Práctica 6: Tecnologías de comunicación MQTT y sensorización BLE

Sistemas Embebidos

Francisco Javier Pérez Martínez Francisco Joaquín Murcia Gómez
Alex Navarro Soria Elvi Mihau Sabau Sabau

30 de abril de 2022

Índice

1. Abstracto **3**

2. Descripción **3**

3. Parte A. Comunicación MQTT **4**
3.1. Instalación y configuración del broker Mosquitto 4

4. Parte B. Trazabilidad BLE **5**
4.1. Configuración de los sensores 5
4.2. Configuración de la aplicación (VPS) 6
4.3. Primeras pruebas 6
4.4. Configuración del Consumidor (Arduino Cloud) 12

5. Conclusión **13**

1. Abstracto

En esta práctica usaremos varias tecnologías como MQTT, BLE, Docker y WiFiNINA para trazar la posición de un dispositivo BLE en vivo.

Keywords: MQTT, Arduino, Mosquitto, Producer, Consumer, BLE, Docker, WiFiNina.

2. Descripción

En esta práctica realizaremos un sistema capaz de determinar la posición de un dispositivo BLE.

Usaremos 3 arduinos como sensores capaces de detectar y emitir la intensidad de la señal BLE del dispositivo a trazar.

Además usaremos un VPS con Docker para servir un Broker MQTT y desplegaremos una aplicación aparte en Docker para recibir dichos datos, donde aplicaremos el algoritmo de trilateración y lo emitiremos vía MQTT.

Nuevamente otro arduino recibirá dichos datos vía MQTT y se encargará de mandar los datos en un proyecto ArduinoIoTCloud, donde plasmaremos dicha posición en un dashboard.

El esquema de la arquitectura sería el siguiente:

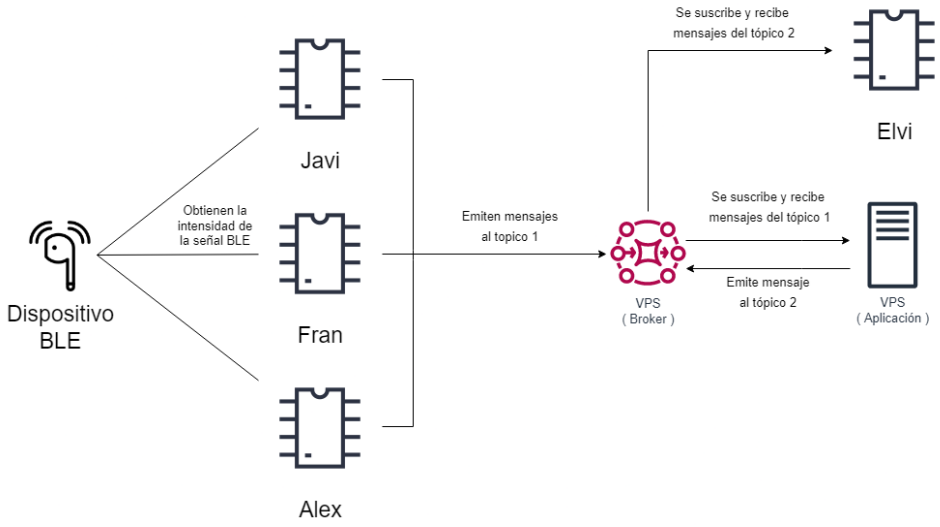

Figura 1: Esquema de la estructura del canal de comunicaciones.

3. Parte A. Comunicación MQTT

3.1. Instalación y configuración del broker Mosquitto

Esta práctica se debía haber usando una Raspberry, en nuestro caso al no tener una Raspberry hemos optado por usar el VPS de Elvi[a] con Docker.

Vía docker, hemos desplegado fácilmente un contenedor de Mosquitto, un broker MQTT.

La página oficial de Mosquitto provee un broker publico, que mucha gente utiliza para realizar pruebas por el hecho de que la conexión y los datos emitidos y recibidos son públicos, y por ello no es recomendable usar dicho broker para otros proyectos que maneje información privada.

En cambio, podemos desplegar nuestra propia instancia de mosquitto privada, y configurar dicha instalación para que use certificados o autenticación con contraseña a cada tópico.

En nuestro caso, hemos optado por desplegar nuestra propia instancia de Mosquitto usando Docker como se muestra en **este enlace**[b].

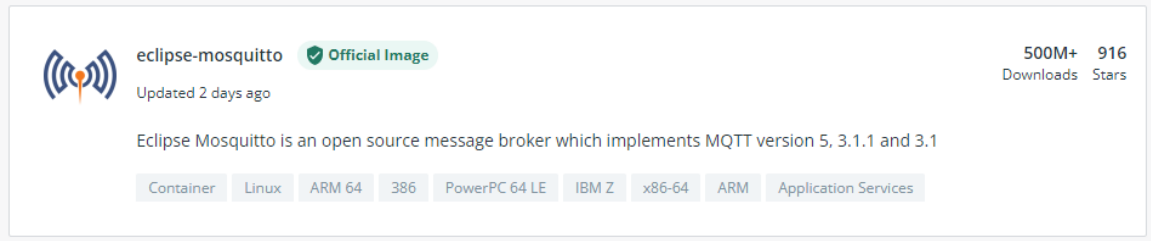

Figura 2: Imagen Docker de Mosquitto en Docker-Hub.

Al estár realizado con docker el despliegue es sencillo y automático, y con una sola linea de comando podemos tener nuestra propia instancia de mosquitto privada en ejecución.

```
docker run -d -p 1883:1883 eclipse-mosquitto
```

Este comando arrancará un contenedor en segundo plano de la imagen `eclipse-mosquitto`, y estará a la escucha de cualquier evento en el puerto `1883`, por defecto la conexión es TCP. Cualquier configuración del broker se puede cambiar desde el archivo `/mosquitto/config/mosquitto.conf` dentro del contenedor.

[a] https://traefik.oldbox.cloud
[b] https://hub.docker.com/$_{/}$*eclipse* − *mosquitto*

4. Parte B. Trazabilidad BLE

4.1. Configuración de los sensores

Para la configuración de los tres sensores se ha usado una modificación del código de escaneo BLE empleado en la practica 4 para poder así usar la antena con funcionalidad WIFI y con BLE. Básicamente se trata de utilizar la función BLE.end() después de escanear y WIFI.end() después de enviar al broker.

```
1  void loop() {
2      int rssi=0;
3    //ConectarBLE
4    if (!BLE.begin()) {
5      Serial.println("starting Bluetooth® Low Energy module failed!");
6      while (1);
7    }
8    Serial.println();
9    while(rssi==0){
10     BLE.scanForUuid("fd6f");
11
12     //. . .
13     // funciones de escaneo
14     //. . .
15
16   }
17   BLE.end();
18   ConectarWifi_Broker(); // nos conectamos a internet y al broker
19
20     //. . .
21     // envio de datos
22     //. . .
23
24   WiFi.end();
25 }
```

Para facilitar la recepción del cliente para que no reciba datos basura, solo se escanea el dispositivo con la función BLE.scanForUuid(), donde se introduce el UUID del dispositivo a rastrear en este caso un teléfono y solo enviamos cuando detectamos intensidad de señal ya que hay veces que escanea y no detecta el dispositivo.

4.2. Configuración de la aplicación (VPS)

Esta aplicación se encargará que recibirá los datos de los sensores vía MQTT a través de un tópico, calculará la posición y la mandará al 4to arduino vía MQTT por otro tópico.

Hemos decidido realizarla usando Node, con la librería `mqtt`.

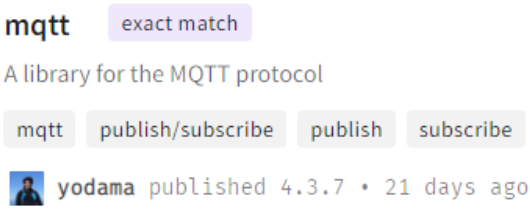

Figura 3: Imagen de la librería mqtt en npm.

4.3. Primeras pruebas

Lo que primero hicimos para familiarizarnos con la librería y probar que el broker desplegado anteriormente funciona fue crear un ejemplo sencillo, donde recibimos mensajes del tópico de la Práctica 5, y los emitimos a un nuevo tópico.

```
1  // Carga la libreria.
2  const mqtt = require('mqtt');
3
4  // Crea un cliente MQTT.
5  const client = mqtt.connect('mqtt://oldbox.cloud');
6
7  // Al conectar al broker, se suscribe a la topic 'SE/practicaUA2022/murcia'.
8  client.on('connect', () => {
9      client.subscribe('SE/practicaUA2022/murcia');
10 });
11
12 // Al recibir un mensaje, se procesa, en este caso se muestra por pantalla
13 //   y se manda a otro topico.
14 client.on('message', (topic, message) => {
15     console.log(message.toString());
16     client.publish('SE/practicaUA2022/otroTopico', message.toString());
17 });
```

Al probar el ejemplo, confirmamos que nuestro broker y nuestro consumidor / productor en Node funcionan correctamente.

Figura 4: Imagen de la ejecución del código anterior, teniendo el dispositivo de Alex funcionando con el código de la práctica anterior.

Basándonos en este ejemplo, hemos realizado un programa que:
1) Recibe 31 medidas de la intensidad de cada uno de los barridos BLE de los arduinos.

2) Saca sus medianas.

3) Obtenemos la distancia de cada arduino al dispositivo usando la tabla de distancias que realizamos en las prácticas anteriores mediante interpolación.

4) Mandará dichas medidas a una función donde a través del algoritmo de trilateración devolverá las coordenadas de la posición del dispositivo medido.

El tablero que usaremos nosotros será un 2mx2m, ya que las medidas que tenemos son hasta 2,5m debido a que usamos una señal de baja potencia para tener medidas más precisas.

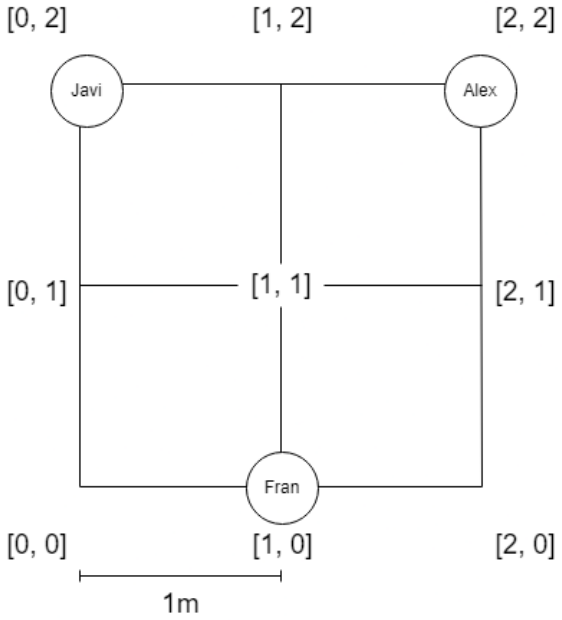

Figura 5: Diagrama del tablero de 2x2m.

Para la trilateración hemos usando una librería de JS que nos permite trilaterar. Nosotros solo tenemos que especificar las coordenadas en metros de cada arduino en el tablero.

```
const trilateration = require('trilateration');
...
trilateration.addBeacon(0, trilateration.vector(1, 0));
trilateration.addBeacon(1, trilateration.vector(0, 2));
trilateration.addBeacon(2, trilateration.vector(2, 2));
...
// function de trilateracion en dos dimensiones
function trilaterationProcess(a, b, c) {

    // Setting the beacons distances
    trilateration.setDistance(0, a);
    trilateration.setDistance(1, b);
    trilateration.setDistance(2, c);

    // Start Calculation
    return trilateration.calculatePosition();
}

```

```
...

// Obtiene la distancia basada en la intensidad, haciendo una interpolación.
function obtenDistancia(ints) {
    if (ints >= -43) return 0;
    let pos = intensity.findIndex((element) => ints >= element);
    // console.log(pos, distance[pos], distance[pos - 1], intensity[pos], intensity[pos - 1]);
    return distance[pos - 1] + (ints - intensity[pos - 1]) * ((distance[pos] - distance[pos - 1]) / (intensity[pos] - inten
}
...
let a = obtenDistancia(median(arduinos[0]));
let b = obtenDistancia(median(arduinos[1]));
let c = obtenDistancia(median(arduinos[2]));

// returns pos.x and pos.y based on measured disntances and beacon positions.
let pos = trilaterationProcess(a, b, c);
client.publish('SE/practicaUA2022/arduinocloud', pos.x.toString() + pos.y.toString());
...
```

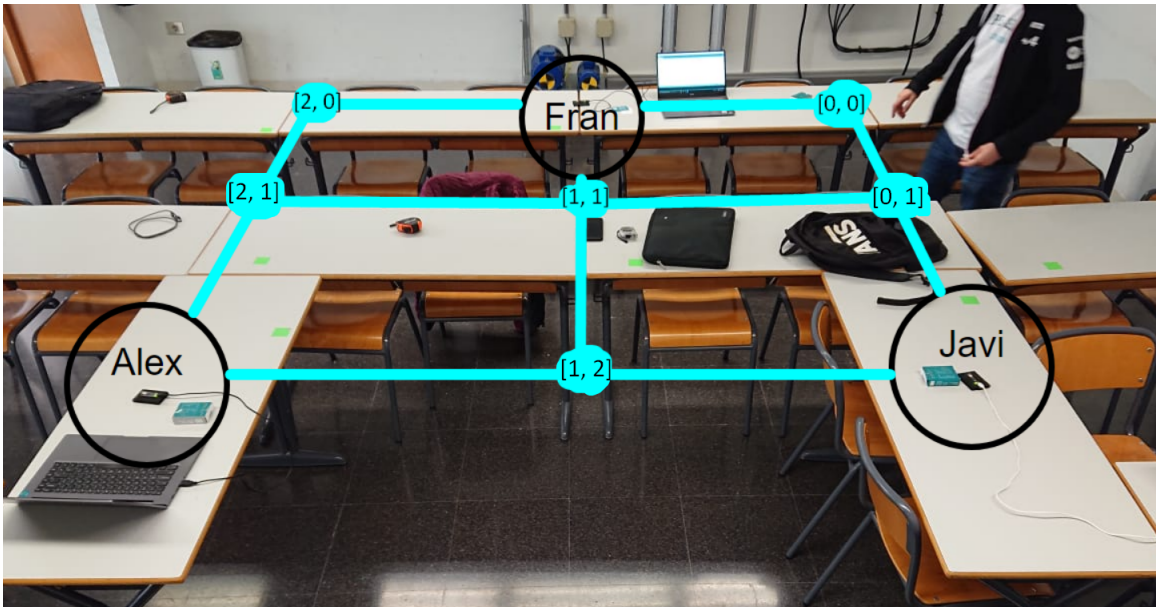

Figura 6: Foto del tablero del aula.

Una vez realizado este calculo, se mandan la posición al 4to arduino, publicando el dato al tópico. `arduinocloud`.

Al realizar las pruebas, obtenemos un error de +- 70cm por cada medida.

Figura 7: Foto de fran moviendo el dispositivo por el tablero.

```
Sincronizando barrido de arduinos, calculando posición...
Coords: 1.63, 1.24
Sincronizando barrido de arduinos, calculando posición...
Coords: 1.32, 1.44
Sincronizando barrido de arduinos, calculando posición...
Coords: 1.45, 1.34
Sincronizando barrido de arduinos, calculando posición...
Coords: 1.57, 1.54
```

Figura 8: Coordenadas obtenidas tras 4 barridos (4x31 medidas de cada arduino, obteniendo sus medianas).

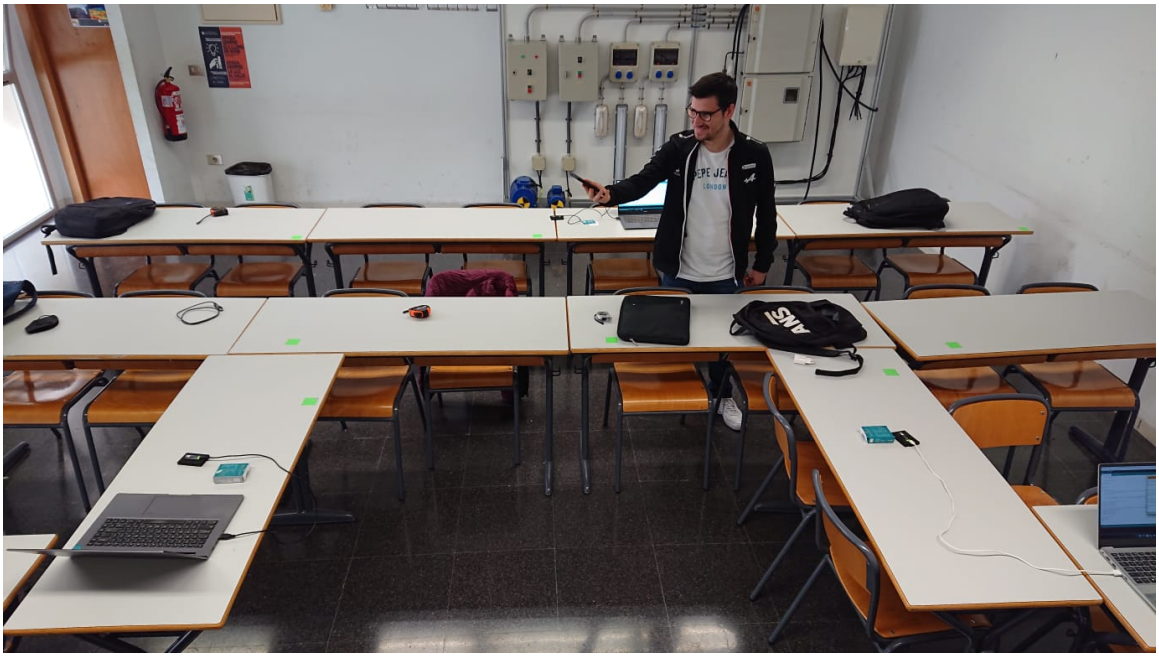

Figura 9: Foto de fran moviendo el dispositivo por el tablero.

```
Sincronizando barrido de arduinos, calculando posición...
Coords: 0.93, 0.24
Sincronizando barrido de arduinos, calculando posición...
Coords: 1.75, 0.52
Sincronizando barrido de arduinos, calculando posición...
Coords: 1.10, 0.32
Sincronizando barrido de arduinos, calculando posición...
Coords: 0.85, 1.02
```

Figura 10: Coordenadas obtenidas tras 4 barridos (4x31 medidas de cada arduino, obteniendo sus medianas).

Recordamos que cada medida de cada coordenada es la mediana de 31 medidas hechas con el dispositivo en la misma posición, pero debido a la inestabilidad de la intensidad bluetooth no hay mucha precisión para la poca zona que de abarca (2,5m x 2,5m máx en baja potencia).

Aún así, podemos observar cambios considerables en las coordenadas cuando el dispositivo cambia de zona.

4.4. Configuración del Consumidor (Arduino Cloud)

El arduino que se encargará de enviar los datos a ArduinoCloud, recibirá las coordenadas desde otro tópico `SE/practicaUA2022/arduinocloud`, donde se mostrarán los datos en el dashboard.

Este sketch tendrá 2 variables actualizables llamadas x e y.

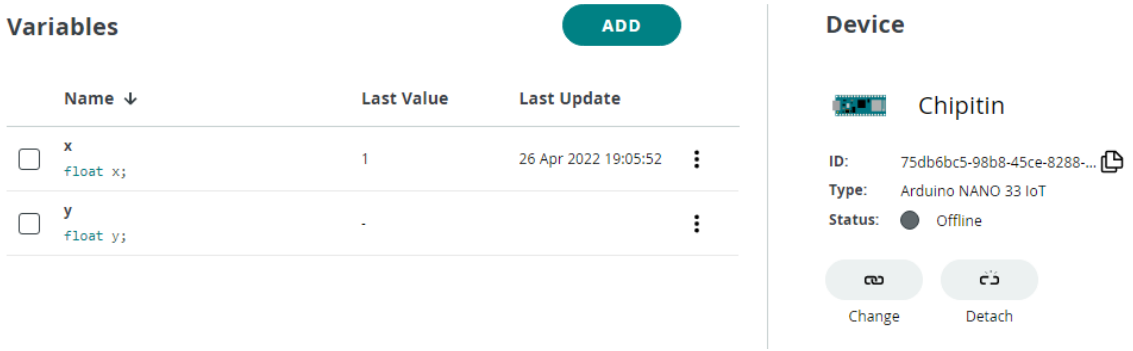

Figura 11: Captura del proyecto de arduinoCloud.

Además arduino tendrá el siguiente código para parsear las coordenadas recibidas via mqtt en el formato "X,Y".

```
...
void loop() {
  ArduinoCloud.update();

  if (mqttClient.parseMessage()) {
    bool next = FALSE;
    char character;
    String x_string;
    String y_string;

    // Obtain data per character.
    while (mqttClient.available()) {
      character = (char)mqttClient.read();

      // Message format is x,y
      if (character != ',' && !next) {
        x_string += (char)mqttClient.read();
      } else if (character == ',') {
        x = (float)x_string;
        next = TRUE;
      } else if ( character != ',' && next) {
        y_string += (char)mqttClient.read();
```

```
23          }
24        }
25        y = (float)y_string;
26      }
27  }
28  ...
```

Con todo esto, podremos plasmar las coordenadas recibidas en un dashboard.

Figura 12: Captura del proyecto de arduinoCloud.

5. Conclusión

De esta practica podemos concluir que un buen sistema de arduinos, coordinado y bien gestionado puede llegar a ser una solución sencilla y barata a un problema complejo. Como en este caso, la trazabilidad de un usuario en un área. También concluimos que la trazabilidad mediante BLE es bastante incorrecta, debido a la alta fluctuación del campo magnético en el área además de la presencia de otros dispositivos en el área.