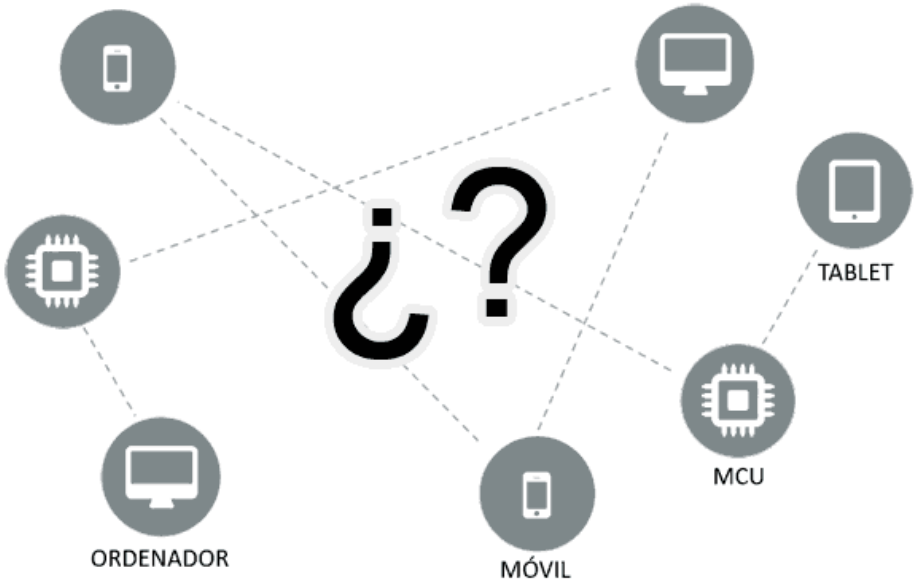
Computación Ubicua

Arduino y MQTT

Protocolos de Comunicación

- Un protocolo de comunicación es una serie de normas que definimos para que dos o más dispositivos puedan comunicarse y entenderse.

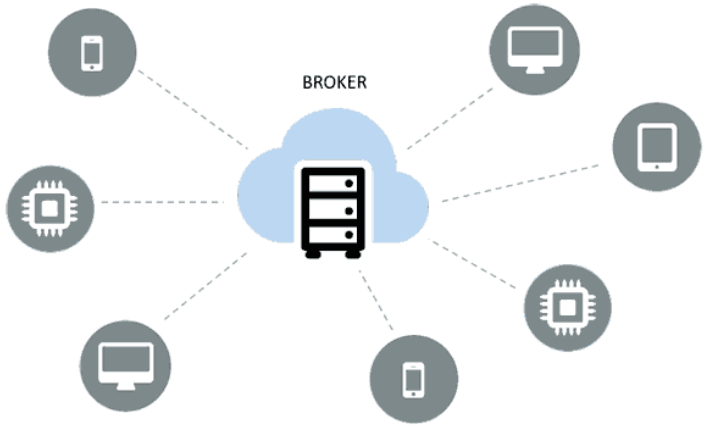


Condicionantes en IoT

- Posibilidad de tener una gran cantidad de dispositivos interconectados
- Escalabilidad del sistema: Que puedan añadirse o retirarse dinámicamente dispositivos sin que el comportamiento global se modifique
- Débil acoplamiento entre dispositivos: la dependencia entre los dispositivos debe ser la menor posible, y deseablemente nula
- Gran número de comunicaciones simultáneas: Respuestas rápidas gracias a mensajes pequeños y que no requieran gran procesamiento
- Seguridad: Dispositivos expuestos a Internet y que transmiten información privada e incluso controlan sistemas físicos

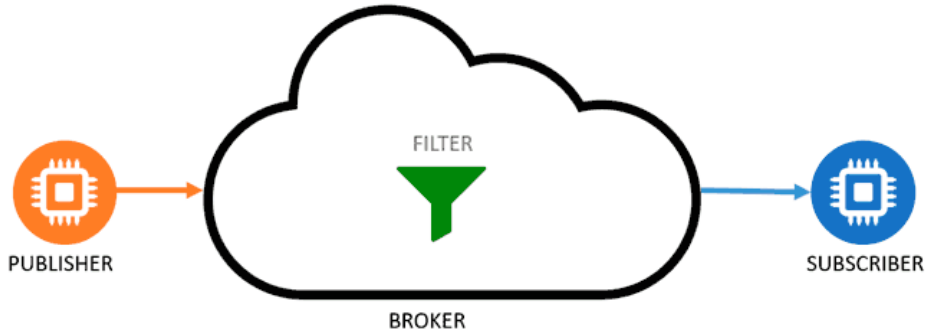
Soluciones de comunicación en IoT

- Externalizar la comunicación un servicio de notificaciones centralizado
- La solución consiste en disponer **un servidor central que se encarga de recibir los mensajes de todos los dispositivos emisores, y distribuirlos a los receptores**. De forma genérica llamaremos a este servidor ‘Router’ o ‘Broker’.



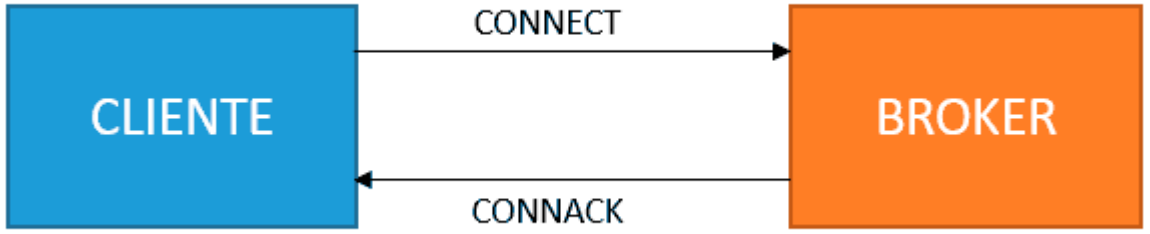
MQTT

- MQTT (Message Queing Telemetry Transport). Es un protocolo de comunicación M2M (machine-to-machine) de tipo message queue (Cola de mensajes).
- El funcionamiento del MQTT es un servicio de mensajería push con patrón publicador/suscriptor (pub-sub).
- Para filtrar los mensajes que son enviados a cada cliente los mensajes se disponen en topics organizados jerárquicamente.



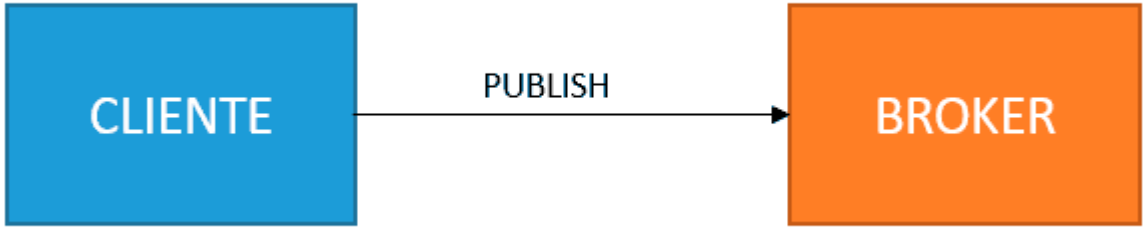
MQTT

- Los clientes inician una conexión TCP/IP con el bróker, el cual mantiene un registro de los clientes conectados. Esta conexión se mantiene abierta hasta que el cliente la finaliza.
- Para conectar el cliente envía un mensaje CONNECT que contiene información necesaria (nombre de usuario, contraseña, client-id…). El broker responde con un mensaje CONNACK, que contiene el resultado de la conexión (aceptada, rechazada, etc).

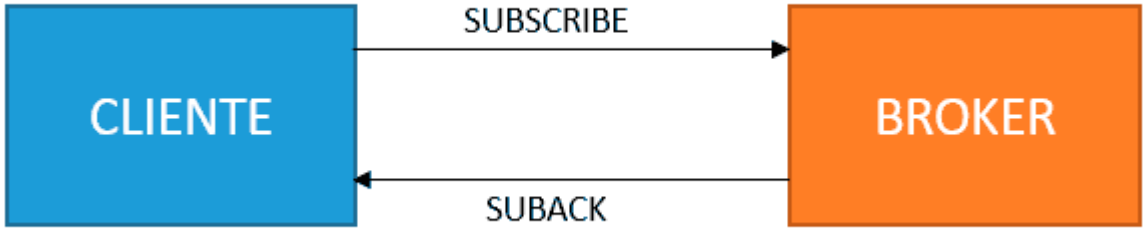


MQTT

- Para enviar los mensajes el cliente emplea mensajes PUBLISH, que contienen el topic y el payload.



- Para suscribirse y desuscribirse se emplean mensajes SUBSCRIBE y UNSUSCRIBE, que el servidor responde con SUBACK y UNSUBACK.



MQTT

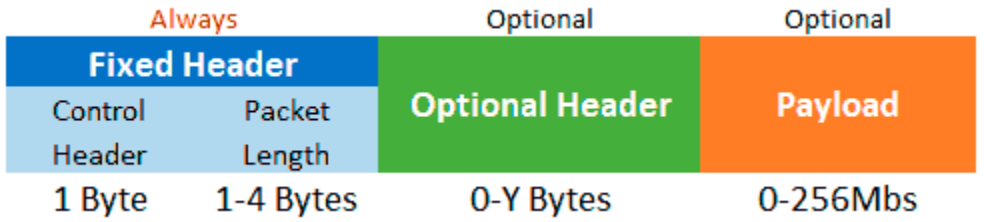
- Para asegurar que la conexión está activa los clientes mandan periódicamente un mensaje PINGREQ que es respondido por el servidor con un PINGRESP. Finalmente, el cliente se desconecta enviando un mensaje de DISCONNECT.

Calidad del servicio (QoS)

- MQTT tiene tres niveles QoS posibles:
  - **QoS 0 unacknowledged (at most one)**: El mensaje se envía una única vez. En caso de fallo por lo que puede que alguno no se entregue.
  - **QoS 1 acknowledged (at least one)**: El mensaje se envía hasta que se garantiza la entrega. En caso de fallo, el suscriptor puede recibir algún mensaje duplicado.
  - **QoS 2 assured (exactly one)**. Se garantiza que cada mensaje se entrega al suscriptor, y únicamente una vez.
- Tanto en QoS 1 como en QoS 2, los mensajes se guardan o se ponen en cola para los clientes que están desconectados y que tienen una sesión persistente establecida. Estos mensajes se reenvían una vez que el cliente vuelve a conectarse
- Usar un nivel u otro depende de las características y necesidades de fiabilidad de nuestro sistema.

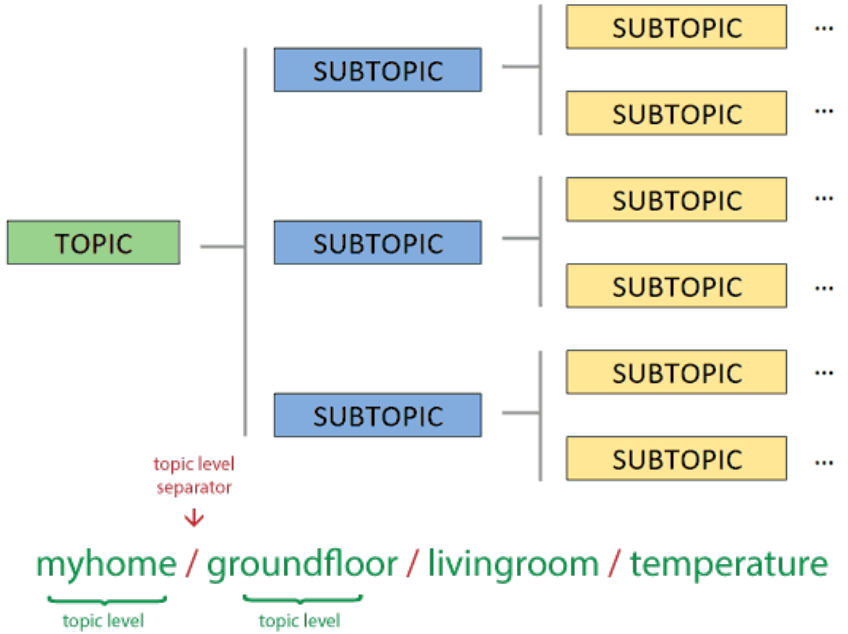
MQTT – Estructura de mensajes

- Cada mensaje consta de 3 partes:
  - **Cabecera fija**. Ocupa de 2 a 5 bytes, obligatorio. Consta de:
    - Código de control, que identifica el tipo de mensaje enviado.
    - Longitud del mensaje, que se codifica en 1 a 4 bytes, de los cuales se emplean los 7 primeros bits, y el último es un bit de continuidad.
  - **Cabecera variable**. Opcional, contiene información adicional que es necesaria en ciertos mensajes o situaciones.
  - **Contenido(payload).** contenido real del mensaje. Puede tener un máximo de 256 Mb aunque en implementaciones reales el máximo es de 2 a 4 kB.



Topics en MQTT

- **Los topics son estructuras** que utilizan el carácter de barra (/) como delimitador similar a la de un árbol de directorios.
- **El Broker acepta todos los Topics**. No es necesario crearlo explícitamente antes de publicar o suscribirse al Broker.
- **Los clientes pueden suscribirse a uno o varios Topic**. Para ello, el cliente puede establecer varias suscripciones y/o emplear Wildcards
- **Los clientes publican mensajes indicando un único Topic**. El Broker recibe el mensaje y, si encuentra alguna suscripción que cumpla con el filtro del Topic, transmite el mensaje a los clientes suscritos.



MQTT y Arduino

- Es necesario disponer de un Broker previamente configurado
- Instalar la librería PubSubClient en el IDE de Arduino
- Árbol de Topics a utilizar
- Se ha estructurado en 5 partes:
  - main.ino: sketch principal que inicia WiFi y la comunicación MQTT
  - config.h: configuración de la conexión WiFi
  - ESP32_Utils.hpp: contiene funciones que genéricas del ESP32
  - ESP32_Utils_MQTT.hpp: contiene funciones que genéricas para comunicación en MQTT
  - MQTT.hpp: funciones que sí son específicas del proyecto y definiciones del Broker

Ejemplo (main.ino)

```
#include <WiFi.h>
#include <AsyncMqttClient.h>
#include <ArduinoJson.h>

#include "config.h"  // Sustituir con datos de vuestra red
#include "MQTT.hpp"
#include "ESP32_Utils.hpp"
#include "ESP32_Utils_MQTT_Async.hpp"

void setup(void)
{
  Serial.begin(115200);
  delay(500);
  WiFi.onEvent(WiFiEvent);
  InitMqtt();
  ConnectWiFi_STA();
}

void loop()
{
  PublishMqtt();
  delay(500);
}
```

Ejemplo (config.h)

```
const char* ssid     = "ssid";
const char* password = "password";
const char* hostname = "ESP32_1";

IPAddress ip(192, 168, 1, 200);
IPAddress gateway(192, 168, 1, 1);
IPAddress subnet(255, 255, 255, 0);
```

Ejemplo (ESP32_Utils.hpp)

```
void ConnectWiFi_STA(bool useStaticIP = false)
{
   Serial.println("");
   WiFi.mode(WIFI_STA);
   WiFi.begin(ssid, password);
   if(useStaticIP) WiFi.config(ip, gateway, subnet);
   while (WiFi.status() != WL_CONNECTED)
   {
     delay(100);
     Serial.print('.');
   }

   Serial.println("");
   Serial.print("Iniciado STA:\t");
   Serial.println(ssid);
   Serial.print("IP address:\t");
   Serial.println(WiFi.localIP());
}
```

```
void ConnectWiFi_AP(bool useStaticIP = false)
{
   Serial.println("");
   WiFi.mode(WIFI_AP);
   while(!WiFi.softAP(ssid, password))
   {
     Serial.println(".");
     delay(100);
   }
   if(useStaticIP) WiFi.softAPConfig(ip, gateway, subnet);

   Serial.println("");
   Serial.print("Iniciado AP:\t");
   Serial.println(ssid);
   Serial.print("IP address:\t");
   Serial.println(WiFi.softAPIP());
}
```

Ejemplo (ESP32_Utils_MQTT.hpp)

```
void InitMqtt()
{
  mqttClient.setServer(MQTT_BROKER_ADRESS, MQTT_PORT);
  SuscribeMqtt();
  mqttClient.setCallback(OnMqttReceived);
}

void ConnectMqtt()
{
  while (!mqttClient.connected())
  {
    Serial.print("Starting MQTT connection...");
    if (mqttClient.connect(MQTT_CLIENT_NAME))
    {
      SuscribeMqtt();
    }
    else
    {
      Serial.print("Failed MQTT connection, rc=");
      Serial.print(mqttClient.state());
      Serial.println(" try again in 5 seconds");

      delay(5000);
    }
  }
}

void HandleMqtt()
{
  if (!mqttClient.connected())
  {
    ConnectMqtt();
  }
  mqttClient.loop();
}
```

Ejemplo (MQTT.hpp)

```
const char* MQTT_BROKER_ADRESS = "192.168.1.150";
const uint16_t MQTT_PORT = 1883;
const char* MQTT_CLIENT_NAME = "ESP8266Client_1";

WiFiClient espClient;
PubSubClient mqttClient(espClient);

void SuscribeMqtt()
{
  mqttClient.subscribe("hello/world");
}

String payload;
void PublisMqtt(unsigned long data)
{
  payload = "";
  payload = String(data);
  mqttClient.publish("hello/world", (char*)payload.c_str());
}
```

```
String content = "";
void OnMqttReceived(char* topic, byte* payload, unsigned int
   length)
{
  Serial.print("Received on ");
  Serial.print(topic);
  Serial.print(": ");

  content = "";
  for (size_t i = 0; i < length; i++) {
    content.concat((char)payload[i]);
  }
  Serial.print(content);
  Serial.println();
}
```