sección III, es lo que realmente nos interesa. Guardamos después, dos cadenas con las MAC's 1 y 2 del header. La primera, aunque se imprimirá por pantalla más adelante, no nos interesa tanto como la segunda.

A continuación realizamos un filtrado de los paquetes con una dirección MAC de destino que coincide con una dirección MAC en `macAddresses` usando la función `strcmp`.

```
  ...

/**/ // COMMENT THIS SECTION FOR NO FILTERING. NOT RECOMMENDED!!!!
DEVICE RUNNING OUT OF MEMORY
  for (int i = 0; i < numAddresses; i++)
  {
    // if (strcmp(mac1, macAddresses[i]) == 0 || strcmp(mac2,
macAddresses[i]) == 0) // STRICT FILTER. MOST PACKETS ARE LOST AND
ONLY SOME USERS ARE SHOWN. MOST READABLE. FOR TESTING PURPOSES ONLY
    if (strcmp(mac2, macAddresses[i]) == 0) // SOFT FILTER. MOST
PACKETS ARE LOST, BUT ALL CLIENTS ARE SHOWN. RECOMMENDED MODE.
    {
      return; // Skip duplicate MAC addresses
    }
  }
...
```

Estos paquetes se omiten y no se procesan más. En los comentarios se muestra otro filtrado más estricto que implementamos, pero decidimos rechazar en favor del actual.

Finalmente, almacenamos todos los datos previamente mencionados en un objeto JSON que será el que se publique en el servidor MQTT y dará paso a la siguiente capa del sistema:

```
  ...
  JsonObject macObject = jsonDocument.createNestedObject();
  macObject["mac"] = mac2;
  macObject["channel"] = ppkt->rx_ctrl.channel;
  macObject["rssi"] = ppkt->rx_ctrl.rssi;
  ...
```

Un par de aclaraciones finales:

- En el archivo `"config.h"` se encuentran todas las configuraciones de WiFi y MQTT necesarias para las funciones de conexión.

```
const char apWifiName[] = "";
const char wifiInitialApPassword[] = "";
//-- VARIABLES DE LOS PARÁMETROS A CONFIGURAR
char mqttServerValue[STRING_LEN] = "aps2023.is-a-guru.com";
const int mqttServerPortValue = 2883;
char mqttUserNameValue[STRING_LEN] = "";
char mqttUserPasswordValue[STRING_LEN] = "";
char mqttTopicValue[STRING_LEN] = "aps2023/Proyecto7/";
char mqttAutoSend[STRING_LEN];
```

- Debido a la longitud del objeto JSON, tuvimos que hacer una serie de cambios en el paquete `<PubSubClient.h>`

```
// Cambios hechos en PubSubClient.h
#define MQTT_MAX_PACKET_SIZE 256  // Original
#define MQTT_MAX_PACKET_SIZE 8192 // Nuevo
size_t buildHeader(uint8_t header, uint8_t* buf, uint16_t length); // Original
size_t buildHeader(uint8_t header, uint8_t* buf, uint64_t length); // Nuevo
// Cambios hechos en PubSubClient.cpp
size_t PubSubClient::buildHeader(uint8_t header, uint8_t* buf, uint16_t length) // Original
size_t PubSubClient::buildHeader(uint8_t header, uint8_t* buf, uint64_t length) // Nuevo
uint16_t len = length; // Original
uint64_t len = length; // Nuevo
```

IV. Procesamiento y Gestión de Datos

Una vez el M5StickC ha procesado los datos, y ha estructurado y enviado con éxito el JSON con información sobre las MAC's al servidor MQTT, los datos se envían a la capa superior para su análisis.

En esta capa utilizamos una combinación de distintos softwares: HomeAssistant, NodeRed y phpMyAdmin. El pipeline que siguen los datos es el siguiente: NodeRed, integrado en HomeAssistant, se mantiene suscrito a la escucha de nuevos mensajes en el tópico correspondiente del bróker MQTT.

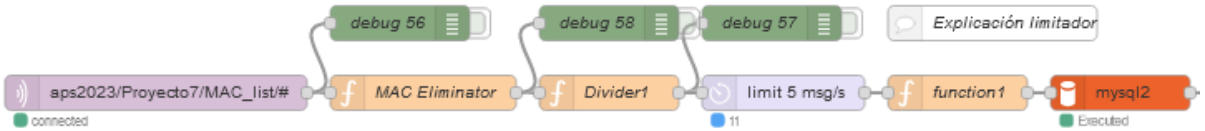
Cuando estos datos llegan al servidor, pasan por el flujo de NodeRed, donde se procesa de nuevo la información para llegar a la base de datos.

A continuación explicamos el flujo de NodeRed que utilizamos para procesar los datos.

1. Lo primero que necesitamos es un nodo que se encargue de suscribirse al tópico MQTT correspondiente. Cada vez que detecte un mensaje en el tópico, lo enviará a través del flujo.

2. Es importante saber que los nodos de Node-Red actúan como funciones que envían y reciben un objeto que contiene información, generalmente el objeto `msg`; un mapa que contiene pares clave-valor a los que se accede a través de llamadas como `msg.payload` . Estos objetos se pueden manipular y gestionar en funciones de JavaScript.

   Por tanto, en el segundo paso, utilizamos una función para aplicar un procesamiento extra al JSON que obtenemos en MQTT., que realiza la eliminación de direcciones MAC duplicadas en un conjunto de datos y devuelve un array que contiene las direcciones MAC únicas junto con sus valores correspondientes de RSSI y canal WiFi.



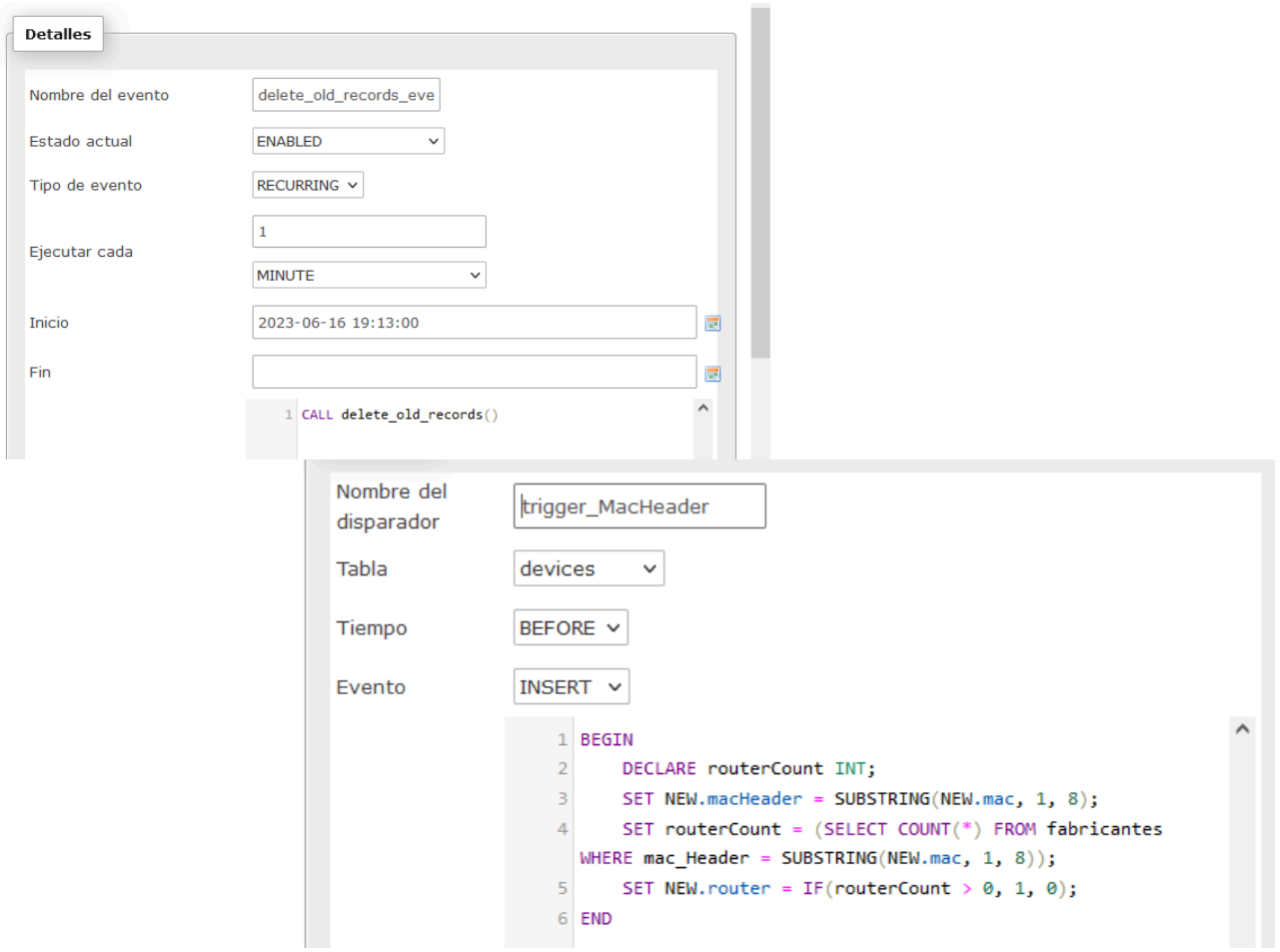
3. Para proporcionar facilidad a la hora de procesar esta información, lo que haremos será enviarla a un nodo SQL que la introducirá en una base de datos de phpMyAdmin. Utilizamos una función para acondicionar el contenido de `msg`, y colocamos un timer que limite el número de entradas por segundo a 5 para evitar la sobrecarga del servidor. Así, los datos `mac`, `channel`, `potencia`, `id_m5`, se insertan en una tabla con columnas para:
   - MAC detectada
   - RSSI / Potencia detectada
   - Canal de detección
   - El ID del dispositivo que las detecta.
   - Fecha y hora de detección
   - Header de la MAC, que será útil para procesamiento posterior
   - Parámetro Router, que será útil para procesamiento posterior

4. La siguiente y última fase del procesamiento se realiza en la propia base de datos de phpMyAdmin. Establecemos el siguiente procedimiento…

```
BEGIN
  DELETE FROM `devices` WHERE `fecha_hora` < (NOW() - INTERVAL 5
MINUTE);
END
```

…y un evento que lo ejecuta cada minuto. Así eliminaremos las filas que llevan más de 5 minutos inactivas en la tabla.

Finalmente, utilizamos un trigger en el server de phpMyAdmin, que compara la columna header de tabla devices con la tabla de fabricantes, y usa esto para determinar que dispositivos no están asociados a una persona. Esto lo hace cambiando el valor de la columna router.



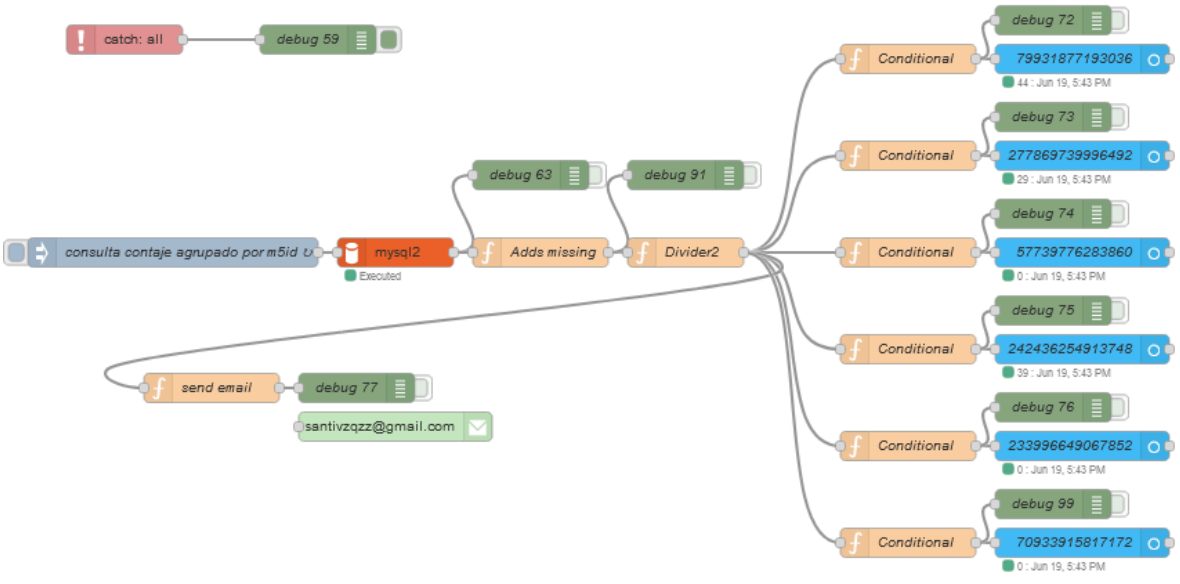
V. Capa de Aplicación y Servicios

Una vez el M5StickC ha procesado los datos, y ha estructurado y enviado con éxito el JSON con información sobre las MAC's al servidor MQTT, los datos se envían a la capa superior para su análisis y toma de decisiones.

En esta capa utilizamos una combinación de distintos softwares: HomeAssistant, NodeRed, phpMyAdmin y InfluxDB. El pipeline que siguen los datos es el siguiente: NodeRed, integrado en HomeAssistant, se mantiene suscrito a la escucha de nuevos mensajes en el tópico correspondiente del bróker MQTT.

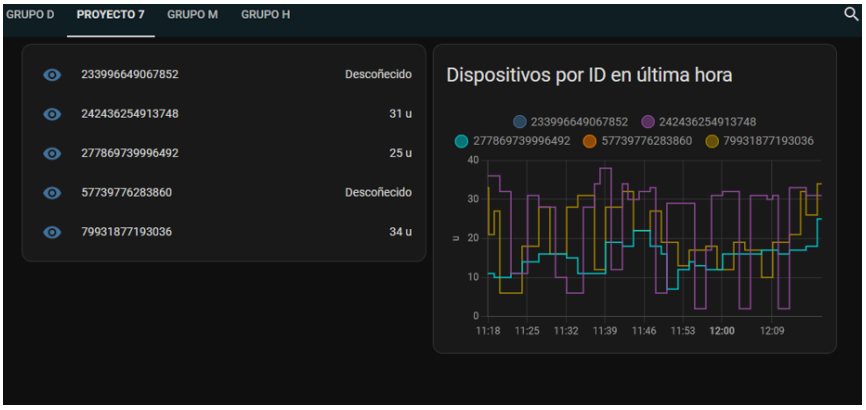
Cuando estos datos llegan al servidor, pasan por el flujo de NodeRed, donde se procesa de nuevo la información para dar la salida que verá el usuario. Estas salidas pueden tomar varias formas: se pueden hacer a través de peticiones a las API's de Telegram o de correo electrónico, para que el usuario reciba un mensaje, o pueden tomar forma de una gráfica o un display en el dashboard principal de HomeAssistant.

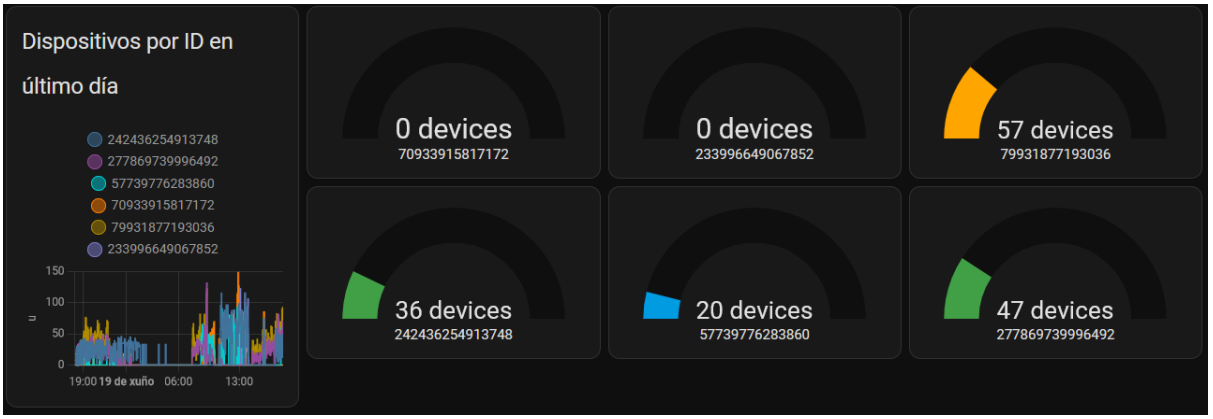
5. A continuación, mandamos los datos a un nodo que se ocupa de realizar una consulta `SELECT id_m5, COUNT(*) FROM devices WHERE router = 0 GROUP BY id_m5;` cada minuto. La función `Adds missing` se ocupa de comparar los identificadores de dispositivos presentes en el array `ids` con los identificadores existentes en el array `msgids`.
   Si un identificador de dispositivo no está presente en `msgids`, se crea un nuevo objeto con ese identificador y un contador inicializado en 0, y se agrega al array `msg.payload`. Así nos aseguramos de que todos los identificadores de dispositivos en `ids` estén representados en `msg.payload` con un contador inicial de 0, en caso de que no estén presentes previamente (ya que de otro modo la gráfica mantenía el último número de dispositivos detectado).
   Un nodo se encarga de enviarlos por correo, y otro usa un bucle junto a un `node.send(msg)` que envía los datos y enlaza cada dato a un nodo sensor con la entidad del dispositivo correspondiente.
   Después de todas las capas y el filtrado previo, esto nos dará el resultado final que nos interesa: el número de personas (o más bien dispositivos) en el área cubierta por cada M5StickC.



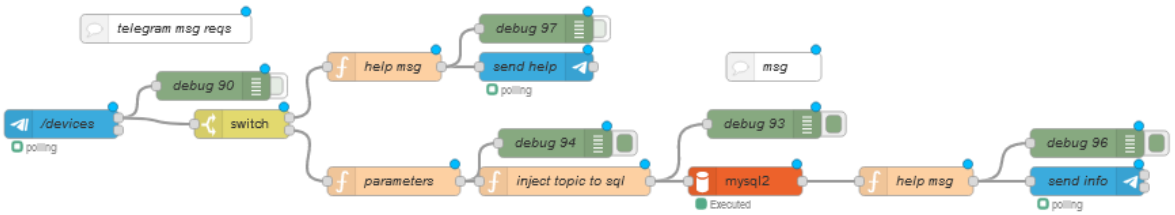
6. Finalmente, tenemos la salida de usuario. Nos interesa visualizar la información de forma intuitiva en el dashboard principal de HomeAssistant, o enviarla a través de algún medio como correo o Telegram.

   Representar la salida es muy sencillo en HomeAssistant. Basta con crear una tarjeta del tipo que nos interese para la entidad correspondiente. Estos son algunos ejemplos:





Por otro lado, decidimos añadir una funcionalidad que permitiese acceder bajo demanda a la información de los dispositivos a través de un bot de Telegram. Usando el bot de *aps2023Bot*, creamos el comando `/devices`, que toma parámetros `help` y `id, rssi` o `distancia.` Esto nos permite, en esencia, mandar querys a través de la API de Telegram para obtener información en tiempo real adaptada a lo que nos interese.



- Como ejemplo, el comando `/devices 245342424 -90` mandaría la query:

```
SELECT id_m5, COUNT(*) FROM devices WHERE router = 0 AND id_m5 = 245342424
AND potencia > -90 GROUP BY id_m5;
```

Ya que la escala de medición de RSSI es logarítmica, implementamos una función que permitiese un *input* en metros, ya que una escala lineal es mucho más intuitiva y los datos que nos interesan usualmente se encuentran en metros, (p.ej: dispositivos en un radio de 5 metros, lo que corresponde aproximadamente a -79 dBm siguiendo la función utilizada.), pero nuestra tabla almacena información en dBm.

Por tanto, tomamos la siguiente función;

$$Distance = 10^{((Measured\ Power - RSSI)/(10 * N))}$$

…y despejamos el RSSI.

$$RSSI = Measured\ Power - 10 \cdot N \cdot log_{10} D$$

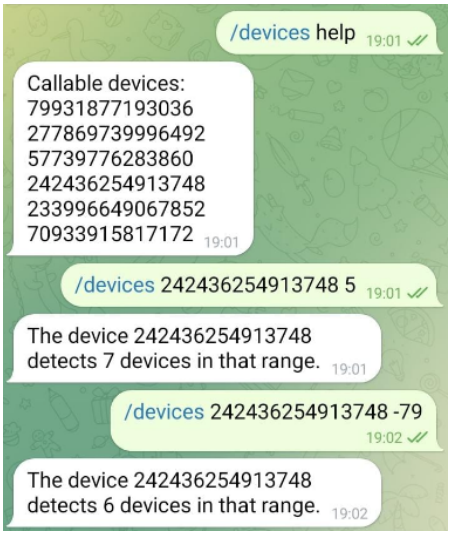
La función final tiene este aspecto:

```
// N and measuredPower are approximations
function calculateRSSI(measuredPower, distance, N) {
    var rssi = measuredPower - (Math.log10(distance) * 10 * N);
    return rssi;
}
const N = 2; // environmental constant
const measuredPower = -65; // reference rssi at 1 meter
```

Como se ve en los comentarios, cabe destacar que *N* y *measuredPower* son constantes aproximadas. *N* puede tomar valores de 2 a 4 y tiene en cuenta el estado del entorno por el que se propaga la señal (densidad / humedad del aire, obstáculos, etc.)

Measured Power es una media de sólo lectura que indica cuál es el RSSI esperado a una distancia de 1 metro del dispositivo. Según las búsquedas de información que realizamos, y siguiendo el consejo del profesor, decidimos utilizar -65 ya que la mayor parte de las fuentes apuntan a constantes cercanas a esta para los ESP32.

Como vemos aquí, la aproximación no es perfecta, especialmente en distancias mayores a 15 metros.



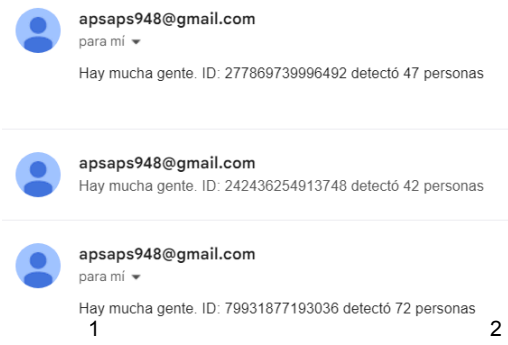
VI. Archivos de Entrada y Salida

Para las pruebas de entrada se usaron datos reales. El M5 detectaba las wifis y eso fue lo que se usaba en los archivos del MQTT y del Node-Red. También para ciertas pruebas del Node-Red se usaron *injects*. Así que los archivos de entrada que usamos fueron los resultados de otras partes del proyecto.

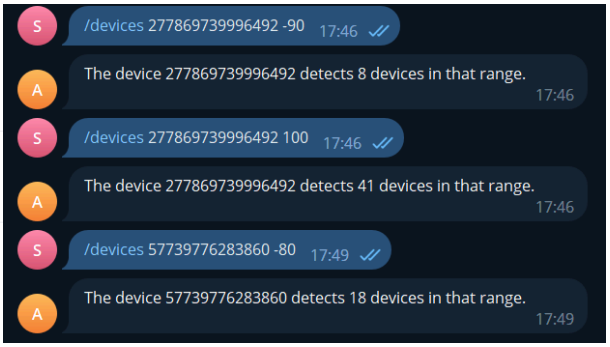
Las salidas eran lo que se imprimía por pantalla de los M5, los debugs del Node-Red, ver el interior de la base de datos, lo que se guarda en los sensores (que se ve en el resumen del Home Assistant en el Proyecto7), los correos electrónicos que se mandan cuándo se detecta más de 100 dispositivos y los *telegrams* con la información que se les pide.

VII. Resultados y Evidencia de Ejecución

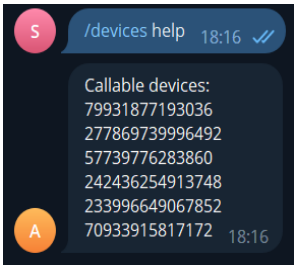
- Presentación de los resultados obtenidos en términos de número de dispositivos detectados.
- Inclusión de capturas de pantalla, registros de consola u otra evidencia visual relevante.



1



2



3

En cuanto a la visibilidad de los resultados finales hemos implementado dos formas de recibirlos. La primera es por correo, utilizando el nodo *email* del Node-RED que enviará un mensaje avisando de cuántas personas ha detectado un M5. Como vemos en la Imagen$_1$, el correo nos muestra el número de personas detectadas y el ID del M5 que lo ha hecho.

La segunda manera, es por telegram donde hemos creado un **bot** y definido el comando */devices* el cual recibe como parámetros el ID de un M5 y la distancia en metros o el rssi.

Además, tiene un parámetro de ayuda "*help*" el cual nos devuelve una lista con los ID de todos los M5 que hayan enviado datos. Como vemos en la Imagen$_2$, el bot nos responde con el número de dispositivos que se han detectado en el rango que le hemos pasado como segundo parámetro. Cabe destacar que si el valor es negativo le estamos indicando al bot que se trata de un rssi, por el contrario si el valor es positivo le estamos indicando la distancia en metros.

Por último, podemos recalcar lo que vimos en el apartado V, donde podemos ver unas gráficas de agujas y una historia que nos indican los dispositivos detectados por cada M5.

VIII. Reflexión y Conclusiones

Este fue un trabajo largo y duro, aunque se nos prestó ayuda y plantillas de código para su realización. No fue sencillo dado que nuestra experiencia con las herramientas empleadas era escasa. Nunca habíamos usado C++ (ni ningún otro lenguaje de bajo nivel), Javascript, Bases de datos, SQL ni Home Assistant.

En primer lugar tuvimos que enfrentarnos a la programación del M5. Fue complicado entender el código y saber porqué al cambiar una línea todo dejaba de funcionar. Tuvimos problemas como:

- Usar las librerías.
- Obtener el identificador de cada M5.
- Cambiar las librerías de MQTT para que mandase mensajes de larga longitud (mayores de 256 bits).
- Coger un gran número de MACs.
- Enviar datos al servidor MQTT.
- Un programa inicial que cogía menos MACs hasta que unimos fuerzas con el grupo D.

Para solucionarlo ayudó que teníamos un proyecto de GitHub en común. En segundo lugar hicimos todo lo que está situado en Home Assistant. No fue tan complicado como lo del M5, pero también fue trabajo.

La parte del Node-Red fue utilizar los ejemplos que estaban disponibles y usar ChatGPT para escribir Javascript. El Node-Red fue más fácil porque se podía dividir el trabajo en cada nodo. Fue complicado conectar al correo, mandar comandos por Telegram y crear los sensores para el Home Assistant, más por las configuraciones que requerían y el mal funcionamiento del servidor que por su programación.

La parte de SQL no dió demasiada dificultad. Hubo que hacer un evento y un trigger que ChatGPT supo programar. El evento era para eliminar los dispositivos antiguos y el trigger para ver cuales están en la tabla de fabricantes.

Con respecto a las posibles mejoras que se podrían hacer en el proyecto, se nos ocurren un par:

- Detectar MACs que están entre varios dispositivos y saber su ubicación relativa a los M5s, una especie de sistema de trilateración. Con esto se podría hacer un mapa tridimensional de la Universidad y como se mueven los dispositivos. Para esto ayudaría que los M5 se detectaran unos a los otros y poder calcular el error de la localización. También se podría usar la localización por Bluetooth para tener más precisión.

- Calibrar los M5 para que no calculen las distancias como si estuviesen en un espacio plano, es decir, que calcule bien si hay una pared en medio o no. Habría que mejorar la conversión de bBm a metros.

- Mejorar y depurar el código del M5 y la comunicación con el servidor. Con un código más optimizado y una comunicación más rápida con la base de datos, la latencia se reduciría y el contaje se acercaría más a tiempo real. Esto implicaría averiguar el tiempo óptimo durante el que un M5 debe escuchar en cada canal, mejorar el tiempo requerido para conectarse a la WiFi y MQTT, etc.

IX. Referencias

Esta es una lista de fuentes utilizadas para el desarrollo del proyecto y el informe:

- https://github.com/SMH17/M5StackWiFiSniffer
- https://chat.openai.com (Para dudas sobre el codigo en C++ y JavaScript, y alguna duda con todo lo relacionado al MQTT)
- https://stackoverflow.com
- https://github.com/m5stack/M5StickC-Plus/tree/master/examples/Advanced
- https://docs.m5stack.com/en/api/stickc/system_m5stickc
- https://iotandelectronics.wordpress.com/2016/10/07/how-to-calculate-distance-from-the-rssi-value-of-the-ble-beacon/
- https://m5stack.hackster.io/ryo-naka/distance-estimation-by-ble-with-ibeacon-and-m5mtickc-483b28
- https://www.adslzone.net/2019/02/15/wifi-canal-14-ilegal/

Este proyecto fue un esfuerzo colaborativo entre los grupos H, D y Q, y el profesor de la asignatura.

X. Anexos

En la entrega se incluyen los siguientes anexos:

- Carpeta del proyecto, en formato de proyecto de PlatformIO, con el código fuente, bibliotecas y archivo de configuración, listo para ser compilado directamente en el M5StickC PLUS. Archivo ZIP.
- Flujo de trabajo de Node-RED, en formato JSON.
- Copia de una instancia de la base de datos en SQL.