Comunicación Cliente – Servidor

Comunicación Cliente – Servidor

- **Cliente**
  - Conexión WiFi en la misma red del servidor
  - Recibe y envía mensajes MQTT
- **Servidor**
  - Servidor MQTT (Mosquitto)
  - Interfaz de control (NodeRED)

Comunicación Cliente – Servidor

- **WiFi**
  - Modos de operación
    - Estación (STA_IF): se conecta a redes existentes
    - AP (AP_IF): permite que otros dispositivos se conecten a ella

Comunicación Cliente – Servidor

- **WiFi (Estación) – Consulta de redes disponibles**

```
1  import network
2
3  wlan = network.WLAN(network.STA_IF)
4  if not wlan.active():
5      wlan.active(True)
6
7  redes = wlan.scan()
8
9  for red in redes:
10     print(f"SSID: {red[0].decode()}")
11     print(f"Canal: {red[2]}")
12     print(f"RSSI: {red[3]}dBm")
13     print("----")
```

Devuelve una lista en la que cada elemento contiene:

- [0] SSID
- [1] BSSID
- [2] Canal
- [3] RSSI
- [4] Modo de autenticación
- [5] Si está oculta

Comunicación Cliente – Servidor

- **WiFi (Estación) – Conexión a una red conocida**

```
1  import network
2  from time import sleep_ms
3
4  wlan = network.WLAN(network.STA_IF)
5  if not wlan.active():
6      wlan.active(True)
7
8  if not wlan.isconnected():
9      wlan.connect("ssid", "contraseña")
10
11     print("Conectando...")
12     while not wlan.isconnected():
13         sleep_ms(1000)
14
15 config = wlan.ifconfig()
16 print(f"Conectado con ip {config[0]}")
```

Inicializa la interfaz de red

Conecta a una red

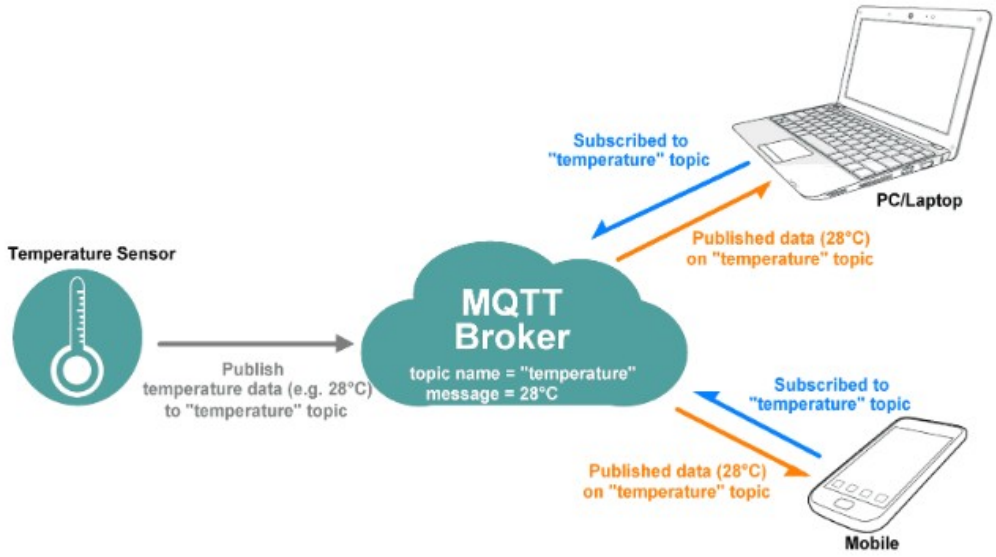
Muestra la IP actual

Comunicación Cliente – Servidor

- **MQTT**
  - Protocolo de mensajería estandarizado que distribuye la información a través de publicación y suscripción (publish/subscribe) a determinado tema (topic)
  - Es eficiente para situaciones que se transporta poca información ya que consume poco ancho de banda

Comunicación Cliente – Servidor

- **MQTT – Funcionamiento**
  - La conexión entre el subscriber y el publisher se encuentran desacopladas.
  - La comunicación es manejada por un tercer componente llamado broker (servidor)
  - Un cliente puede suscribirse o publicar en diferentes topics en simultáneo
  - Una red MQTT responde a una topología estrella



Comunicación Cliente – Servidor

- **MQTT – Formato de topics sugerido**
  - Único dispositivo: **/<parámetro>**
  - Múltiplies dispositivos: **/<dispositivo>/<parámetro>**
  - Ejemplos:
    - /humedad
    - /cocina/temperatura

Comunicación Cliente – Servidor

- **MQTT – Instalación de cliente en MicroPython**
  - En línea de comandos de MicroPython:

```
>>> import upip
>>> upip.install("micropython-umqtt.simple")
  Installing to: /lib/
  Warning: micropython.org SSL certificate is not validated
  Installing micropython-umqtt.simple 1.3.4 from https://micropython.org/
  pi/umqtt.simple/umqtt.simple-1.3.4.tar.gz
>>> upip.install("micropython-umqtt.robust")
  Installing to: /lib/
  Installing micropython-umqtt.robust 1.0.1 from https://micropython.org/
  pi/umqtt.robust/umqtt.robust-1.0.1.tar.gz
```

Comunicación Cliente – Servidor

- **MQTT – Cliente**
  - umqtt.simple: provee la funcionalidad básica del cliente de MQTT
  - umqtt.robust: agrega la capacidad de reconectar en caso pérdida de conexión

Comunicación Cliente – Servidor

- **MQTT – Creación del cliente**

```
28 from umqtt.robust import MQTTClient
29 cliente = MQTTClient("nombre", "servidor", keepalive=30)
30 print("Conectando a servidor MQTT...")
31 cliente.connect(clean_session=False)
32 print("Conectado")
```

  - Nombre: nombre arbitrario del cliente conectado. Tiene que ser único en la red
  - Servidor: dirección del servidor
  - Keep alive: verifica cada cierto intervalo de tiempo que el dispositivo siga conectado

Comunicación Cliente – Servidor

- **MQTT – Suscripciones**
  - Escucha continuamente topics
  - Ejecuta una función *callback* al recibir un mensaje con los parámetros:
    - Topic
    - Mensaje
  - Los parámetros llegan como *bytes*: deben convertirse a texto con el método .decode()

Comunicación Cliente – Servidor

- MQTT – Suscripciones

```
19 def callback(topic, msg):
20     print(f"Llegó {msg.decode()} de {topic.decode()}")
21
22 cliente = MQTTClient("nombre", "servidor", keepalive=30)
23 print("Conectando a servidor MQTT...")
24 cliente.set_callback(callback)
25 cliente.connect(clean_session=False)
26 cliente.subscribe("#")
27 print("Conectado")
28
29 while True:
30     cliente.check_msg()
31     sleep_ms(500)
```

Función *callback*

Configuración

Leo mensajes nuevos

Comunicación Cliente – Servidor

- **MQTT – Publicaciones**
  - No lleva ninguna configuración adicional más que la creación del cliente
  - No requiere *callback*
  - Sintáxis:
    - cliente.publish(topic, mensaje)

Comunicación Cliente – Servidor

- **MQTT – Ejemplo**
  - Escribir un programa que se conecte a un servidor MQTT y emita un mensaje cada un segundo en un topic arbitrario en */alumnos/* y muestre los mensajes que escucha desde el topic */servidor*. Usar MQTT-Explorer para enviar y recibir mensajes.

Comunicación Cliente – Servidor

- **Ejercicio**
  - Reutilizando el programa anterior, agregar el manejo de los siguientes mensajes:
    - Suscripciones
      - **/alumnos/<alumno>/led_verde**: encender el LED verde si se recibe 1, sino apagar (repetir para el resto)
      - **/alumnos/<alumno>/servo:** leer el número recibido en el rango 0-180 y ubicar el servo en esa posición
      - **/alumnos/<alumno>/neopixel**: se reciben tres valores separados por comas (R,G,B) y encender el NeoPixel como corresponda (ayuda: usar método .split(“,”) en el mensaje)
    - Publicaciones
      - **/servidor/<alumno>**: al pulsar uno de los botones, enviar la posición del potenciómetro