Lab9: UMQTT

Introduction

This lab cover the installation and use of 'umqtt.simple', the MicroPython version of the Paho MQTT client software, and develop some IoT applications to capture and publish data.

Learning outcomes

1. Use umqtt.simple to publish values to an MQTT broker
2. Use a sensor to publish temperature values to an MQTT broker

Task 1: Using umqtt.simple Library module

1. At the REPL, type `import umqtt.simple` – if all is well there will be no errors (Thonny can find the module and import it).

2. The first program we will create will be a MicroPython version of the random number publishing program you wrote and tested in Lab 6. The code is shown below:

```
1  from umqtt.simple import MQTTClient
2  import time, urandom
3  server = 'broker.hivemq.com'
4  c = MQTTClient('ESP32_yyy', server)   #yyy = your initials
5  c.connect()
6  for i in range(20):
7      value = urandom. randrange(100)
8      c.publish(b'CM125/yyy', str(value))
9      print(value)
10     time.sleep(3)
```

   Save it along with your other ESP32 files. Note that it will publish 20 random values in the range 0-100, one every 3 seconds. Change the range value if you want it to run for longer.

3. Before you can run it you need to connect your ESP32 to the network (use the code from last week). When it runs it will print the random values to Thonny AND publish them to the HiveMQ broker.

4. Now substitute the IP address if your Ubuntu machine as the server. The broker on your machine is not running yet, so set it running using the -v switch so that you can see messages, and re-start your modified code.

5. Then set up a Mosquitto subscribing client to read your published messages, and confirm that everything is working.

6. Use Wireshark to see how the name 'ESP32yyy' is present in the exchanges to the broker.

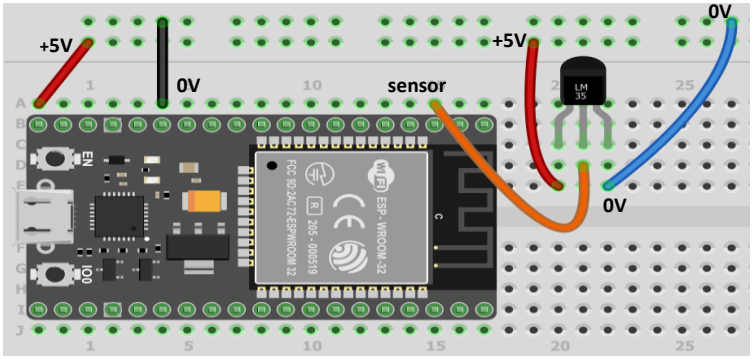
7. Use the MQTT-Explorer client to check that it works by getting it to create a graph of the output.

You now have a functioning IoT MQTT publishing system running on a low-power IoT device, and you have confirmed the operation using 2 different brokers and a range of diagnostic techniques.

Task 2: Temperature Measurement

The simplest of IoT systems often consist of a sensor and publishing system. Task 2 implements a temperature sensing system.

1. Get an LM35 temperature sensor from your pack. Make sure you can identify the ground, signal and Supply pins (look it up on the Internet if necessary). Get some connecting wires. Build the following circuit:



Check that it is correct – seek help if you are not 100% sure before powering it up.

2. The LM35 outputs a small voltage of 10mV/°C, so if we measure the voltage, and multiply by 100, we can determine the temperature. An 'ADC' pin like pin 'C1' can measure a voltage and provide a number, n, in the range 0 – 4095 representing the range 0 – 1.0 V using the following command:

```
n = adc.read()     # n is between 0 and 4095
```

3. The partially completed code below will print 'n'. Try it out in Thonny

```
#  ESP32 LM35 code
from machine import ADC, Pin
import time
adc = ADC(Pin(36))  #Pin 36 is analog in

for i in range (20):
    n=adc.read()
    print(n)
    time.sleep (1)
```

The code should continuously output a value of around 600 for a typical ambient temperature in the low 20's. If you warm up the LM35 by holding it between your fingers (whilst the circuit it still running) you should find that it can rise to as much as 900. This confirms that the system is functioning correctly.

4. The value of ‘n’ is proportional to temperature. To get the actual temperature, you need to convert ‘n’ into a voltage, V (where V = n/4096). THEN you need to convert the voltage into a temperature, recalling that the output voltage is 10mV/degree C. Write the code to do this and satisfy yourself that the system is outputting realistic temperature values.

Task 3: Putting it all together

At this point you have two separate systems, one for publishing, and one for sensing. You need to combine these two systems so that you can take the data from the sensing system and publish it to HiveMQ.

Display the temperature as a graph on MQTT-Explorer, noting that this data is now available to anyone, anywhere in the world, who has an Internet connection.

The forthcoming assignment will require all the skills mentioned in the labsheet.

Challenge: Controlling remote device

The blue LED on an ESP32 could be controlled if the ESP32 subscribed to a topic which published messages like ’ON’ and ‘OFF’. The publishing client could be any of the following:

a) mosquitto_pub client
b) MQTT-Explorer (it can publish and subscribe)
c) Python3 ‘Paho’ client on an Ubuntu machine
d) Another ESP32
e) Mobile phone running a client like MQTT Dashboard or MQTT Snooper (both Android)

If you can meet the above challenges, then you will be well-prepared for the assignment.

(solution overleaf)

```
from umqtt.simple import MQTTClient
import time, urandom
from machine import Pin

def command(topic, msg):
    print('I have received a command:', msg )
    if msg == b'on':
        LED.on()
    if msg == b'off':
        LED.off()

LED=Pin(2, Pin.OUT)
server = 'broker.hivemq.com'
c = MQTTClient('MyESP32', server)
c.set_callback(command)
c.connect()
c.subscribe(b'LED_control')
while True:
    c.wait_msg()

```