Práctica 3

Aplicación CoAP

Jorge Alejandro Flores Triana

ID: 714510

ITESO, Universidad Jesuita de Guadalajara

Instituto Tecnológico y de Estudios Superiores de Occidente

Profesor – Sergio Nicolás Santana

Tópico: Sistemas de Comunicación

Aplicación CoAP

Link Video: https://drive.google.com/drive/folders/18rpOBIPAgmBrwcaXkDslcyxDzO0WqaTx?usp=sharing

Link Código y Reporte: https://github.com/a01633675/SistemasComunicacion

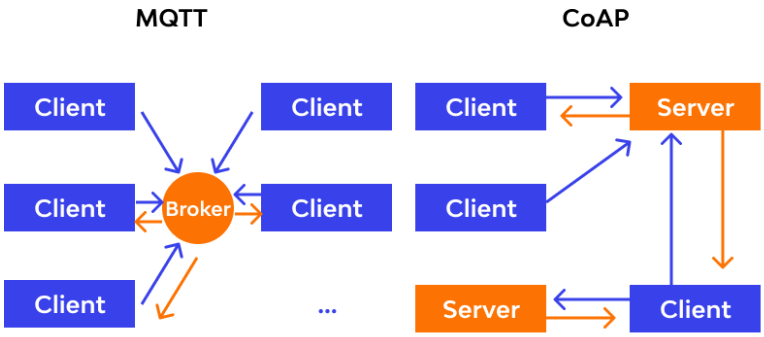
Objetivo:

El objetivo de esta práctica es implementar un dispositivo conectado a una red Wi-Fi que interactúe con otros módulos en la red local a través de CoAP. Además, utilizar el protocolo mDNS para que el dispositivo pueda ser fácilmente encontrado por otros en la misma red.

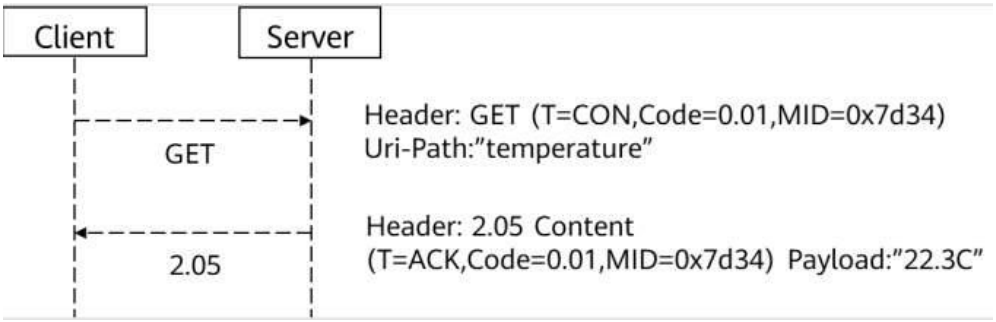
Marco de Referencia:

CoAP es un protocolo de aplicación utilizado en redes Wi-Fi conversacionales porque se basa en solicitudes y respuestas. Al contrario del protocolo MQTT que tiene un diseño de publicación/subscripción.

En un nivel inferior, los mensajes de CoAP se envían y reciben a través del Protocolo de Datagramas de Usuario (UDP), que no es confiable por naturaleza. Por lo tanto, para mayor seguridad, los mensajes se pueden enviar utilizando el protocolo Seguridad de la Capa de Transporte de Datagramas (DTLS) [1].



CoAP mantiene la misma arquitectura cliente-servidor y solo admite un subconjunto de métodos que son GET, PUT, POST y DELETE, así como un subconjunto de posibles códigos de respuesta. También es posible realizar solicitudes agregando la función de vigilancia/observador, que permite al cliente recibir actualizaciones sobre un determinado recurso que ha solicitado previamente [2].



Implementación:

La aplicación que se busca implementar es para monitorear un robot de rescate. Los robots de rescate se han utilizado desde hace tiempo ya que su tecnología ha avanzado de manera importante a lo largo de los años gracias a su fiabilidad mecánica cuando se emplean sobre el terreno.

Los robots de rescate están diseñados especialmente para ayudar en la búsqueda y el rescate de personas tras un desastre natural como los terremotos. Por lo que deben monitorear distintas variables en el entorno:

- Velocidad de los motores [Duty Cycle %]: Ayuda a controlar la forma en la que se desplaza el robot, gracias a un enconder. Información de la aplicación al microcontrolador.

- Sentido de Giro: Ayuda a controlar la dirección del robot. Información de la aplicación al microcontrolador.

- Temperatura del Ambiente [°C]: Ayuda a determinar si hay fuentes de calor cercanas que representen una amenaza. Información del microcontrolador a la aplicación.

- Sensor de Humo/Gas: Ayuda a saber si hay una fuga de gas cercana o algún otro material tóxico en el aire, así como posibles incendios. Información del microcontrolador a la aplicación.

- Sensor de Corriente [A]: Ayuda a conocer el consumo de corriente de los motores para saber si hay algún riesgo de dañar la circuitería interna. Esto es común ya que el robot se mueve en terrenos abruptos y muchas veces puede quedar atorado, causado estrés en los motores. Información del microcontrolador a la aplicación.

Cada variable descrita anteriormente está relacionada con un URI establecido:

- Espressif/speed
- Espressif /direction
- Espressif /temperature
- Espressif /gas
- Espressif /current

Finalmente, varias funciones que venían del ejemplo de Espressif CoAP fueron utilizadas. Las más importantes son:

- `espressif_get ( )`: El método GET obtiene la información del recurso identificado por el URI de la solicitud. En caso de éxito, el servidor devuelve un código 200 (OK) como respuesta..

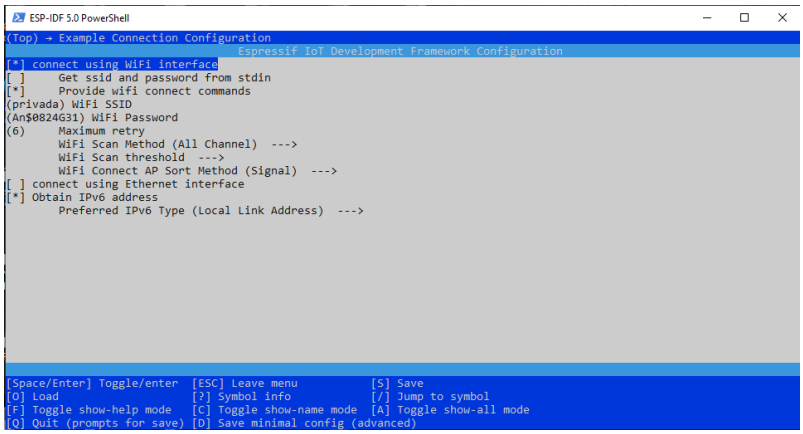
- `espressif_put ( )`: El método PUT se utiliza para actualizar el recurso identificado por el URI de la solicitud. Si existe un recurso en ese URI, el cuerpo del mensaje debe contener las modificaciones del recurso, y debería devolverse una respuesta 200 (OK). Si no existe ningún recurso en la URI de la solicitud, el servidor puede crear uno nuevo con ese URI devolviendo un código 201 (Creado) como respuesta

- `espressif_delete ( )`: El método DELETE solicita que se elimine el recurso identificado por el URI de la solicitud. En caso que se pueda eliminar correctamente el recurso, el servidor debería devolver un código 200 como respuesta.

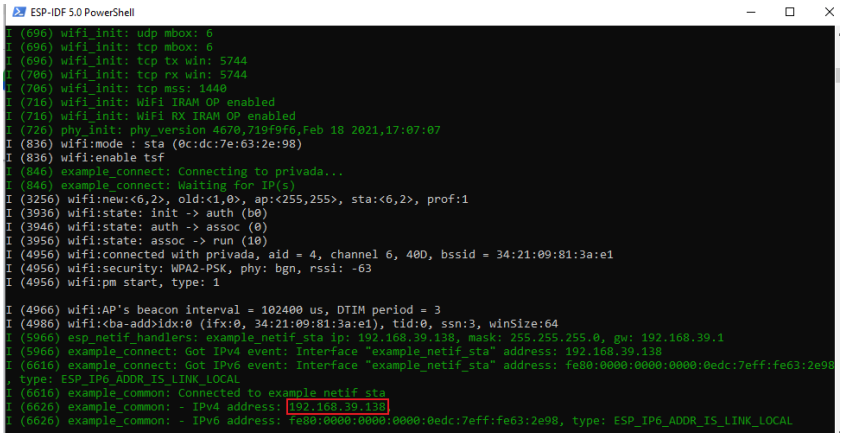
Resultados:

Primero que nada se descargó el IDE para trabajar con el ESP32. Posteriormente se cargó el código de *frameworks\esp-idf-v4.4\examples\protocols\coap_server* para utilizarlo como base.

En esta práctica, con ayuda del comando idf.py menuconfig hay que configurar la red de Wi-Fi y la contraseña que serán utilizados por el ESP32.



Para probar el código, se realizó una pequeña aplicación en donde la ESP-32 era el servidor y el script de Python era el cliente. Para esto, fue necesario obtener la dirección IP de la tarjeta ESP-32 y agregarla al script de Python




```python
import logging
import asyncio

from aiocoap import *

# put your board's IP address here
ESP32_IP = "192.168.39.138"
```

Aquí es importante recalcar que en los requerimientos originales de la práctica se debió agregar mDNS para identificar a la tarjeta a través de un nombre significativo. Sin embargo, debido a que la integración dentro de Espressif versión 5.0 era incierta, se optó en clase por continuar sin esta parte.

Después, se configuraron las URIs, así como los callbacks de cada recurso.

```
365  static void coap_example_server(void *p)
366  {
367      /* CoAP context and server address */
368      coap_context_t *ctx = NULL;
369      coap_address_t serv_addr;
370
371      /* Define resources */
372      coap_resource_t *resource_direction   = NULL;
373      coap_resource_t *resource_speed       = NULL;
374      coap_resource_t *resource_temperature = NULL;
375      coap_resource_t *resource_current     = NULL;
376      coap_resource_t *resource_gas         = NULL;
```

Adicionalmente, se crearon métodos (get, put, delete) individuales para cada uno de los recursos (URIs). Cabe mencionar, que el script de Python cuenta con una serie de #defines para seleccionar el recurso deseado así como el método a utilizar.

```
# un comment the type of test you want to execute
#TEST = "GET"
TEST = "PUT"
#TEST = "DELETE"

URI = "Espressif/direction"
#URI = "Espressif/speed"
#URI = "Espressif/current"
#URI = "Espressif/temperature"
#URI = "Espressif/gas"
```

Por otro lado, para demostrar la funcionalidad de la práctica, se optó por modificar los callbacks de cada recurso en el ejemplo de CoAP (servidor), así como en el script de Python (cliente) para generar una aplicación con cierta lógica en el flujo de datos.

Además, no en todos los recursos se incluyeron los métodos de GET y PUT ya que no hacía sentido con la lógica de las solicitudes. El método DELETE siempre se incluyó en los recursos. A continuación se explica de manera breve cada recurso implementado.

Recurso: Espressif/direction

El recurso “direction” tiene un método PUT para actualizar la dirección de movimiento del robot en el servidor y un método GET para obtener la información de hacia dónde se está moviendo el robot.

```
/* GET Handlers */
static void
hnd_espressif_get_direction(coap_resource_t *resource,
                  coap_session_t *session,
                  const coap_pdu_t *request,
                  const coap_string_t *query,
                  coap_pdu_t *response)
{
    coap_pdu_set_code(response, COAP_RESPONSE_CODE_CONTENT);
    coap_add_data_large_response(resource, session, request, response,
                                 query, COAP_MEDIATYPE_TEXT_PLAIN, 60, 0,
                                 (size_t)direction_data_len,
                                 (const u_char *)direction_data,
                                 NULL, NULL);

    /* Imprimir datos en terminal */
    printf("Direction: ");

    if (direction_data[0] == 'F')
        printf("Front");

    else if (direction_data[0] == 'B')
        printf("Back");

    else if (direction_data[0] == 'L')
        printf("Left");

    else if (direction_data[0] == 'R')
        printf("Right");

    else
        printf("DIRECTION ERROR");

    printf("\r\n");
}
```

GET

```
/* PUT Handlers */
static void
hnd_espressif_put_direction(coap_resource_t *resource,
                  coap_session_t *session,
                  const coap_pdu_t *request,
                  const coap_string_t *query,
                  coap_pdu_t *response)
{
    size_t size;
    size_t offset;
    size_t total;
    const unsigned char *data;

    coap_resource_notify_observers(resource, NULL);

    if (strcmp (direction_data, INITIAL_DATA_DIRECTION) == 0) {
        coap_pdu_set_code(response, COAP_RESPONSE_CODE_CREATED);
    } else {
        coap_pdu_set_code(response, COAP_RESPONSE_CODE_CHANGED);
    }

    /* coap_get_data_large() sets size to 0 on error */
    (void)coap_get_data_large(request, &size, &data, &offset, &total);

    if (size == 0) {      /* re-init */
        snprintf(direction_data, sizeof(direction_data), INITIAL_DATA_DIRECTION);
        direction_data_len = strlen(direction_data);
    } else {
        espressif_data_len = size > sizeof (direction_data) ? sizeof (direction_data) : size;
        memcpy (direction_data, data, direction_data_len);
    }
}
```

PUT

Las direcciones validas son:

- Front – F
- Back – B
- Right – R
- Left – L

En ambos recursos se implementó un candado en donde si no se reconoce una dirección correcta, se manda un error como se ve en las siguientes imágenes.

```
I (3986) wifi:connected with privada, aid = 3, channel 6, 40D, bssid = 34:21:09:81:3a:e1
I (3986) wifi:security: WPA2-PSK, phy: bgn, rssi: -51
I (3986) wifi:pm start, type: 1

I (4026) wifi:AP's beacon interval = 102400 us, DTIM period = 3
I (4026) wifi:<ba-add>idx:0 (ifx:0, 34:21:09:81:3a:e1), tid:0, ssn:2, winSize:64
I (4986) esp_netif_handlers: example_netif_sta ip: 192.168.39.138, mask: 255.255.255.0, gw: 192
I (4986) example_connect: Got IPv4 event: Interface "example_netif_sta" address: 192.168.39.138
I (5616) example_connect: Got IPv6 event: Interface "example_netif_sta" address: fe80:0000:0000
, type: ESP_IP6_ADDR_IS_LINK_LOCAL
I (5616) example_common: Connected to example_netif_sta
I (5626) example_common: - IPv4 address: 192.168.39.138,
I (5626) example_common: - IPv6 address: fe80:0000:0000:0000:0edc:7eff:fe63:2e98, type: ESP_IP6_
I (14016) wifi:<ba-add>idx:1 (ifx:0, 34:21:09:81:3a:e1), tid:6, ssn:0, winSize:64
Direction: Front
Direction: Front
Direction: Back
Direction: Right
Direction: Left
Direction: DIRECTION ERROR
```

ESP32 - Servidor

```
nxf63522@NXL13161 MINGW64 ~/Desktop/Python
$ python test_coap.py
*** PUT ***
Result: 2.04 Changed
b''
*** GET ***
Result: 2.05 Content
b'R'
Direction: Right


nxf63522@NXL13161 MINGW64 ~/Desktop/Python
$ python test_coap.py
*** PUT ***
Result: 2.04 Changed
b''
*** GET ***
Result: 2.05 Content
b'L'
Direction: Left


nxf63522@NXL13161 MINGW64 ~/Desktop/Python
$ python test_coap.py
*** PUT ***
Result: 2.04 Changed
b''
*** GET ***
Result: 2.05 Content
b'J'
Direction: DIRECTION ERROR
```

Python - Cliente

Recurso: Espressif/speed

El recurso “speed” tiene un método PUT para actualizar la velocidad en términos de duty cycle del robot en el servidor y un método GET para obtener la información de qué tan rápido se está moviendo el robot

```c
static void
hnd_espressif_get_speed(coap_resource_t *resource,
                        coap_session_t *session,
                        const coap_pdu_t *request,
                        const coap_string_t *query,
                        coap_pdu_t *response)
{
    if (speed_data[0] < '0' || speed_data[0] > '9')
    {
        snprintf(speed_data, sizeof(speed_data), "SPEED ERROR");
        speed_data_len = strlen(speed_data);
    }

    coap_pdu_set_code(response, COAP_RESPONSE_CODE_CONTENT);
    coap_add_data_large_response(resource, session, request, response,
                                 query, COAP_MEDIATYPE_TEXT_PLAIN, 60, 0,
                                 (size_t)speed_data_len,
                                 (const u_char *)speed_data,
                                 NULL, NULL);

    /* Imprimir datos en terminal */
    printf("Velocidad - Duty Cycle: ");
    printf("%s", (char *)speed_data);
    printf(" %%");
    printf("\r\n");
}
```

GET

```c
static void
hnd_espressif_put_speed(coap_resource_t *resource,
                        coap_session_t *session,
                        const coap_pdu_t *request,
                        const coap_string_t *query,
                        coap_pdu_t *response)
{
    size_t size;
    size_t offset;
    size_t total;
    const unsigned char *data;

    coap_resource_notify_observers(resource, NULL);

    if (strcmp (speed_data, INITIAL_DATA_SPEED) == 0) {
        coap_pdu_set_code(response, COAP_RESPONSE_CODE_CREATED);
    } else {
        coap_pdu_set_code(response, COAP_RESPONSE_CODE_CHANGED);
    }

    /* coap_get_data_large() sets size to 0 on error */
    (void)coap_get_data_large(request, &size, &data, &offset, &total);

    if (size == 0) {      /* re-init */
        snprintf(speed_data, sizeof(speed_data), INITIAL_DATA_SPEED);
        speed_data_len = strlen(speed_data);
    } else {
        speed_data_len = size > sizeof (speed_data) ? sizeof (speed_data) : size;
        memcpy (speed_data, data, speed_data_len);
    }
}
```

PUT

Las velocidades validas son:

- Entre el 0% y el 100%

En ambos recursos se implementó un candado en donde si no se reconoce una velocidad correcta, se manda un error como se ve en las siguientes imágenes.

```
I (3936) wifi:state: init -> auth (b0)
I (3946) wifi:state: auth -> assoc (0)
I (3966) wifi:state: assoc -> run (10)
I (3986) wifi:connected with privada, aid = 3, channel 6, 40D, bssid = 34:21:09:81:3a:e1
I (3986) wifi:security: WPA2-PSK, phy: bgn, rssi: -52
I (3986) wifi:pm start, type: 1

I (3996) wifi:AP's beacon interval = 102400 us, DTIM period = 3
I (4016) wifi:<ba-add>idx:0 (ifx:0, 34:21:09:81:3a:e1), tid:0, ssn:2, winSize:64
I (5006) esp_netif_handlers: example_netif_sta ip: 192.168.39.138, mask: 255.255.255.0, gw
I (5006) example_connect: Got IPv4 event: Interface "example_netif_sta" address: 192.168.3
I (5616) example_connect: Got IPv6 event: Interface "example_netif_sta" address: fe80:0000
, type: ESP_IP6_ADDR_IS_LINK_LOCAL
I (5616) example_common: Connected to example_netif_sta
I (5626) example_common: - IPv4 address: 192.168.39.138,
I (5626) example_common: - IPv6 address: fe80:0000:0000:0000:0edc:7eff:fe63:2e98, type: ESP
I (5916) wifi:<ba-add>idx:1 (ifx:0, 34:21:09:81:3a:e1), tid:6, ssn:0, winSize:64
Velocidad - Duty Cycle: 50 %
Velocidad - Duty Cycle: 30 %
Velocidad - Duty Cycle: 70 %
Velocidad - Duty Cycle: SPEED ERROR %
```

ESP32 - Servidor

```
nxf63522@NXL13161 MINGW64 ~/Desktop/Python
$ python test_coap.py
*** GET ***
Result: 2.05 Content
b'50'
Velocidad - Duty Cycle: b'50' %

nxf63522@NXL13161 MINGW64 ~/Desktop/Python
$ python test_coap.py
INFO:coap:Retransmission, Message ID: 32887.
*** PUT ***
Result: 2.04 Changed
b''
*** GET ***
Result: 2.05 Content
b'30'
Velocidad - Duty Cycle: b'30' %

nxf63522@NXL13161 MINGW64 ~/Desktop/Python
$ python test_coap.py
*** PUT ***
Result: 2.04 Changed
b''
*** GET ***
Result: 2.05 Content
b'70'
Velocidad - Duty Cycle: b'70' %

nxf63522@NXL13161 MINGW64 ~/Desktop/Python
$ python test_coap.py
*** PUT ***
Result: 2.04 Changed
b''
*** GET ***
Result: 2.05 Content
b'SPEED ERROR'
Velocidad - Duty Cycle: b'SPEED ERROR' %
```

Python - Cliente

Recurso: Espressif /temperature

El recurso “temperature” tiene solamente un método GET para obtener la temperatura del ambiente. No existe el método PUT ya que no es posible modificar esta variable en la aplicación.

```c
static void
hnd_espressif_get_temperature(coap_resource_t *resource,
                              coap_session_t *session,
                              const coap_pdu_t *request,
                              const coap_string_t *query,
                              coap_pdu_t *response)
{
    char temperature_data[100];
    int temperature_data_len = 0;

    snprintf(temperature_data, sizeof(temperature_data), temperatureValues[temperature_indx]);
    temperature_data_len = strlen(temperature_data);

    coap_pdu_set_code(response, COAP_RESPONSE_CODE_CONTENT);
    coap_add_data_large_response(resource, session, request, response,
                                 query, COAP_MEDIATYPE_TEXT_PLAIN, 60, 0,
                                 (size_t)temperature_data_len,
                                 (const u_char *)temperature_data,
                                 NULL, NULL);

    /* Imprimir datos en terminal */
    printf("Temperatura Ambiente: ");
    printf("%s", (char *)temperature_data);
    printf(" °C\r\n");

    /* Actualizar indice */
    temperature_indx++;

    /* Reiniciar indice */
    if (temperature_indx >= 10)
        temperature_indx = 0;
}
```

GET

En este caso, la temperatura ambiente se calcula aleatoriamente de un arreglo predefinido de valores, por lo que cuando se hace una solicitud, se envía un valor predeterminado.

```
I (3986) wifi:security: WPA2-PSK, phy: bgn, rssi: -52
I (3986) wifi:pm start, type: 1

I (4016) wifi:AP's beacon interval = 102400 us, DTIM period = 3
I (4016) wifi:<ba-add>idx:0 (ifx:0, 34:21:09:81:3a:e1), tid:0, ssn:2, winSize:64
I (4986) esp_netif_handlers: example_netif_sta ip: 192.168.39.138, mask: 255.255.255.0,
I (4986) example_connect: Got IPv4 event: Interface "example_netif_sta" address: 192.168
I (5616) example_connect: Got IPv6 event: Interface "example_netif_sta" address: fe80:00
, type: ESP_IP6_ADDR_IS_LINK_LOCAL
I (5616) example_common: Connected to example_netif_sta
I (5626) example_common: - IPv4 address: 192.168.39.138,
I (5626) example_common: - IPv6 address: fe80:0000:0000:0000:0edc:7eff:fe63:2e98, type:
Temperatura Ambiente: 10 °C
Temperatura Ambiente: 52 °C
Temperatura Ambiente: 45 °C
Temperatura Ambiente: 82 °C
```

ESP32 - Servidor

```
nxf63522@NXL13161 MINGW64 ~/Desktop/Python
$ python test_coap.py
*** GET ***
Result: 2.05 Content
b'10'
Temperatura Ambiente: b'10' C

nxf63522@NXL13161 MINGW64 ~/Desktop/Python
$ python test_coap.py
*** GET ***
Result: 2.05 Content
b'52'
Temperatura Ambiente: b'52' C

nxf63522@NXL13161 MINGW64 ~/Desktop/Python
$ python test_coap.py
*** GET ***
Result: 2.05 Content
b'45'
Temperatura Ambiente: b'45' C

nxf63522@NXL13161 MINGW64 ~/Desktop/Python
$ python test_coap.py
*** GET ***
Result: 2.05 Content
b'82'
Temperatura Ambiente: b'82' C
```

Python - Cliente

Recurso: Espressif /current

El recurso “current” tiene solamente un método GET para obtener la corriente que está consumiendo el robot. No existe el método PUT ya que no es posible modificar esta variable en la aplicación.

```
static void
hnd_espressif_get_current(coap_resource_t *resource,
                          coap_session_t *session,
                          const coap_pdu_t *request,
                          const coap_string_t *query,
                          coap_pdu_t *response)
{
    char current_data[100];
    int current_data_len = 0;

    snprintf(current_data, sizeof(current_data), currentValues[current_indx]);
    current_data_len = strlen(current_data);

    coap_pdu_set_code(response, COAP_RESPONSE_CODE_CONTENT);
    coap_add_data_large_response(resource, session, request, response,
                                 query, COAP_MEDIATYPE_TEXT_PLAIN, 60, 0,
                                 (size_t)current_data_len,
                                 (const u_char *)current_data,
                                 NULL, NULL);

    /* Imprimir datos en terminal */
    printf("Consumo de Corriente: ");
    printf("%s", (char *)current_data);
    printf(" mA\r\n");

    /* Actualizar indice */
    current_indx++;

    /* Reiniciar indice */
    if (current_indx >= 10)
        current_indx = 0;
}
```

GET

En este caso, la corriente se calcula aleatoriamente de un arreglo predefinido de valores, por lo que cuando se hace una solicitud, se envía un valor predeterminado.

```
I (4016) wifi:AP's beacon interval = 102400 us, DTIM period = 3
I (4046) wifi:<ba-add>idx:0 (ifx:0, 34:21:09:81:3a:e1), tid:0, ssn:2, winSize:64
I (4986) esp_netif_handlers: example_netif_sta ip: 192.168.39.138, mask: 255.255.255.0, g
I (4986) example_connect: Got IPv4 event: Interface "example_netif_sta" address: 192.168.
I (5616) example_connect: Got IPv6 event: Interface "example_netif_sta" address: fe80:000
, type: ESP_IP6_ADDR_IS_LINK_LOCAL
I (5616) example_common: Connected to example_netif_sta
I (5626) example_common: - IPv4 address: 192.168.39.138,
I (5626) example_common: - IPv6 address: fe80:0000:0000:0000:0edc:7eff:fe63:2e98, type: E
Consumo de Corriente: 10 mA
Consumo de Corriente: 500 mA
Consumo de Corriente: 750 mA
I (19176) wifi:<ba-add>idx:1 (ifx:0, 34:21:09:81:3a:e1), tid:6, ssn:0, winSize:64
Consumo de Corriente: 310 mA
```

ESP32 - Servidor

```
nxf63522@NXL13161 MINGW64 ~/Desktop/Python
$ python test_coap.py
*** GET ***
Result: 2.05 Content
b'10'
Consumo de Corriente: b'10' mA


nxf63522@NXL13161 MINGW64 ~/Desktop/Python
$ python test_coap.py
*** GET ***
Result: 2.05 Content
b'500'
Consumo de Corriente: b'500' mA


nxf63522@NXL13161 MINGW64 ~/Desktop/Python
$ python test_coap.py
*** GET ***
Result: 2.05 Content
b'750'
Consumo de Corriente: b'750' mA


nxf63522@NXL13161 MINGW64 ~/Desktop/Python
$ python test_coap.py
*** GET ***
Result: 2.05 Content
b'310'
Consumo de Corriente: b'310' mA
```

Python - Cliente

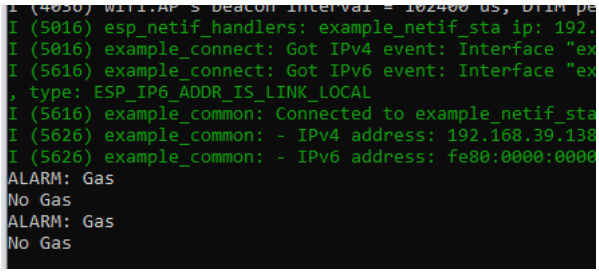
Recurso: Espressif /gas

El recurso “gas” tiene solamente un método GET para obtener la información del sensor de gas. No existe el método PUT ya que no es posible modificar esta variable en la aplicación.

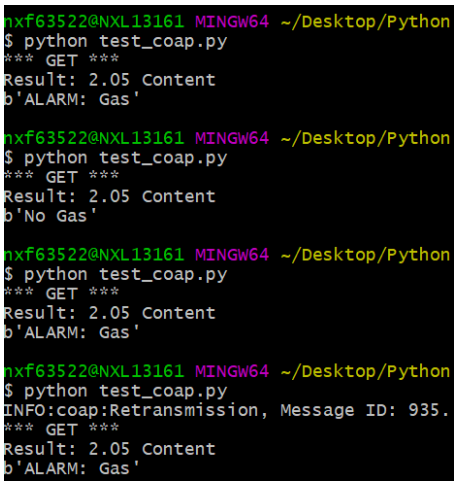
```c
static void
hnd_espressif_get_gas(coap_resource_t *resource,
                      coap_session_t *session,
                      const coap_pdu_t *request,
                      const coap_string_t *query,
                      coap_pdu_t *response)
{
    char gas_data[100];
    int gas_data_len = 0;

    if (gas_status)
        snprintf(gas_data, sizeof(gas_data), "No Gas");

    else
        snprintf(gas_data, sizeof(gas_data), "ALARM: Gas");

    gas_data_len = strlen(gas_data);

    coap_pdu_set_code(response, COAP_RESPONSE_CODE_CONTENT);
    coap_add_data_large_response(resource, session, request, response,
                                 query, COAP_MEDIATYPE_TEXT_PLAIN, 60, 0,
                                 (size_t)gas_data_len,
                                 (const u_char *)gas_data,
                                 NULL, NULL);

    /* Imprimir datos en terminal */
    printf("%s", (char *)gas_data);
    printf("\r\n");

    gas_status = gas_status ^ 1;
}
```

GET

En este caso, para activar la alarma de gas, se optó por una lógica de que cada solicitud va a estar alternando entre si hay o no hay gas en el ambiente. De esta manera se pueden probar ambos escenarios de manera ficticia.



ESP32 - Servidor



Python - Cliente

Conclusión:

Durante la práctica aprendí las bases de CoAP como un protocolo que forma parte de la capa de aplicación del modelo UDP. En resumen, creo que no hubo algo realmente complicado con la práctica ya que gracias a las actividades de clase y lo scripts de Python, estábamos avanzados.

Creo que uno de los problemas que tuve fue el compilar el programa ya que en la computadora del trabajo no podía conectarme a los repositorios de las librerías requeridas. Por lo que, tuve que cambiar mi entorno de trabajo a mi computadora personal. Por otro lado, creo que una la manera de establecer las URIs fue algo sencillo con el código base y sólo le tuve que dedicar tiempo a la lógica de la aplicación

Finalmente, creo que me hubiera gustado alcanzar a incluir mDNS en la práctica pero lamentablemente por el tiempo no pude completarlo. Sin embargo, realicé la actividad de clase para que ese tema quedara más claro de mi parte.

Referencias:

[1] IoT Academy. (2021). Introducción a CoAP. Recuperado de: https://www.gotoiot.com/pages/articles/coap_intro/index.html#iframe-parent

[2] ORACLE. (2022). Introducing the Constrained Application Protocol (CoAP) for Java. Recuperado de: https://blogs.oracle.com/javamagazine/post/java-coap-constrained-application-protocol-introduction