Table of Contents

1. IoT Networks: Laboratory 2 - MQTT 2
1.1. Hardware 2
1.1.1. Hardware-software components and libraries 2
1.2. Software 4
1.2.1. Software libraries 4
1.3. Networking 8
1.4. Network services 8
1.4.1. MQTT 8
1.5. Laboratory scenarios 9
1.5.1. Scenario 1 10
1.5.2. Scenario 2 11
1.5.3. Scenario 3 12

1. IoT Networks: Laboratory 2 - MQTT

Network programming for the IoT Devices using MQTT protocol.

The following manual assumes you're familiar with InT Networks: Laboratory 1 - Introduction. In case you're not, please get familiar with it and spend some time practicing. Start your project stub using hints and guidance in the InT Networks: Laboratory 1 - Introduction manual and modify it / extend it as guided below.



This manual is not intended to be stand-alone. You need to follow the Laboratory 1 manual to prepare the basics for the project, then continue here.

1.1. Hardware

You will use a sensor that measures temperature + humidity to publish this information to the MQTT broker:

- DHT 11 temperature and humidity sensor



Fig. 1: DHT 11 temperature and humidity sensors

1.1.1. Hardware-software components and libraries

Interfacing hardware requires the use of particular GPIOs. Laboratory nodes and hardware used need specific configuration. The following chunks of code will help to set up proper hardware-software configuration.

1.1.1.1. LCD display

Built-in LED on the microcontroller development board can help to manage simple tracing of the code execution:

```
#define LED_BUILTIN 14
```

Your LCD shield is a 2×16 character one, driven by a Hitachi HD44780 controller. You can use a library to use it easily. The PlatformIO will pull the library automatically if defined in *platformio.ini* file:

```
lib_deps = arduino-libraries/LiquidCrystal
```

```
#include <LiquidCrystal.h>
```

To initialize it and instantiate the controller object, use the following GPIO definition and initialization:

```
const int rs = 12, en = 13, d4 = 4, d5 = 0, d6 = 2, d7 = 14;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
```

It is also necessary to tell the library about the display organization as various configurations are possible. Here, the organization is 2 lines, 16 columns each, so in the code, you need to explicitly initialize it:

```
...
lcd.begin(16,2);
...
```

1.1.1.2. Temperature and humidity sensor

There is a DHT11 (blue) sensor connected to the GPIO 5.
There are two libraries needed to handle reading from the sensor: a common sensor communication library and a specific one for DHT sensors, both by Adafruit, so two more entries are needed in the *platformio.ini* file, and you need to refer to it in the source code:

```
lib_deps =
    adafruit/Adafruit Unified Sensor@^1.1.4
    adafruit/DHT sensor library@^1.4.3
```

```
#include <Adafruit_Sensor.h>
#include <DHT.h>
```

To access sensor readings, it is necessary to instantiate a communication class that binds the hardware to the code:

```
#define DHTTYPE DHT11   // DHT 11
#define DHTPIN 5
DHT dht(DHTPIN, DHTTYPE,50);
```

It is also necessary to initialize the sensor class and give it some time to set up:

```
...
dht.begin();
delay(1000);
...
```

The sensor class returns data (among others) in a float containing temperature (in C) and relative humidity (in percent):

```
...
  float hum = dht.readHumidity();
  float temp = dht.readTemperature();
...
```

1.2. Software

String formatting may be a challenge. A straightforward approach is to allocate a character buffer of the size of the LCD or just one display's line or any other size large enough to handle your text.

```
...
char buffer [50];
...
```

To convert float into the string with formatting, use `sprintf` function:

```
...
  sprintf(buffer,"%2.1f C",temp);
...
  sprintf(buffer,"%2.1f%%Rh",hum);
...
```



`sprintf` uses a number of wildcards rendered with data. Refer to the c/c++ documentation on `sprintf`. Here

```
%2.1f
```

means: having a float number render it to string using two digits before the decimal point and one after. The temperature in our lab is always above 10C.

```
%%
```

is an escape character to print a

```
%
```

(percent) symbol.

```
delay(time)
```

is measured in milliseconds.

1.2.1. Software libraries

Software libraries are needed to communicate between your IoT device and network and services. In this laboratory, you will connect your device to the WiFi access point (see the Networking section below for credentials) and use TCP. Your device will be automatically given an IP address that you may show on the LCD screen. Once WiFi is connected and works (PHY+MAC+NET layer protocols), then you can connect over it to the MQTT Broker (APP layer protocol).
Mind, the connection may become broken, so ensure a re-connection mechanism:

- dummy: in case the connection is broken, restart the device to let it perform the connection process again;
- smart: include checking and re-connection into your code using connection status tracking, mind you need to reconnect WiFi and high-level communication protocol: MQTT.

1.2.1.1. WIFI and TCP

A WiFi library is already included in the ESP8266 framework for the Arduino programming model, so there is no need to use a library manager to add it. All that is necessary is to have a header included to your C code:

```
...
#include <ESP8266WiFi.h>
...
```

Then create some constant variables or declare token string identifiers (as below) with `sSsid` and `sPassword` (see Networking section for credentials) and instantiate the WiFi communication object:

```
...
#define sSsid "<here comes ssid to connect to>"
#define sPassword "<here comes a password>"
...
```

The `WiFi` object is a singleton, static one. Try to connect to the AP, i.e., this way:

```
...
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
...
```

Some useful code snippets are below.
Checking if the WiFi connection is successful:

```
...
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
...
```

Checking the current IP address:

```
...
  Serial.println(WiFi.localIP());
...
```

Once WiFi is connected, it is necessary to use a `WiFiClient` object to pass the connection object to the application layer libraries.
The declaration can be as follows:

```
...
  WiFiClient espClient;
...
```

Note this object becomes automatically active and bound to the `WiFi` connection when available. You do not need explicitly to initialize it or force connect it.

1.2.1.2. MQTT

To ease and speed up implementation, you will use an MQTT library that implements acting both as a publisher and subscriber `platformio.ini` file:

```
lib_deps = ..., PubSubClient, ...
```



Please note, on the NET protocol level, in this scenario, you're connecting to the WiFi AP (STA), not setting your device to be a WiFi server (AP), so you need to set the appropriate WiFi mode, as it may be previously used in AP, not STA mode.

MQTT clients need to be instantiated using a WiFi client connection, i.e., this way:

```
...
#define mqtt_server "<here is your MQTT broker IP or FQDN>"
#define mqtt_user "<mqtt user>"
#define mqtt_password "<mqtt password>"
...
PubSubClient client(espClient);
...
```

and in the setup code, you need to initialize the `client` with the MQTT address and port number. Do not use SSL / TLS encoding, as this library is not able to support encrypted communication with an MQTT broker:

```
...
client.setServer(mqtt_server, 1883);
...
```

Consider encapsulating MQTT connecting process as a separate function, i.e., as below. Note the code below publishes a hello world message to the broker to emphasize its successful presence in the network:

```
void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Create a random client ID
    String clientId = "ESP8266Client-";
    clientId += String(random(0xffff), HEX);
```

```
    // Attempt to connect
    if (client.connect(clientId.c_str(),mqtt_user,mqtt_password)) {
      Serial.println("connected");
      // Once connected, publish an announcement...
      client.publish("outTopic", "hello world");
      // ... and resubscribe
      client.subscribe("#");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}
```

Receiving an MQTT message is done with the use of the asynchronously called callback function, i.e., like this one (it switches on and off the LED):

```
...
void callback(char* topic, byte* payload, unsigned int length) {
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  for (int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
  }
  Serial.println();

  // Switch on the LED if an 1 was received as first character
  if ((char)payload[0] == '1') {
    digitalWrite(BUILTIN_LED, LOW);   // Turn the LED on (Note that LOW is
the voltage level
    // but actually the LED is on; this is because
    // it is active low on the ESP-01)
  } else {
    digitalWrite(BUILTIN_LED, HIGH);  // Turn the LED off by making the
voltage HIGH
  }

}
...
```

You also need to bind the callback function with the MQTT client:

```
...
  client.setCallback(callback);
...
```

And then periodically execute the communication routine for the handling of the incoming messages,

usually in the `loop()` function of your code (unless using FreeRTOS tasks asynchronous model):

```
...
  client.loop();
...
```

1.3. Networking

There is a WiFi 2.4GHz, IPv4 network available to the students. This is a PRIVATE network. This network is separated from the Internet, but all connected devices can talk to one another. There is a DHCP server with networking addressing scope 192.168.90.10-192.168.90.254.



Do not use static IP! Some services in the network have fixed IP addresses; overlapping causes the unavailability of your solution and this service, also to other teams! Use only DHCP for obtaining a network address.

To connect to the wireless network, use the following credentials:

- SSID: internal.IOT
- Passkey: IoTlab32768
- IP address: DHCP, 192.168.90.10-192.168.90.254

1.4. Network services

1.4.1. MQTT

There is an MQTT Broker, a binding WiFi PRIVATE network, and the PUBLIC Campus network. This way, you can route MQTT messages from your end node devices connected to the `internal.IOT` network to the external tools, i.e., cloud-based dashboard or storage via, i.e., publishing from your end node IoT device through the PRIVATE network to MQTT Broker and subscribing to those messages from the cloud using PUBLIC Campus network. Please note the context (thus, the IP address of the MQTT broker) changes depending on which device and network you're using. The following network schema should be of help 2.

You can access the MQTT broker from the Campus PUBLIC network using the following configuration and credentials:

- Address: 157.158.56.54
- Port: 1883 for plain text, 8883 for secure communication
- User: vrel
- Password: vrel2018

You can also access the MQTT broker from the `internal.IOT` PRIVATE network using the following configuration and credentials:

- Address: 192.168.90.5
- Port: 1883 for plain text, 8883 for secure communication

- User: vrel
- Password: vrel2018

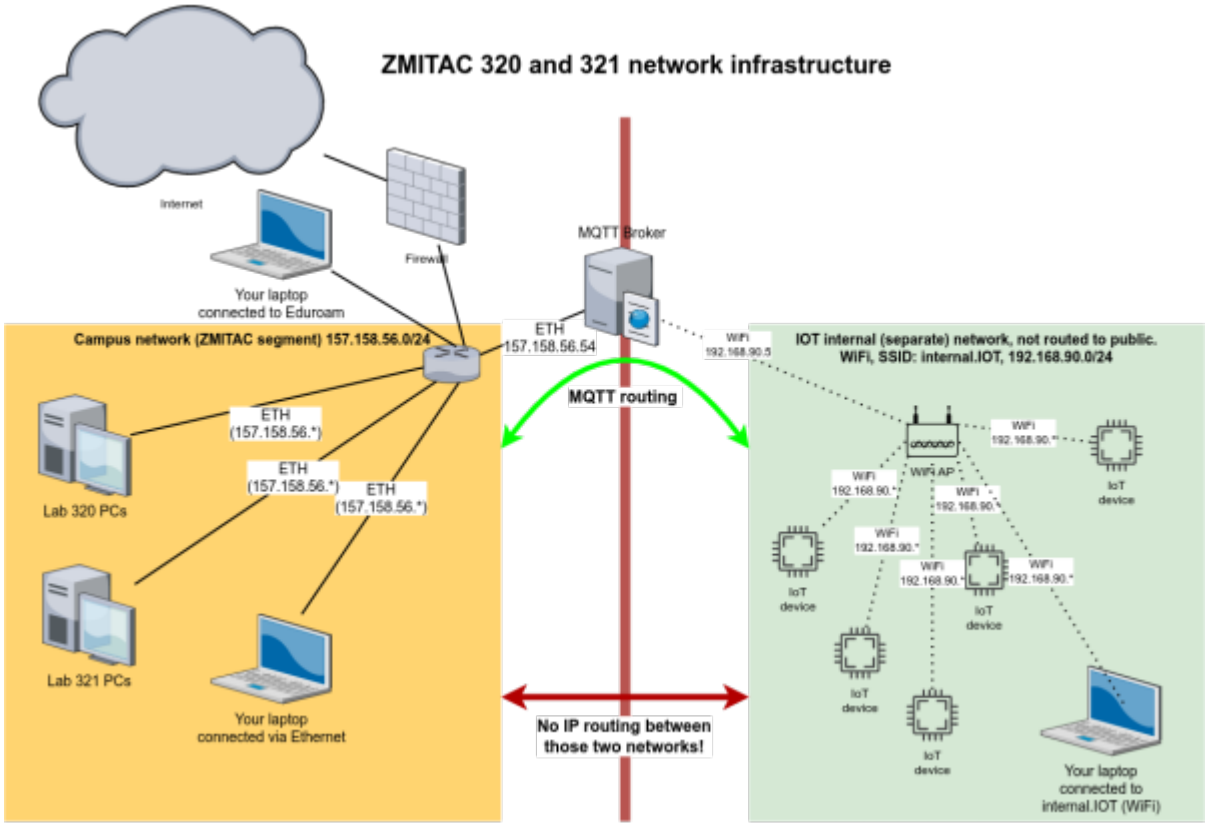


Fig. 2: Zmitac network infrastructure for the IoT labs

1.5. Laboratory scenarios

All teams share the same scenario yet, please note that, in particular, MQTT messages exchanged between devices shall be coordinated among team members not to overlay one another.
You can implement 3 separate codes or bind them into one solid solution.



Mind keeping your code between meetings. It is regular that at every next laboratory meeting, you will extend your code by something extra.

1.5.1. Scenario 1

** Connecting to the WiFi **
Make your IoT device connect to the Internal.IOT network and present the device's MAC and IP address on the LCD screen.

Report:

- Describe your approach to the WiFi re-connection and document IP and MAC with a screenshot.

1.5.2. Scenario 2

** Sending MQTT message **
Read temperature and humidity and send it as two separate MQTT messages (as payloads).



Warning! Agree with other sections to use UNIQUE MQTT client names and MQTT topics. Otherwise, you will interfere with one another on the broker level.

Observe your messages and their change over time using any software-based MQTT client (i.e., MQTT Explorer or MQTT Spy apps).

Report:

- Describe your interface, including client name, MQTT topics, publishing frequency, etc.

1.5.3. Scenario 3

** Receiving MQTT message **
Prepare a solution where your IoT device subscribes to the selected message that will be received, and the payload will be present on the LCD screen. You can assume the maximum payload size is 16 characters to fit a single LCD display line. You should be able then to send any text to your LCD remotely. Use 2 separate messages to control the text in line 0 and line 1 (first and second) of the LCD screen.
Use MQTT Spy or MQTT Explorer app to send a message with a payload.

Report:

- Document your interface (topic, payload format). Present your approach to buffer the message.

From:
https://dokuwiki.zmitac.aei.polsl.pl/ - **RAU6 Dokuwiki**

Permanent link:
https://dokuwiki.zmitac.aei.polsl.pl/en:edu:zmitac:s2:iotn:lab2

Last update: **2023/04/21 16:42**