Table of Contents

1. IoT Networks: Laboratory 3 - CoAP 2
1.1. Hardware 2
1.1.1. Hardware-software components and libraries 2
1.2. Software 3
1.2.1. Software libraries 4

1. IoT Networks: Laboratory 3 - CoAP

Network programming for the IoT Devices using CoAP protocol.

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



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



Computers in laboratory rooms 320, 321 and 336 use freezing software which means that for every startup, computers are restored to their initial state.

1.1. Hardware

We do not use any specific hardware here other than an LCD screen to present visual information. If you intend to extend your project and i.e. use a temperature and humidity sensor, refer to the IoT Networks: Laboratory 2 - MQTT documentation on necessary libraries and configuration.

1.1.1. Hardware-software components and libraries

Interfacing hardware requires the use of particular GPIOs. Laboratory nodes and hardware used needs specific configuration. 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 initialise it and instantiate controller object, use following GPIO definition and initialisation:

```
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 organisation as there is a variety of configurations possible. Here, the organisation is 2 lines, 16 columns each, so in the code, you need to explicitly initialise it:

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

1.2. Software

String formatting may be a challenge. An easy 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 number of wildcards that are rendered with data. Refer to the c/c++ documentation on `sprintf`. Here

```
%2.1f
```

means: having float number render it to string using two digits before decimal point and one after. 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 handle communication 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 UDP (opposite to the Lab2 where you have used 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 instantiate a CoAP client and ask the laboratory server or any other service present in your network (i.e. other device acting as CoAP server) for data or activity.

The great feature of the CoAP is, you do not need to take care of the instant connection to the service on the application-level protocol. Just reconnecting WiFi, if the connection is dropped, is quite enough. UDP is a “connectionless” protocol and acts more-less as “send and forget”, at least on the single message level. Obviously, ACK messages are expected but this approach is so much different to the MQTT on TCP, where connection break is instantly detected and yielded if only happens.

1.2.1.1. WIFI and UDP

A WiFi library is already included in the ESP8266 framework for the Arduino programming model, so there is no need to use library manager to add it. All is necessary is to include a header 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 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 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 `WiFiUDP` object to pass the connection object to the application layer libraries.
The declaration can be as follows:

```
...
  WiFiUDP Udp;
...
```

Note, this object becomes automatically active and bound to the `WiFi` connection, when available. You do not need explicitly to initialise it or force connect it, however, you must initialise an object with an appropriate port. The default port for CoAP server is 5683 (UDP):

```
...
int port = 5683; //Standard CoAP port
```

```
...
  Udp.begin(port);
...
```

1.2.1.2. CoAP

To ease and speed up implementation, you will use a CoAP library that implements acting both as a client and server. It is necessary to inform the library manager to download and include `hirotakaster/CoAP simple library` in the configuration section of the `platformio.ini` file:

```
lib_deps = ..., hirotakaster/CoAP simple library, ...
```



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 appropriate WiFi mode, as it may be previously used in AP, not STA mode.

CoAP client or server both require to be instantiated using WiFi UDP connection (aforeinitialised `Udp` object), i.e. this way:

```
...
Coap coap(Udp);
...
```

Because UDP is asynchronous due to its nature, each transaction (sending a request and waiting for the response) must be handled asynchronous way. For this purpose, we need to use callbacks. The sample callback function may have the following parameters:

```
...
void callback_response(CoapPacket &packet, IPAddress ip, int port); //this is only prototype
...
```

As code runs asynchronously and many messages may come one by one, you should not process the `packet` itself but rather make a copy as quickly as possible, to release resources for handling another request (if any):

```
...
void callback_response(CoapPacket &packet, IPAddress ip, int port) {
    char p[packet.payloadlen + 1];
    memcpy(p, packet.payload, packet.payloadlen);
    p[packet.payloadlen] = NULL;
    //and then process "p" here
    ...
}
...
```



You may have a variety of frames received but the common is to “ping” someone. If you detect `packet.type==3 && packet.code==0` then you were pinged. The following code snippet may help to identify ping and distinguish from the data packets:

```
...
    if(packet.type==3 && packet.code==0){
      Serial.println("COAP ping ok");
    }
...
</note>

Once your callback function is ready it is time to send a request and wait for the callback function to be triggered when the response comes:
<code c>
...
  coap.response(callback_response);
  coap.start();
...
```



Warning! You MUST configure and bind all response callback functions

PRIOR to starting a CoAP communication, as below

Once the infrastructure is ready you can send a request and observe a response, i.e. in a `void loop()` function:

```
int msgid = coap.get(ip,port,"");
bool state = coap.loop();
```

Note, in the code above, `coap.get(…)` has three parameters:

1. IP address of the server you want to connect to (`IPAddress` object);
2. port number (`int`);
3. URL, or actually its part coming AFTER the IP section of the URL.

Regarding last parameter, “” denotes root, so in fact the request URI is composed as follows: `coap://<IP>:<port>/<URL>` ==== 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 to it can talk to 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 present in the network have fixed IP addresses, overlapping it causes unavailability of your solution and this service, also to other teams! Use the 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

==== Network services ====

=== MQTT === There is an MQTT Broker, binding WiFi PRIVATE network and the PUBLIC Internet. 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 Internet.

You can access it from the Internet 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 it from the aforementioned 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



==== 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 every next laboratory meeting you will extend your code by something extra.



=== Scenario 1 === ** Connecting to the WiFi **
Make your IoT device connect to the Internal.IOT network and present MAC of the device and IP address on the LCD screen.

Report:

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

=== 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 changing over time using any software-based MQTT client (i.e. MQTT Explorer or MQTT Spy apps).

Report:

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



=== 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 line of the LCD display. 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 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:lab3?rev=1647638779

Last update: **2022/03/18 22:26**