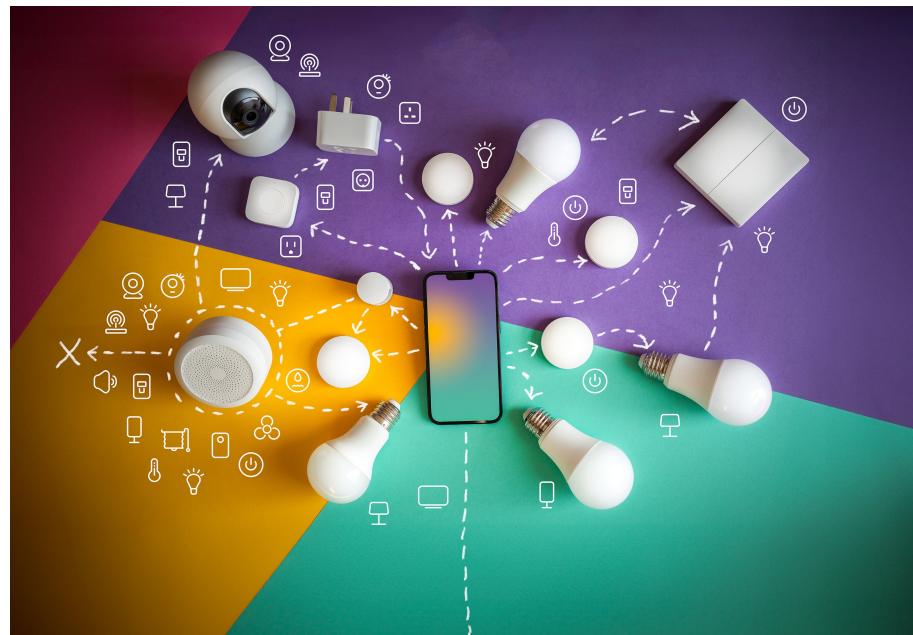


REDES INALÁMBRICAS DE SENSORES



Universidad de Sevilla

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA

Reporte de prácticas

Autor:
Sergio León Doncel

Diciembre 2023

Índice general

1. INTRODUCCIÓN A LA PROGRAMACIÓN EN EL SISTEMA OPERATIVO CONTIKI	1
1.1. Ejercicio 1: Búsqueda de ficheros de la plataforma nRF52840	1
1.2. Ejercicio 2: Hello World	2
1.3. Ejercicio 3: Hello World modificado	3
1.4. Ejercicio 4: Parpadeo de leds	5
1.5. Ejercicio 5: Medida de la temperatura interna	6
2. SIMULACIÓN DE REDES INALÁMBRICAS CONTIKI: MSPSIM Y COOJA	9
2.1. Ejercicio 1. Parpadeo Leds	9
2.2. Ejercicio 2	10
2.3. Ejercicio 3	12
3. REDES EN CONTIKI	17
3.1. Ejercicio 1	17
3.2. Ejercicio 2	21
3.3. Ejercicio 3	23
4. Monitorización de la información	24
4.1. Ejercicio 1	24
4.2. Ejercicio 2	27

Resumen

Este documento consiste en un reporte sobre el procedimiento y resultados conseguidos durante las prácticas de la asignatura Redes Inalámbricas de Sensores para el Máster de Ingeniería Electrónica, Robótica y Automática.

Las prácticas se orientan en torno al sistema operativo open-source de Contiki, enfocado hacia dispositivos de recursos limitados para el IoT. Esto permite una programación a alto nivel para centrarse en el desarrollo de la capa de aplicación.

Capítulo 1

INTRODUCCIÓN A LA PROGRAMACIÓN EN EL SISTEMA OPERATIVO CONTIKI

1.1. Ejercicio 1: Búsqueda de ficheros de la plataforma nRF52840

Los distintos drivers de los que Contiki dispone están alojados dentro del directorio arch/cpu/nrf52840, el cuál es una copia del Software Development Kit que el fabricante facilita para su plataforma hardware.

Las rutas de los drivers para los siguientes chips son:

- Microcontrolador
 - Archivos de cabecera:
`arch/cpu/nrf52840/lib/nrf52-sdk/...`
`.../modules/nrfx/drivers/include/**`
 - Archivos fuente:
`arch/cpu/nrf52840/lib/nrf52-sdk/...`
`.../modules/nrfx/drivers/src/**`
- Transceiver radio
 - Archivos de cabecera:
`arch/cpu/nrf52840/lib/nrf52-sdk/...`
`.../modules/nrfx/hal/nrf_radio.h`
 - Archivos fuente:
`arch/cpu/nrf52840/rf/nrf52840-ieee.c`

- #### ■ Almacenamiento externo

- Archivos de cabecera:

arch/cpu/nrf52840/lib/nrf52-sdk/
 .../components/libraries/fstorage/...
 .../nrf_fstorage_sd.h

- Archivos fuente:

```
arch/cpu/nrf52840/lib/nrf52-sdk/...
    .../components/libraries/fstorage/...
        .../nrf_fstorage_sd.c
```

- #### ■ Almacenamiento interno

- Archivos de cabecera:

arch/cpu/nrf52840/lib/nrf52-sdk/...
 .../components/libraries/fstorage/...
 .../nrf_fstorage_nvmc.h

- Archivos fuente:

```
arch/cpu/nrf52840/lib/nrf52-sdk/...
    .../components/libraries/fstorage/...
        .../nrf_fstorage_nvmc.c
```

1.2. Ejercicio 2: Hello World

A continuación, se adjuntan capturas demostrativas de los pasos seguidos para reproducir el test inicial propuesto.

Figura 1.1: Compilación

Figura 1.2: Carga del programa

Figura 1.3: Ejecución

1.3. Ejercicio 3: Hello World modificado

Partiendo del programa `hello_world` de ejemplo, se añade un nuevo proceso `periodic_process`, el cual contará con un temporizador configurado para expirar a los 5 segundos.

En un bucle indefinido, el proceso entrará en espera hasta ser reactivado para la recepción del evento generado por su propio timer al expirar, momento en el cuál resetará el contador del timer y utilizará un mecanismo de **POST** síncrono que enviará directamente un evento **PROCESS_EVENT_AWAKE**. Este identificador

de evento no está definido por el kernel de Contiki, pero tal y como indica su documentación [2] , los valores por debajo de 128 quedan libres para uso de los procesos de usuario, lo cual se ha aprovechado para utilizar uno propio en este caso.

Por tanto, *hello_world_process* será despertado para mostrar el mensaje, actualizar el contador y volverse a quedar en espera. Al haber generado un evento síncrono, lo que ocurrirá, de manera invisible para el usuario, es que el planificador programará de manera inmediata el proceso receptor del evento dándole absoluta prioridad, y por lo tanto deteniendo la ejecución del proceso originario. De esta manera, una vez el proceso receptor vuelva a la espera, el proceso secundario retomará su ejecución por donde estaba.[1]

```

1 /**
2  * \file
3  *      Ejercicio 3: Hello World modificado
4  * \author
5  *      Sergio Leon <schoolion01@gmail.com>
6 */
7
8 #include "contiki.h"
9
10 #include <stdint.h>
11 #include <stdio.h>
12
13 #define PROCESS_EVENT_AWAKE 0 // Custom user defined event identifier
14 /*-----*/
15 PROCESS(hello_world_process, "Hello world process");
16 PROCESS(periodic_process, "Periodic process");
17 AUTOSTART_PROCESSES(&hello_world_process, &periodic_process);
18 /*-----*/
19 PROCESS_THREAD(hello_world_process, ev, data) {
20     static uint8_t event_counter = 0;
21
22     PROCESS_BEGIN();
23
24     while (1) {
25         /* Wait to receive an event from another process */
26         PROCESS_WAIT_EVENT_UNTIL(ev == PROCESS_EVENT_AWAKE);
27
28         printf("Hello World! (number %d)\n", event_counter);
29
30         if (event_counter < 20) {
31             event_counter++;
32         } else {
33             event_counter = 0;
34         }
35     }
36
37     PROCESS_END();
38 }
39
40 PROCESS_THREAD(periodic_process, ev, data) {
41     static struct etimer timer;
42
43     PROCESS_BEGIN();
44
45     /* Setup a periodic timer that expires after 5 seconds. */
46     etimer_set(&timer, CLOCK_SECOND * 5);
47
48     while (1) {
49         /* Wait for the periodic timer to expire and then restart the timer. */
50         PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&timer));
51         etimer_reset(&timer);
52         process_post_synch(&hello_world_process, PROCESS_EVENT_AWAKE, NULL);

```

```

53     }
54
55     PROCESS_END();
56 }
57 /*-----*/

```

hello_world_modificado.c

1.4. Ejercicio 4: Parpadeo de leds

Este programa se encuentra dividido en 3 procesos: *parpadeo_1_process*, *parpadeo_2_process* y *timer_process*. El proceso con temporizador tendrá como misión única despertar el resto de los procesos al principio de la ejecución, tras 5 segundos, enviando, mediante un mecanismo **POLL**, eventos a los procesos de parpadeos, y muriendo tras ello. A diferencia del programa anterior, al haber utilizado un *process_poll()*, aunque los procesos objetivos de los eventos serán planificados con alta prioridad por el kernel de Contiki, el proceso generador del evento podrá terminar su ejecución antes, ya que el evento es asíncrono.

Por su lado, los procesos de parpadeo comutarán el estado de los leds de la placa cada 2 y 3 segundos respectivamente, gracias a sus propios temporizadores que se encargarán de despertarlos.

```

1 /**
2 * \file
3 *          Ejercicio 4: Parpadeo LEDs
4 * \author
5 *          Sergio Leon <schoolion01@gmail.com>
6 */
7
8 #include "contiki.h"
9 #include "dev/leds.h"
10
11 /*-----*/
12 PROCESS(parpadeo_1_process, "Blinking led 1 process");
13 PROCESS(parpadeo_2_process, "Blinking led 2 process");
14 PROCESS(timer_process, "Periodic process");
15 AUTOSTART_PROCESSES(&parpadeo_1_process, &parpadeo_2_process, &timer_process);
16 /*-----*/
17 PROCESS_THREAD(parpadeo_1_process, ev, data) {
18     static struct etimer timer;
19
20     PROCESS_BEGIN();
21
22     /* Wait to receive an event from timer_process in order to continue */
23     PROCESS_WAIT_EVENT_UNTIL(ev == PROCESS_EVENT_POLL);
24
25     /* Setup a periodic timer that expires after 2 seconds. */
26     etimer_set(&timer, CLOCK_SECOND * 2);
27
28     while (1) {
29         leds_single_toggle(LED1);
30         etimer_reset(&timer);
31
32         /* Wait for the periodic timer to expire and then restart the timer. */
33         PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&timer));
34     }

```

```

35     PROCESS_END();
36 }
37
38
39 PROCESS_THREAD(parpadeo_2_process, ev, data) {
40     static struct etimer timer;
41
42     PROCESS_BEGIN();
43
44     /* Wait to receive an event from timer_process in order to continue */
45     PROCESS_WAIT_EVENT_UNTIL(ev == PROCESS_EVENT_POLL);
46
47     /* Setup a periodic timer that expires after 3 seconds. */
48     etimer_set(&timer, CLOCK_SECOND * 3);
49
50     while (1) {
51         leds_single_toggle(LEDS_LED2);
52
53         /* Wait for the periodic timer to expire and then restart the timer. */
54         PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&timer));
55         etimer_reset(&timer);
56     }
57
58     PROCESS_END();
59 }
60
61 PROCESS_THREAD(timer_process, ev, data) {
62     static struct etimer timer;
63
64     PROCESS_BEGIN();
65
66     /* Setup a timer that expires after 5 seconds. */
67     etimer_set(&timer, CLOCK_SECOND * 5);
68
69     PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&timer));
70
71     process_poll(&parpadeo_1_process);
72     process_poll(&parpadeo_2_process);
73
74     PROCESS_END();
75 }
76 /*-----*/

```

blink.c

1.5. Ejercicio 5: Medida de la temperatura interna

Para la lectura del sensor de temperatura interna del dongle, se diseñara un programa de 2 procesos:

1. *timer_process*: A través de un temporizador ajustado con tiempo de expiración de 3 segundos, mediante un mecanismo de post síncrono, se encargará de enviar un evento **PROCESS_EVENT_CONTINUE** al proceso de *sensor_reading* para despertarlo.
2. *sensor_reading*: Realizará la lectura del registro del sensor de temperatura y, gracias al conocimiento de la resolución del sensor, calculará la temperatura medida con la siguiente expresión:

$$temperature_value[C] = register_value[bit] * 0,25[C/bit]$$

Dada la importancia en dispositivos IoT del ahorro de energía debido a su alimentación por pila, el sensor se activará y desactivará cada vez que se quiera realizar una medición, disminuyendo así el consumo medio.

Además, cabe destacar que a pesar de que el microcontrolador nRF52840 dispone de una FPU (floating point unit) para cálculos aritméticos con flotantes, esto tiene un consumo de recursos innecesarios para esta ocasión. Por tanto, se realiza el cálculo de manera equivalente. Para el cálculo de los 2 decimales de temperatura, se aplica una máscara binaria para obtener los 2 últimos bits del registro del sensor, mientras que para la parte entera, se lleva a cabo un desplazamiento de 2 bits a la derecha para eliminar la mantisa.

Por último, se ha tenido en cuenta que la función printf de la librería de contiki no permite formatear este tipo de dato. Por ello, se muestra parte entera y decimal por separado, en lugar de almacenar la información en una variable de tipo *float*.

```
1 /**
2 * \file
3 *      Ejercicio 5: Medida de la temperatura interna
4 * \author
5 *      Sergio Leon <schoolion01@gmail.com>
6 */
7
8 #include "contiki.h"
9 #include "lib/sensors.h"
10 #include "temperature-sensor.h"
11
12 #include <stdint.h>
13 #include <stdio.h>
14
15 #define PROCESS_EVENT_AWAKE 0 // Custom user defined event identifier
16 /*-----*/
17 PROCESS(sensor_reading, "Sensor reading process");
18 PROCESS(timer_process, "Timer process");
19 AUTOSTART_PROCESSES(&sensor_reading, &timer_process);
20 /*-----*/
21 PROCESS_THREAD(sensor_reading, ev, data) {
22     uint16_t sensor_register_value;
23     struct temperature_t {
24         uint8_t temperatureInt;
25         uint8_t temperatureDec;
26     };
27     struct temperature_t temperature;
28
29     PROCESS_BEGIN();
30
31     while (1) {
32         /* Wait to receive an event in order to read again */
33         PROCESS_WAIT_EVENT_UNTIL(ev == PROCESS_EVENT_AWAKE);
34
35         /* Read temperature value from sensor*/
36         SENSORS_ACTIVATE(temperature_sensor);
37         sensor_register_value = temperature_sensor.value(0);
38         SENSORS_DEACTIVATE(temperature_sensor);
39
40         /* Print read value */
41         temperature.temperatureInt = (sensor_register_value >> 2);
42         temperature.temperatureDec = ((sensor_register_value & 0x3) * 25U);
43         printf("Temperature: %.2d%.2d C \n", temperature.temperatureInt,
```

```

44         temperature.temperatureDec);
45     }
46
47     PROCESS_END();
48 }
49
50 PROCESS_THREAD(timer_process, ev, data) {
51     static struct etimer timer;
52
53     PROCESS_BEGIN();
54
55     /* Setup a periodic timer that expires after 3 seconds. */
56     etimer_set(&timer, CLOCK_SECOND * 3);
57
58     while (1) {
59         /* Wait for the periodic timer to expire and then restart the timer. */
60         PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&timer));
61         etimer_reset(&timer);
62
63         process_post_synch(&sensor_reading, PROCESS_EVENT_AWAKE, NULL);
64     }
65
66     PROCESS_END();
67 }
68 /*-----*/

```

temperature.c

Capítulo 2

SIMULACIÓN DE REDES INALÁMBRICAS CONTIKI: MSPSIM Y COOJA

2.1. Ejercicio 1. Parpadeo Leds

Partiendo como base del ejercicio 3 de la práctica 1 mostrado en el capítulo anterior, se intercambia la llamada a la función *leds_single_toggle* por *leds_toggle*, y se ajusta el tiempo de expiración de los temporizadores.

```
1  /**
2   * \file
3   *      Ejercicio 1: Parpadeo LEDs
4   * \author
5   *      Sergio Leon <schoolion01@gmail.com>
6   */
7
8 #include "contiki.h"
9 #include "dev/leds.h"
10
11 /*****
12 PROCESS(parpadeo_1_process, "Blinking led 1 process");
13 PROCESS(parpadeo_2_process, "Blinking led 2 process");
14 PROCESS(timer_process, "Periodic process");
15 AUTOSTART_PROCESSES(&parpadeo_1_process, &parpadeo_2_process, &timer_process);
16 *****/
17 PROCESS_THREAD(parpadeo_1_process, ev, data) {
18     static struct etimer timer;
19
20     PROCESS_BEGIN();
21
22     /* Wait to receive an event from timer_process in order to continue */
23     PROCESS_WAIT_EVENT_UNTIL(PROCESS_EVENT_POLL);
24
25     /* Setup a periodic timer that expires after 2 seconds. */
26     etimer_set(&timer, CLOCK_SECOND * 2);
27
28     while (1) {
29         leds_toggle(LED_GREEN);
30
31         /* Wait for the periodic timer to expire and then restart the timer. */
32         PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&timer));
33         etimer_reset(&timer);
34     }
35 }
```

```

36     PROCESS_END();
37 }
38
39 PROCESS_THREAD(parpadeo_2_process, ev, data) {
40     static struct etimer timer;
41
42     PROCESS_BEGIN();
43
44     /* Wait to receive an event from timer_process in order to continue */
45     PROCESS_WAIT_EVENT_UNTIL(PROCESS_EVENT_POLL);
46
47     /* Setup a periodic timer that expires after 4 seconds. */
48     etimer_set(&timer, CLOCK_SECOND * 4);
49
50     while (1) {
51         leds_toggle(LEDS_YELLOW);
52
53         /* Wait for the periodic timer to expire and then restart the timer. */
54         PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&timer));
55         etimer_reset(&timer);
56     }
57
58     PROCESS_END();
59 }
60
61 PROCESS_THREAD(timer_process, ev, data) {
62     static struct etimer timer;
63
64     PROCESS_BEGIN();
65
66     /* Setup a timer that expires after 3 seconds. */
67     etimer_set(&timer, CLOCK_SECOND * 3);
68
69     PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&timer));
70
71     process_poll(&parpadeo_1_process);
72     process_poll(&parpadeo_2_process);
73
74     PROCESS_END();
75 }
76 /*-----*/

```

blink.c

2.2. Ejercicio 2

Al iniciar la simulación, tras esperar un cierto tiempo de establecimiento de la red, todos los nodos terminales Z1 establecen conexión con el nodo pasarela Sky, como muestra la figura 2.1.

En ella se puede ver en los mensajes emitidos por los nodos terminales, indicando que han enviado una petición a la dirección IPv6 fd00::212:7401:1:101 del nodo sumidero Z1.

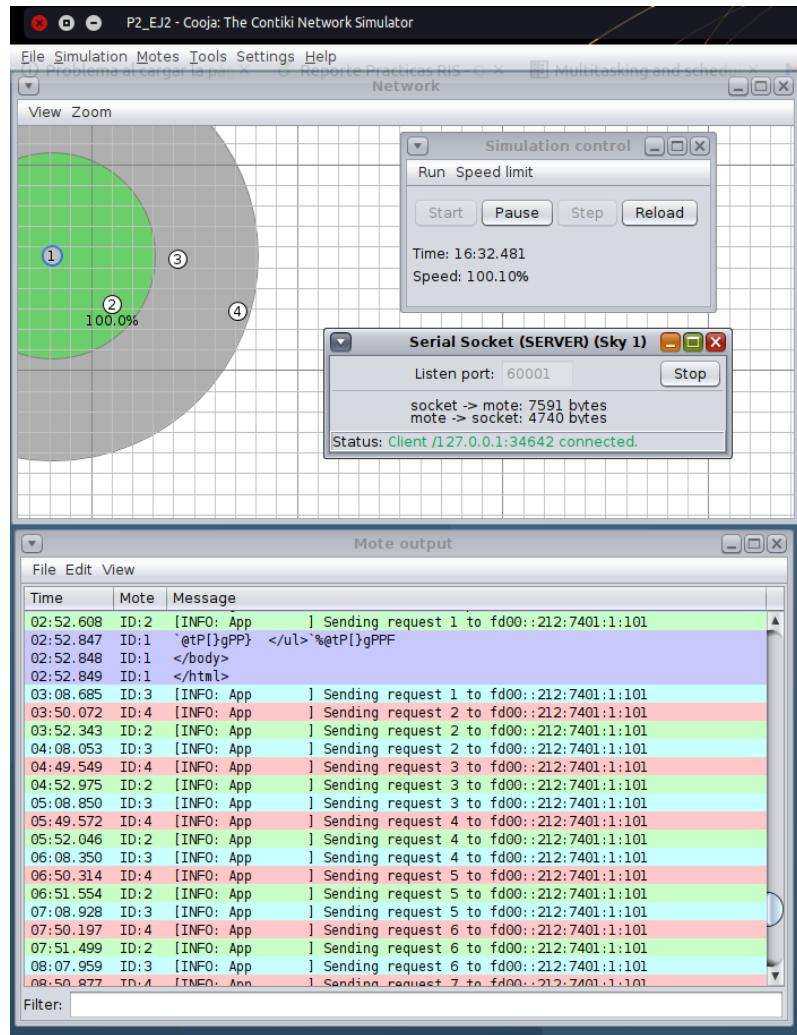


Figura 2.1: Resultado de simulación tras conexión

Tras abrir el socket serie, se puede acceder al servidor web alojado por el nodo border-router Z1. Desde el navegador web se introduce su dirección y se observa una lista de los nodos vecinos.



Figura 2.2: Web del nodo frontera

El primer bloque muestra aquellos nodos con los que existe conexión directa. En este caso, aparece la dirección del nodo 2 ya que es el único al que tiene alcance sin intermediarios.

El segundo bloque muestra todos los nodos de la red a los que tiene alcance. Por ello, en la configuración montada, aparecen las direcciones de los nodos terminales 2, 3 y 4.

Analizando los archivos fuente tanto de cliente como servidor, se pueden concluir los siguientes puntos:

- Servidor: En el fichero principal *border-router.c* se crea un único proceso cuya finalidad es la de instanciar un nuevo proceso de servidor web cuando la macro **BORDER_ROUTER_CONF_WEBSERVER** se encuentre definida a 1 en *module-macros.h*, donde también se configura el máximo de número de vecinos en la tabla, así como el máximo de entradas de ruteo, con el fin de ahorrar memoria, dada la limitada capacidad del nodo para alojar el servidor web. Por otro lado, los archivos *webserver.c*, junto a *httpd-simple.c* y *httpd-simple.h*, propios del sistema operativo de contiki, definen el proceso anteriormente mencionado, y las funciones de generación de rutas y obtención de direcciones ipv6 de cada nodo.
- Cliente: Programado en el fichero fuente *udp-client.c*, cuenta con un único proceso el cual obtiene direcciones ip del nodo padre que tiene a su alcance y envía como payload mediante protocolo UDP una cadena del formato "hello x", siendo x un contador que simboliza el número de peticiones o envíos realizados al servidor. Este código también asocia un callback en caso de respuesta del servidor, pero para este ejemplo esto no tiene relevancia ya que esto no se produce.

Por tanto, la información que los nodos terminales envían hacia el router es un payload que contiene un mensaje en formato de cadena saludando junto al número de veces que se ha enviado el mensaje al servidor por parte de dicho nodo. Dado que el intervalo entre envíos se ha definido como 60 segundos, puede entenderse como un cronómetro en minutos, asumiendo cierto error aleatorio por fluctuaciones en las comunicaciones de la red.

2.3. Ejercicio 3

Se modifica el código del cliente UDP, de manera que envíen alternativamente información sobre la temperatura interna y externa, medida por los sensores del microcontrolador del nodo. Dado que se trata de una simulación, se definen estos valores como constantes dentro del código (**INTERNAL_TEMPERATURE** y **EXTERNAL_TEMPERATURE** respectivamente).

Dado que se pretende enviar la temperatura externa en las veces impares, e interna en las pares, se declara la variable *informationSelector* como un enumerado

con las posibilidades *internal_temperature* y *external_temperature*, e inicializada a este último valor. De esa manera, en cada iteración del bucle infinito que gobierna el funcionamiento del nodo, se alterna su valor al contrario, para utilizarlo a la hora de elegir qué información enviar.

El código cuenta con un único proceso *udp_client_process* el cual registrará inicialmente una conexión UDP con la llamada a la función *simple_udp_register*, y seteará un timer con un periodo de 60 segundos (aproximadamente, ya que se aplica un error aleatorio que simula la posible fluctuación que cabría esperar en una red real).

Por tanto, el proceso será despertado cada minuto, e informará por puerto serie mediante un mensaje qué tipo de información está enviando, así como dicho valor, y lo enviará como payload usando la conexión UDP, gracias a la llamada a la función *simple_udp_sendto*, siempre y cuándo el nodo padre sea alcanzable.

Por último, cabe mencionar que el callback asociado, *udp_rx_callback*, recibirá un mensaje por parte del servidor, con el resultado del cálculo de conversión de grados Celsius a Fahrenheit, y lo mostrará al usuario a través del puerto serie como una cadena.

```

1 #include "contiki.h"
2 #include "net/ipv6/simple-udp.h"
3 #include "net/netstack.h"
4 #include "net/routing/routing.h"
5 #include "random.h"
6 #include "sys/log.h"
7
8 #include <stdint.h>
9
10 #define LOG_MODULE "Client"
11 #define LOG_LEVEL LOG_LEVEL_INFO
12
13 #define WITH_SERVER_REPLY 1
14 #define UDP_CLIENT_PORT 8765
15 #define UDP_SERVER_PORT 5678
16
17 #define SEND_INTERVAL (60 * CLOCK_SECOND)
18
19 static struct simple_udp_connection udp_conn;
20
21 /*-----*/
22 PROCESS(udp_client_process, "UDP client");
23 AUTOSTART_PROCESSES(&udp_client_process);
24 /*-----*/
25 static void udp_rx_callback(struct simple_udp_connection *c,
26                             const uip_ipaddr_t *sender_addr,
27                             uint16_t sender_port,
28                             const uip_ipaddr_t *receiver_addr,
29                             uint16_t receiver_port, const uint8_t *data,
30                             uint16_t datalen) {
31
32     LOG_INFO("Temperatura en Fahrenheit calculada por el servidor= '%d' grados.", *data);
33     #if LLSEC802154_CONF_ENABLED
34     LOG_INFO_( " LLSEC LV:%d", uipbuf_get_attr(UIPBUF_ATTR_LLSEC_LEVEL));
35     #endif
36     LOG_INFO_( "\n");
37 }
38 /*-----*/
39 PROCESS_THREAD(udp_client_process, ev, data) {

```

```

41 static struct etimer periodic_timer;
42 uip_ipaddr_t dest_ipaddr;
43
44 typedef uint8_t CelsiusDegrees_t;
45 const CelsiusDegrees_t INTERNAL_TEMPERATURE = 38U;
46 const CelsiusDegrees_t EXTERNAL_TEMPERATURE = 35U;
47 enum informationType { internal_temperature, external_temperature };
48 static enum informationType informationSelector = external_temperature;
49 CelsiusDegrees_t temperatureToSend;
50
51 PROCESS_BEGIN();
52
53 /* Initialize UDP connection */
54 simple_udp_register(&udp_conn, UDP_CLIENT_PORT, NULL, UDP_SERVER_PORT,
55                     udp_rx_callback);
56
57 etimer_set(&periodic_timer, random_rand() % SEND_INTERVAL);
58 while (1) {
59     PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&periodic_timer));
60
61     if (NETSTACK_ROUTING.node_is_reachable() &&
62         NETSTACK_ROUTING.get_root_ipaddr(&dest_ipaddr)) {
63         /* Send to DAG root */
64         if (informationSelector == external_temperature) {
65             LOG_INFO("Enviando temperatura externa en Celsius = %d\n",
66                     EXTERNAL_TEMPERATURE);
67             temperatureToSend = EXTERNAL_TEMPERATURE;
68             informationSelector = internal_temperature;
69         } else {
70             LOG_INFO("Enviando temperatura interna en Celsius = %d\n",
71                     INTERNAL_TEMPERATURE);
72             temperatureToSend = INTERNAL_TEMPERATURE;
73             informationSelector = external_temperature;
74         }
75         simple_udp_sendto(&udp_conn, &temperatureToSend,
76                           sizeof(temperatureToSend), &dest_ipaddr);
77     } else {
78         LOG_INFO("Not reachable yet\n");
79     }
80
81     /* Add some jitter */
82     etimer_set(&periodic_timer, SEND_INTERVAL - CLOCK_SECOND +
83                (random_rand() % (2 * CLOCK_SECOND)));
84 }
85
86 PROCESS_END();
87 }
88 /*-----*/

```

udp-client.c

Por parte del servidor, el código tiene un funcionamiento similar. Existe un proceso en el cual se define el nodo como raíz, y se registra una conexión UDP con callback de recepción asociado. En esta función, se recibe el payload que contiene la información de temperatura enviada por el nodo terminal y se procesa un cálculo de conversión a Fahrenheit según la expresión:

$$F = 2 * C + 32$$

Por último, se envía el resultado del cálculo al nodo terminal utilizando la dirección IP del nodo que envió el mensaje.

```

1 /*
2  * Redistribution and use in source and binary forms, with or without
3  * modification, are permitted provided that the following conditions
4  * are met:
5  * 1. Redistributions of source code must retain the above copyright
6  *    notice, this list of conditions and the following disclaimer.
7  * 2. Redistributions in binary form must reproduce the above copyright
8  *    notice, this list of conditions and the following disclaimer in the
9  *    documentation and/or other materials provided with the distribution.
10 * 3. Neither the name of the Institute nor the names of its contributors
11 *    may be used to endorse or promote products derived from this software
12 *    without specific prior written permission.
13 */
14
15 * THIS SOFTWARE IS PROVIDED BY THE INSTITUTE AND CONTRIBUTORS ``AS IS'' AND
16 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
17 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
18 * ARE DISCLAIMED. IN NO EVENT SHALL THE INSTITUTE OR CONTRIBUTORS BE LIABLE
19 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
20 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
21 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
22 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
23 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
24 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
25 * SUCH DAMAGE.
26 */
27
28 */
29
30 #include "contiki.h"
31 #include "net/ipv6/simple-udp.h"
32 #include "net/netstack.h"
33 #include "net/routing/routing.h"
34 #include "sys/log.h"
35
36 #include <stdint.h>
37
38 #define LOG_MODULE "Server"
39 #define LOG_LEVEL LOG_LEVEL_INFO
40
41 #define WITH_SERVER_REPLY 1
42 #define UDP_CLIENT_PORT 8765
43 #define UDP_SERVER_PORT 5678
44
45 static struct simple_udp_connection udp_conn;
46
47 PROCESS(udp_server_process, "UDP server");
48 AUTOSTART_PROCESSES(&udp_server_process);
49 /*-----*/
50 static void udp_rx_callback(struct simple_udp_connection *c,
51                             const uip_ipaddr_t *sender_addr,
52                             uint16_t sender_port,
53                             const uip_ipaddr_t *receiver_addr,
54                             uint16_t receiver_port, const uint8_t *data,
55                             uint16_t datalen) {
56     typedef uint8_t CelsiusDegrees_t;
57     typedef uint8_t FahrenheitDegrees_t;
58     CelsiusDegrees_t receivedTemperature;
59     FahrenheitDegrees_t processedTemperature;
60
61     receivedTemperature = *data;
62     LOG_INFO("Info recibida del nodo: '%d' grados\n", receivedTemperature);
63     LOG_INFO("Direccion = ");
64     LOG_INFO_6ADDR(sender_addr);
65     LOG_INFO_("\\n");
66     LOG_INFO_("TEMPERATURA RECIBIDA = %d\n", receivedTemperature);
67
68     processedTemperature = (2U * receivedTemperature + 32U);
69
70 #if WITH_SERVER_REPLY

```

```

71  /* send back the same string to the client as an echo reply */
72  LOG_INFO("Enviando temperatura en Fahrenheit = %d\n", processedTemperature);
73  LOG_INFO("Destino = ");
74  LOG_INFO_6ADDR(sender_addr);
75  LOG_INFO_( "\n");
76  simple_udp_sendto(&udp_conn, &processedTemperature,
77                      sizeof(processedTemperature), sender_addr);
78 #endif /* WITH_SERVER_REPLY */
79 }
80 */
81 PROCESS_THREAD(udp_server_process, ev, data) {
82     PROCESS_BEGIN();
83
84     /* Initialize DAG root */
85     NETSTACK_ROUTING.root_start();
86
87     /* Initialize UDP connection */
88     simple_udp_register(&udp_conn, UDP_SERVER_PORT, NULL, UDP_CLIENT_PORT,
89                         udp_rx_callback);
90
91     PROCESS_END();
92 }
93 */

```

udp-server.c

Se adjunta junto a esta memoria los códigos .c del servidor y cliente, así como el fichero .csc resultante de la simulación.

Tras la compilación, carga de los programas, y simulación en Cooja, se obtiene como resultado:

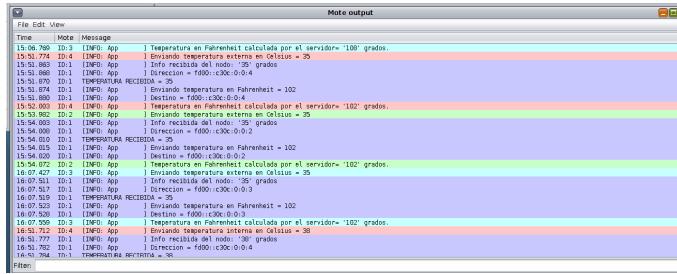


Figura 2.3: Resultado de simulación de la red entre nodo servidor y cliente

De este resultado, se puede ver el correcto funcionamiento de la red. Si nos fijamos en el ID del nodo 4, podemos ver como existe un primer mensaje de envío de temperatura externa de 35°C, seguido de un mensaje del servidor confirmando la recepción de la información y el cálculo del valor 102 equivalente en Fahrenheit. Por último, el callback de recepción del nodo terminal recibe el cálculo del servidor y muestra el correspondiente mensaje.

Si se revisa para cada uno de los nodos terminales, se puede concluir que en todos ellos se sigue este mismo patrón, cumpliendo con las especificaciones del enunciado.

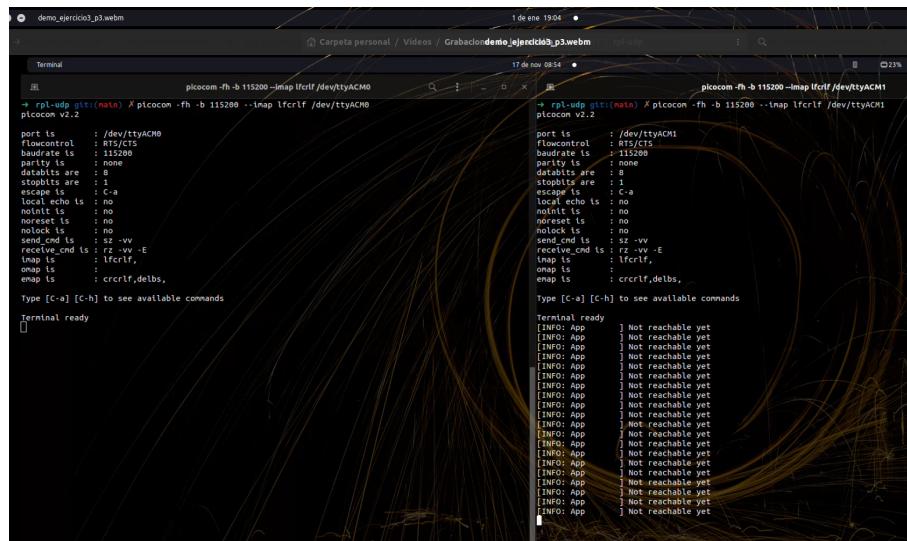
Capítulo 3

REDES EN CONTIKI

3.1. Ejercicio 1

Se procede al despliegue de la red del ejercicio 3 de la práctica anterior, con nodos reales, utilizando 2 dongles nRF52840 de Nordic. Para que ambos dispositivos puedan comunicarse bajo la misma red de área personal (PAN), deben tener configurados el mismo PANID así como canal de transmisión en los programas flasheados. El valor de estos puede encontrarse en el código dentro de la cabecera de configuración de Contiki `/contiki-ng/os/contiki-default-conf.h`, bajo las siguientes macros: **IEEE802154_CONF_PANID** y **IEEE802154_CONF_DEFAULT_CHANNEL**, con valores por defecto de **0xabcd** y **26** respectivamente.

A continuación se muestran capturas de pantalla que muestran los mensajes obtenidos por puerto serie en ambos nodos al establecer la comunicación (a la izquierda nodo servidor y a la derecha nodo cliente):



The screenshot shows two terminal windows side-by-side. Both windows are titled "Terminal ready". The left window is labeled "demo_ejercicio1_p1.webm" and the right window is labeled "demo_ejercicio1_p2.webm". Both windows show the same command being run: "picocom -fh -b 115200 -imap lfcrlf /dev/ttyACM0". Below this command, the configuration for the serial port is displayed:

```
port is : /dev/ttyACM0
flowcontrol : RTS/CTS
baudrate is : 115200
parity is : none
databits are : 8
stopbits are : 1
escape is : C-a
oversize is : no
noinit is : no
noreset is : no
noctrl is : no
send cmd is : sz -vv
receive cmd is : rz -vv -E
tmap is : lfcrlf,
imap is :
enap is : crrcrlf,dels,
```

At the bottom of each window, it says "Type [C-a] [C-h] to see available commands".

Figura 3.1: Mensaje por puerto serie de la red desplegada con nodos reales, sin establecimiento de conexión

Existe un primer periodo de tiempo donde la conexión aún no se ha establecido

y se reintenta hasta conseguirse, como muestra la figura 3.1.

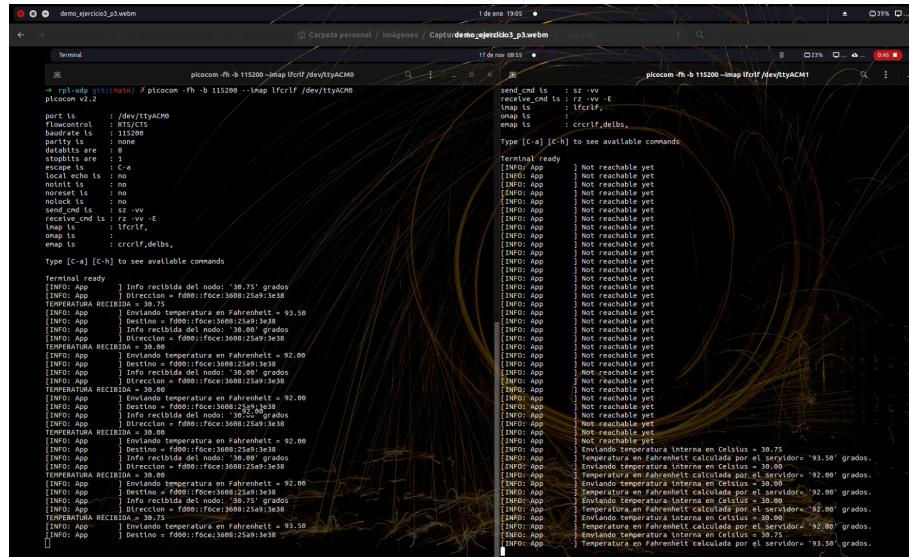


Figura 3.2: Mensaje por puerto serie de la red desplegada con nodos reales, comparando información entre sí

No obstante, tras varios intentos, la conexión se establece y la red comienza a funcionar según lo previsto, permitiendo a los nodos el intercambio de información mediante envío de mensajes en el proceso principal, y recepción a través de los callback definidos, e imprimiendo los logs programados por puerto serie para dar información al usuario.

Cabe destacar que para este ejercicio, se ha añadido una pequeña modificación respecto al código de la solución del ejercicio 3 de la práctica anterior. Previamente, dado que los valores de temperatura en simulación eran constantes y se evitaba, explícitamente como indicaba el enunciado, los valores decimales, el payload enviado y recibido codificaba números enteros. Sin embargo, para el caso real, esto sería erróneo.

Tras la modificación, el payload enviado incluye parte entera y mantisa en los bytes transmitidos, gracias a la interpretación de tipo float en los punteros utilizados. Esto, sin embargo, no evita que los nodos tengan que obtener de aquí parte entera y decimal por separado para la representación por puerto serie de los mensajes.

A continuación el código actualizado:

```

1 #include "contiki.h"
2 #include "net/ipv6/simple-udp.h"
3 #include "net/netstack.h"
4 #include "net/routing/routing.h"
5 #include "random.h"
6 #include "sys/log.h"
7 #include "temperature-sensor.h"
8
9 #include <stdint.h>
10
11 #define LOG_MODULE "Client"
12 #define LOG_LEVEL LOG_LEVEL_INFO

```

```

13 #define WITH_SERVER_REPLY 1
14 #define UDP_CLIENT_PORT 8765
15 #define UDP_SERVER_PORT 5678
16
17 #define SEND_INTERVAL (1 * CLOCK_SECOND)
18
19 static struct simple_udp_connection udp_conn;
20
21 /*-----*/
22 PROCESS(udp_client_process, "UDP client");
23 AUTOSTART_PROCESSES(&udp_client_process);
24 /*-----*/
25
26 static void udp_rx_callback(struct simple_udp_connection *c,
27                             const uip_ipaddr_t *sender_addr,
28                             uint16_t sender_port,
29                             const uip_ipaddr_t *receiver_addr,
30                             uint16_t receiver_port, const uint8_t *data,
31                             uint16_t datalen) {
32     typedef float FahrenheitDegrees_t;
33     uint8_t temperatureInt;
34     uint8_t temperatureDec;
35
36     FahrenheitDegrees_t receivedTemperature = *((float *)data);
37     temperatureInt = (uint8_t)receivedTemperature;
38     uint8_t temperatureDec =
39         ((receivedTemperature - (float)temperatureInt) * 100U);
40     LOG_INFO("Temperatura en Fahrenheit calculada por el servidor= '%.2d.%2d' "
41             "grados.", temperatureInt, temperatureDec);
42     #if LLSEC802154_CONF_ENABLED
43     LOG_INFO(" LLSEC LV:%d", uipbuf_get_attr(UIPBUF_ATTR_LLSEC_LEVEL));
44     #endif
45     LOG_INFO("\n");
46 }
47 /*-----*/
48 PROCESS_THREAD(udp_client_process, ev, data) {
49     static struct etimer periodic_timer;
50     uip_ipaddr_t dest_ipaddr;
51
52     typedef float CelsiusDegrees_t;
53     CelsiusDegrees_t temperatureToSend;
54     uint16_t sensorRegisterValue;
55     uint8_t temperatureInt;
56     uint8_t temperatureDec;
57
58     PROCESS_BEGIN();
59
60     /* Initialize UDP connection */
61     simple_udp_register(&udp_conn, UDP_CLIENT_PORT, NULL, UDP_SERVER_PORT,
62                         udp_rx_callback);
63
64     etimer_set(&periodic_timer, random_rand() % SEND_INTERVAL);
65     while (1) {
66         PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&periodic_timer));
67
68         if (NETSTACK_ROUTING.node_is_reachable() &&
69             NETSTACK_ROUTING.get_root_ipaddr(&dest_ipaddr)) {
70             SENSORS_ACTIVATE(temperature_sensor);
71             sensorRegisterValue = temperature_sensor.value(0);
72             SENSORS_DEACTIVATE(temperature_sensor);
73             temperatureInt = (sensorRegisterValue >> 2);
74             temperatureDec = ((sensorRegisterValue & 0x3) * 25U);
75             LOG_INFO("Enviando temperatura interna en Celsius = %.2d.%2d\n",
76                     temperatureInt, temperatureDec);
77             temperatureToSend = ((CelsiusDegrees_t)temperatureInt +
78                                 ((CelsiusDegrees_t)temperatureDec / 100.0F));
79             simple_udp_sendto(&udp_conn, (void *)&temperatureToSend,
80                               sizeof(temperatureToSend), &dest_ipaddr);
81         } else {
82

```

```

83     LOG_INFO("Not reachable yet\n");
84 }
85
86 /* Add some jitter */
87 etimer_set(&periodic_timer, SEND_INTERVAL - CLOCK_SECOND +
88             (random_rand() % (2 * CLOCK_SECOND)));
89 }
90
91 PROCESS_END();
92 }
93 /*-----*/

```

udp-client.c

```

1 /*
2  * Redistribution and use in source and binary forms, with or without
3  * modification, are permitted provided that the following conditions
4  * are met:
5  * 1. Redistributions of source code must retain the above copyright
6  *    notice, this list of conditions and the following disclaimer.
7  * 2. Redistributions in binary form must reproduce the above copyright
8  *    notice, this list of conditions and the following disclaimer in the
9  *    documentation and/or other materials provided with the distribution.
10 * 3. Neither the name of the Institute nor the names of its contributors
11 *    may be used to endorse or promote products derived from this software
12 *    without specific prior written permission.
13 *
14 * THIS SOFTWARE IS PROVIDED BY THE INSTITUTE AND CONTRIBUTORS ``AS IS'' AND
15 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
16 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
17 * ARE DISCLAIMED. IN NO EVENT SHALL THE INSTITUTE OR CONTRIBUTORS BE LIABLE
18 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
19 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
20 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
21 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
22 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
23 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
24 * SUCH DAMAGE.
25 *
26 * This file is part of the Contiki operating system.
27 *
28 */
29
30 #include "contiki.h"
31 #include "net/ipv6/simple-udp.h"
32 #include "net/netstack.h"
33 #include "net/routing/routing.h"
34 #include "sys/log.h"
35
36 #include <stdint.h>
37
38 #define LOG_MODULE "Server"
39 #define LOG_LEVEL LOG_LEVEL_INFO
40
41 #define WITH_SERVER_REPLY 1
42 #define UDP_CLIENT_PORT 8765
43 #define UDP_SERVER_PORT 5678
44
45 static struct simple_udp_connection udp_conn;
46
47 PROCESS(udp_server_process, "UDP server");
48 AUTOSTART_PROCESSES(&udp_server_process);
49 /*-----*/
50 static void udp_rx_callback(struct simple_udp_connection *c,
51                             const uip_ipaddr_t *sender_addr,
52                             uint16_t sender_port,
53                             const uip_ipaddr_t *receiver_addr,
54                             uint16_t receiver_port, const uint8_t *data,

```

```

55         uint16_t datalen) {
56
57     typedef float CelsiusDegrees_t;
58     typedef float FahrenheitDegrees_t;
59     CelsiusDegrees_t receivedTemperature;
60     uint8_t temperatureInt;
61     uint8_t temperatureDec;
62     FahrenheitDegrees_t processedTemperature;
63
63     receivedTemperature = *((float *)data);
64     temperatureInt = (uint8_t)receivedTemperature;
65     temperatureDec = ((receivedTemperature - (float)temperatureInt) * 100U);
66     LOG_INFO("Info recibida del nodo: '%.2d.%.2d' grados\n", temperatureInt,
67               temperatureDec);
68     LOG_INFO("Direccion = ");
69     LOG_INFO_6ADDR(sender_addr);
70     LOG_INFO_(("\n"));
71     LOG_INFO_("TEMPERATURA RECIBIDA = %.2d.%.2d\n", temperatureInt,
72               temperatureDec);
73
74     processedTemperature = (2 * receivedTemperature + 32.0F);
75
76 #if WITH_SERVER_REPLY
77 /* send back the same string to the client as an echo reply */
78     temperatureInt = (uint8_t)processedTemperature;
79     temperatureDec = ((processedTemperature - (float)temperatureInt) * 100U);
80     LOG_INFO("Enviando temperatura en Fahrenheit = %.2d.%.2d\n", temperatureInt,
81               temperatureDec);
82     LOG_INFO("Destino = ");
83     LOG_INFO_6ADDR(sender_addr);
84     LOG_INFO_(("\n"));
85     simple_udp_sendto(&udp_conn, (void *)&processedTemperature,
86                       sizeof(processedTemperature), sender_addr);
87 #endif /* WITH_SERVER_REPLY */
88 }
89 /*-----*/
90 PROCESS_THREAD(udp_server_process, ev, data) {
91     PROCESS_BEGIN();
92
93     /* Initialize DAG root */
94     NETSTACK_ROUTING.root_start();
95
96     /* Initialize UDP connection */
97     simple_udp_register(&udp_conn, UDP_SERVER_PORT, NULL, UDP_CLIENT_PORT,
98                         udp_rx_callback);
99
100    PROCESS_END();
101 }
102 /*-----*/

```

udp-server.c

3.2. Ejercicio 2

A pesar de que como se ha indicado en el ejercicio anterior, el canal y PANIDs utilizados por los nodos se defina dentro de la cabecera de configuración de Contiki */contiki-ng/os/contiki-default-conf.h*, este archivo pertenece al propio sistema operativo, y modificarlo supondría hacerla para cualquier proyecto, por lo que no es una práctica aconsejada.

Por ello, para este ejercicio se ha preferido trabajar con una configuración local del proyecto. Para ello se ha creado una cabecera *project-conf.h*, que sobreescribe los valores de las macros del archivo anterior. Dentro de este, se ha elegido un

PANID=0xF00D y **canal=15**. El motivo para este segundo valor es el de evitar interferencias con posible actividad Wifi en la zona, dado que este canal no solaparía siguiendo el estándar IEEE802.11b para redes WLAN europeas y americanas.

```

1  *-----*/
2  /* Link-layer options
3  */
4
5  /* IEEE802154_CONF_PANID defines the default PAN ID for IEEE 802.15.4 networks */
6  #ifndef IEEE802154_CONF_PANID
7  #define IEEE802154_CONF_PANID 0xF00D
8  #endif /* IEEE802154_CONF_PANID */
9
10 /* IEEE802154_CONF_DEFAULT_CHANNEL defines the default channel for IEEE 802.15.4
11 * networks, for MAC layers without a channel selection or channel hopping
12 * mechanism. Current 802.15.4 MAC layers:
13 * - CSMA: uses IEEE802154_CONF_DEFAULT_CHANNEL
14 * - TSCW: uses its own TSCH_DEFAULT_HOPPING_SEQUENCE instead
15 */
16 #ifndef IEEE802154_CONF_DEFAULT_CHANNEL
17 #define IEEE802154_CONF_DEFAULT_CHANNEL 15
18 #endif /* IEEE802154_CONF_DEF_CHANNEL */

```

project-conf.h

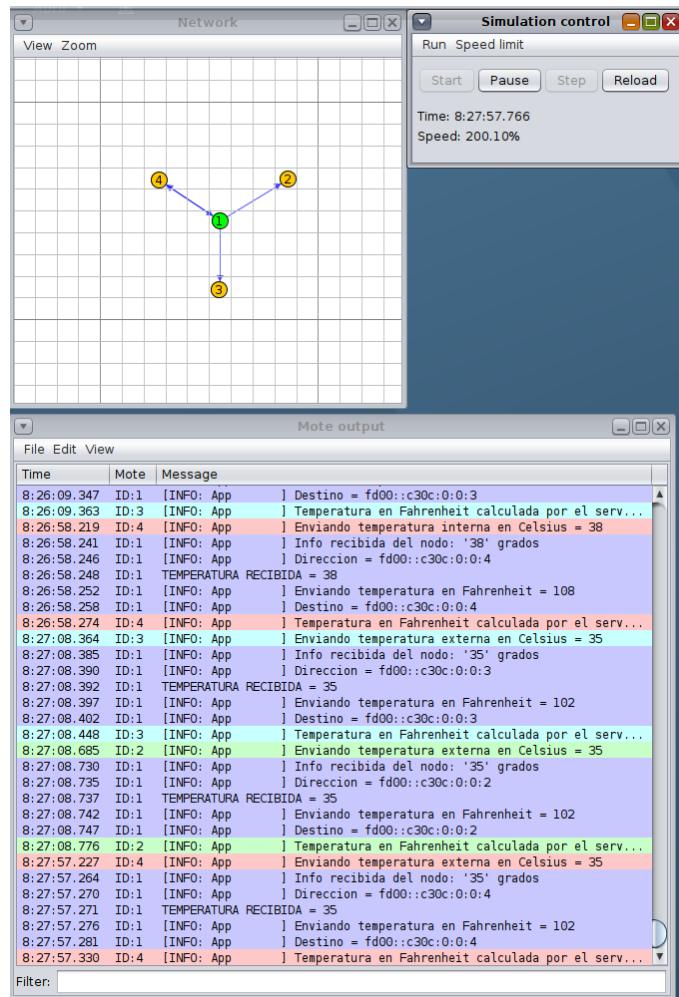


Figura 3.3: Simulación de red en estrella del ejercicio

3.3. Ejercicio 3

Se procede al despliegue de las redes de los ejercicios anteriores en el simulador de Cooja. Sin embargo, dado que los nodos usados en la realidad para el ejercicio 1 no están disponibles en esta aplicación, se recurre al Sky como servidor y al Z1 como cliente. Además, como solicita el enunciado de este ejercicio, contará con 3 nodos cliente en lugar de uno sólo.

La siguiente captura demuestra la utilidad de estos parámetros a la hora de configurar una red. En el caso del PANID, nos permite justamente diferenciar 2 redes desplegadas en la misma ubicación, mientras que el canal nos permite evitar interferencias al realizar la comunicación física sin solapes de manera simultánea.

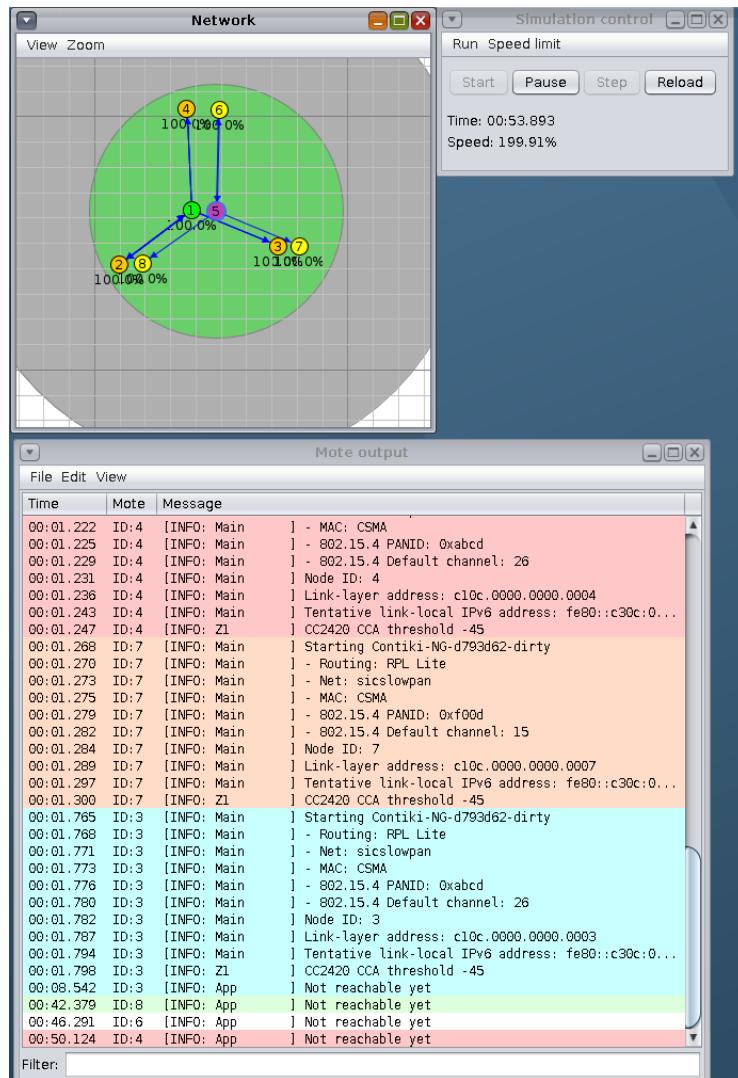


Figura 3.4: Despliegue de 2 redes simultáneas con PANIDs y canales distintos

Capítulo 4

Monitorización de la información

4.1. Ejercicio 1

En este ejercicio se pretende leer la lectura de la temperatura interna de la CPU, enviar esta temperatura en grados Fahrenheit a mqtt-exporter con una periodicidad de 2 segundos a través del puerto serie, comunicarlo con Prometheus a través del tópico "temp_F" para alojar los datos, y por último llevar a cabo la representación final al usuario en Grafana.

Modificando levemente el código de la práctica anterior, se procede a integrar la conversión a grados Fahrenheit de la temperatura leída por el sensor, que previamente realizaba el nodo servidor. Teniendo en cuenta que mqtt-exporter "parseará" los mensajes que se envíen por puerto serie para captar la información, se envía la temperatura ya convertida con printf(). El código resultante es el siguiente:

```
1 /**
2  * \file
3  *      Ejercicio 1: Medida de la temperatura interna
4  * \author
5  *      Sergio Leon <schoolion01@gmail.com>
6 */
7
8 #include "contiki.h"
9 #include "lib/sensors.h"
10 #include "temperature-sensor.h"
11
12 #include <stdint.h>
13 #include <stdio.h>
14
15 #define PROCESS_EVENT_AWAKE 0 // Custom user defined event identifier
16 /*-----*/
17 PROCESS(sensor_reading, "Sensor reading process");
18 PROCESS(timer_process, "Timer process");
19 AUTOSTART_PROCESSES(&sensor_reading, &timer_process);
20 /*-----*/
21 PROCESS_THREAD(sensor_reading, ev, data) {
22     uint16_t temperature_register_value;
23     struct temperature_t {
24         uint8_t temperatureDec;
25         uint8_t temperatureInt;
26     };
27     struct temperature_t temperature_C;
28     struct temperature_t temperature_F;
29
30     PROCESS_BEGIN();
```

```

31
32     while (1) {
33         /* Read temperature value from sensor*/
34         SENSORS_ACTIVATE(temperature_sensor);
35         temperature_register_value = temperature_sensor.value(0);
36         SENSORS_DEACTIVATE(temperature_sensor);
37
38         /* Print read value */
39         temperature_C.temperatureInt = (temperature_register_value >> 2);
40         temperature_C.temperatureDec = (temperature_register_value & 0x3) * 25U;
41         temperature_F.temperatureInt = (2U * temperature_C.temperatureInt + 32U);
42         temperature_F.temperatureDec = (2U * temperature_C.temperatureDec);
43         if (temperature_F.temperatureDec >= 100U) {
44             temperature_F.temperatureDec -= 100U;
45             temperature_F.temperatureInt++;
46         }
47
48         printf("%.2d.%2d\n", temperature_F.temperatureInt,
49                temperature_F.temperatureDec);
50
51         /* Wait to receive an event in order to read again */
52         PROCESS_WAIT_EVENT_UNTIL(ev == PROCESS_EVENT_AWAKE);
53     }
54
55     PROCESS_END();
56 }
57
58 PROCESS_THREAD(timer_process, ev, data) {
59     static struct etimer timer;
60
61     PROCESS_BEGIN();
62
63     /* Setup a periodic timer that expires after 2 seconds. */
64     etimer_set(&timer, CLOCK_SECOND * 2);
65
66     while (1) {
67         /* Wait for the periodic timer to expire and then restart the timer. */
68         PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&timer));
69         etimer_reset(&timer);
70
71         process_post_sync(&sensor_reading, PROCESS_EVENT_AWAKE, NULL);
72     }
73
74     PROCESS_END();
75 }
76 /*-----*/

```

temperature.c

Además de ello, se modifica ligeramente el archivo main.py facilitado para programar el comportamiento de mqtt_exporter, de forma que en los métodos de suscripción y publicación tenga como parámetro el tópico "temp_F".

Con el comando `docker-compose up` levantamos los servicios necesarios para nuestra aplicación. Lo primero de todo a fijarse es en el log recibido que indica ya un correcto funcionamiento de los datos enviados por mqtt_exporter a través del tópico deseado:

```

Reporte Practicas RIS - Ejercicio 1
localhost:9000
Payload: 94.25 | Serial Data: 94.50
ejercicio1-mqtt-exporter-1 | Field[0]: 94.50
ejercicio1-mqtt-exporter-1 | [Topic: temp_F] b'94.50'
ejercicio1-mqtt-exporter-1 | Topic: temp_F
ejercicio1-mqtt-exporter-1 | Payload: 94.5
ejercicio1-mqtt-exporter-1 | Serial Data: 96.00
ejercicio1-mqtt-exporter-1 | Field[0]: '96.000000
ejercicio1-mqtt-exporter-1 | [Topic: temp_F] b'96.00'
ejercicio1-mqtt-exporter-1 | Topic: temp_F
ejercicio1-mqtt-exporter-1 | Payload: 96.0
ejercicio1-mqtt-exporter-1 | Serial Data: 96.25
ejercicio1-mqtt-exporter-1 | Field[0]: 96.250000
ejercicio1-mqtt-exporter-1 | [Topic: temp_F] b'96.25'
ejercicio1-mqtt-exporter-1 | Topic: temp_F
ejercicio1-mqtt-exporter-1 | Payload: 96.25
ejercicio1-mqtt-exporter-1 | Serial Data: 96.50
ejercicio1-mqtt-exporter-1 | Field[0]: 96.500000
ejercicio1-mqtt-exporter-1 | [Topic: temp_F] b'96.50'
ejercicio1-mqtt-exporter-1 | Topic: temp_F
ejercicio1-mqtt-exporter-1 | Payload: 96.5
ejercicio1-mqtt-exporter-1 | Serial Data: 96.50
ejercicio1-mqtt-exporter-1 | Field[0]: 96.500000
ejercicio1-mqtt-exporter-1 | [Topic: temp_F] b'96.50'
ejercicio1-mqtt-exporter-1 | Topic: temp_F
ejercicio1-mqtt-exporter-1 | Payload: 96.5
ejercicio1-mqtt-exporter-1 | Serial Data: 96.75
ejercicio1-mqtt-exporter-1 | Field[0]: 96.750000

```

Figura 4.1: Log de la terminal de docker

A través del navegador web, usando localhost:9000, accedemos al servicio de mqtt_exporter, donde podemos comprobar la temperatura que el cliente está obteniendo del sensor a través del mensaje por puerto serie.

```

localhost:9000

# HELP python_gc_objects_collected_total Objects collected during gc
# TYPE python_gc_objects_collected_total counter
python_gc_objects_collected_total{generation="0"} 294.0
python_gc_objects_collected_total{generation="1"} 195.0
python_gc_objects_collected_total{generation="2"} 0.0
# HELP python_gc_objects_uncollectable_total Uncollectable objects found during GC
# TYPE python_gc_objects_uncollectable_total counter
python_gc_objects_uncollectable_total{generation="0"} 0.0
python_gc_objects_uncollectable_total{generation="1"} 0.0
python_gc_objects_uncollectable_total{generation="2"} 0.0
# HELP python_gc_collections_total Number of times this generation was collected
# TYPE python_gc_collections_total counter
python_gc_collections_total{generation="0"} 44.0
python_gc_collections_total{generation="1"} 4.0
python_gc_collections_total{generation="2"} 0.0
# HELP python_info Python platform information
# TYPE python_info gauge
python_info{implementation="CPython",major="3",minor="10",patchlevel="13",version="3.10.13"} 1.0
# HELP process_virtual_memory_bytes Total virtual memory size in bytes.
# TYPE process_virtual_memory_bytes gauge
process_virtual_memory_bytes 2.0088e+07
# HELP process_resident_memory_bytes Resident memory size in bytes.
# TYPE process_resident_memory_bytes gauge
process_resident_memory_bytes 2.0619264e+07
# HELP process_start_time_seconds Start time of the process since unix epoch in seconds.
# TYPE process_start_time_seconds gauge
process_start_time_seconds 1.70415175765e+09
# HELP process_cpu_seconds_total Total user and system CPU time spent in seconds.
# TYPE process_cpu_seconds_total counter
process_cpu_seconds_total 0.16
# HELP process_open_fds Number of open file descriptors.
# TYPE process_open_fds gauge
process_open_fds 15.0
# HELP process_max_fds Maximum number of open file descriptors.
# TYPE process_max_fds gauge
process_max_fds 1.048576e+06
# HELP number_msgs_total Number of received messages
# TYPE number_msgs_total counter
number_msgs_total 32.0
# HELP number_msgs_created Number of received messages
# TYPE number_msgs_created gauge
number_msgs_created 1.7041517586698406e+09
# HELP temp Temperature [Fahrenheit Degrees]
# TYPE temp gauge
temp 96.75

```

Figura 4.2: Servicio de mqtt_exporter

Luego, escuchando por el puerto 9090, accedemos al servicio web de Prometheus, donde podemos ver los datos que están siendo registrados de las lecturas del sensor.

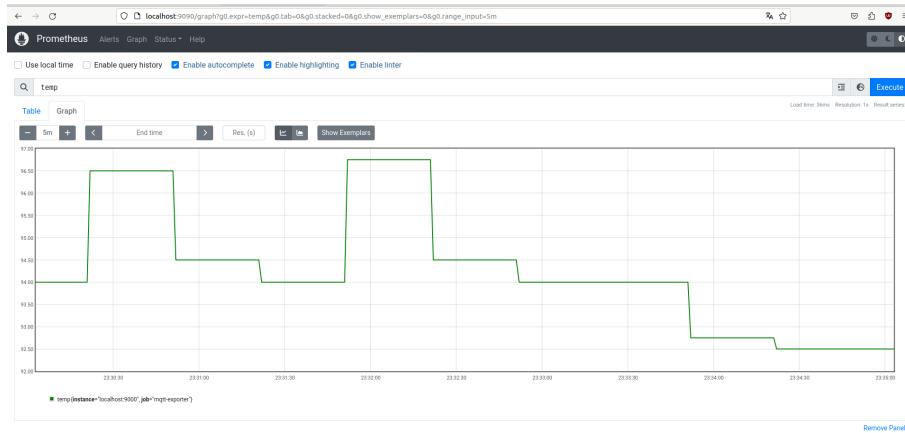


Figura 4.3: Gráfico de Prometheus

Por último, haciendo uso del puerto 3000, accedemos al portal de Grafana, donde tras configurar de forma amigable un dashboard con los widgets elegidos, se obtiene como resultado final:

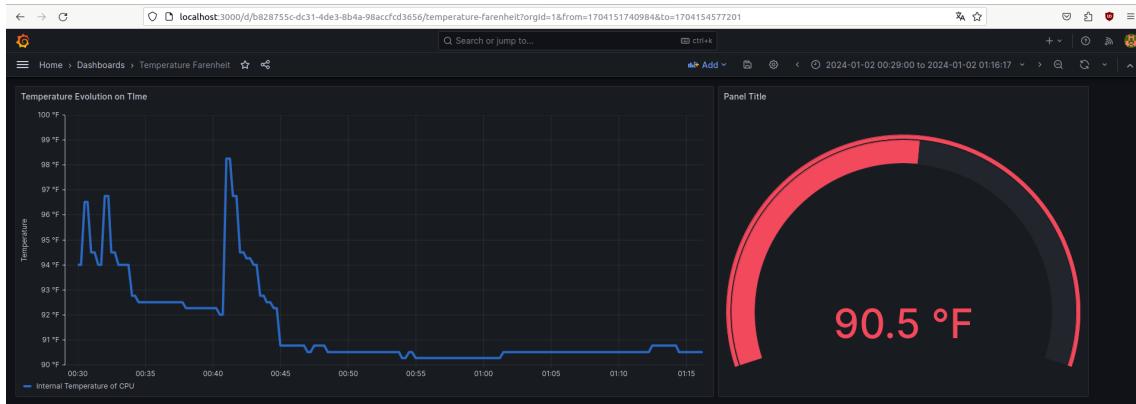


Figura 4.4: Dashboard de Grafana

4.2. Ejercicio 2

De forma similar al ejercicio anterior, se modifica el código fuente del programa a flashear en el nodo, para que esta vez envíe por puerto serie con formato CSV 2 datos simultáneamente: temperatura en grados Celsius, temperatura en grados Fahrenheit. El primero de ellos debe enviarse por un tópico de nombre "temp_C", mientras que el segundo se mantendrá publicado por "temp_F". Además, se añade un tercer dato asociado al valor del pulsador del dongle, realizando enclavamiento y anti-debouncing software, para tratarlo como un interruptor ideal, y publicarlo por el tópico "switch".

Para este último propósito, se recurre a la librería de la capa HAL que provee el

sistema operativo. Ya que en las cabeceras de nuestra plataforma se define la macro PLATFORM_HAS_BUTTON, la función button_hal_init(), encargada de activar los manejadores para las interrupciones hardware por GPIO configuradas para el botón de la placa, ya es llamada por la función principal del sistema operativo en la inicialización de plataforma en el estadio 2 (platform_init_stage_two). Además, ya utiliza internamente timers configurados para evitar el rebote en la lectura del valor del pulsador. Por tanto, la única tarea a resolver ha sido la de realizar enclavamiento software. Dado que se han definido como símbolos el '1' para indicar que el interruptor se encuentra liberado, y '2' para indicar que se encuentra presionado, el hecho de alternar su valor en cada pulsación es fácil de resolver negando los bits de la variable y seleccionando con una máscara los 2 menos significativos.

Para este programa, se divide la ejecución en 3 procesos diferentes:

- temperature_sensor_reading: Recibirá eventos de tipo PROCESS_EVENT_REFRESH por parte del proceso serial_port_sender, de manera que su única función será despertar por orden de este último para actualizar la lectura de medición del sensor de temperatura, calculando a su misma vez la conversión en Fahrenheit.
- button_status_reading: Realizará una lectura inicial del estado del botón para inicializar la variable global virtual_switch_state. Posteriormente, estará a la espera de recepción de eventos generados en relación al botón (por interrupción hardware: bien sea de pulsación, liberación del botón, o software: evento periódico para el conteo de segundos pulsados), tras lo cuál de recibir un evento del tipo button_hal_press_event intercambiará el estado.
- serial_port_sender: Podría considerarse el proceso principal. De forma periódica, lo cuál le vendrá marcado por los eventos de expiración de su propio temporizador, realizará POST síncronos al proceso de lectura del sensor, para asegurarse de tener el último valor disponible, tras lo cuál retomará su ejecución para enviar por puerto serie una cadena con formato CSV: temperatura en Celsius, temperatura en Fahrenheit, estado del pulsador.

Cabe destacar que dada la naturaleza mononúcleo del procesador del nrf52840, no es necesario tener consideraciones de condiciones de carrera y acceso simultáneo a las variables compartidas de los procesos, dado que no puede haber varios procesos corriendo simultáneamente en esta plataforma. De esta manera, el código resultante es:

```

1 /**
2 * \file
3 *      Ejercicio 2: Medida de la temperatura interna + Interruptor
4 * \author
5 *      Sergio Leon <schoolion01@gmail.com>
6 */
7
8 #include "button-hal.h"
9 #include "contiki.h"
10 #include "lib/sensors.h"
11 #include "temperature-sensor.h"
12
13 #include <stdint.h>
14 #include <stdio.h>
```

```

15 #define PROCESS_EVENT_REFRESH
16     0 // Custom user event identifier to awake a process in order to update a
17     // measurement
18
19 enum switch_status { OPEN = 1, CLOSE = 2 };
20 static enum switch_status virtual_switch_state;
21 struct temperature_t {
22     uint8_t temperatureDec;
23     uint8_t temperatureInt;
24 };
25
26 static struct temperature_t temperature_C;
27 static struct temperature_t temperature_F;
28 */
29 PROCESS(temperature_sensor_reading,
30         "Sensor reading process"); // Process to update temperature sensor
31                                     // measurement
32 PROCESS(button_status_reading,
33         "Sensor reading process"); // Process to receive hardware interruptions
34                                     // from physical button
35 PROCESS(serial_port_sender,
36         "Sensor reading process"); // Process to send data to mqtt_exporter via
37                                     // serial port msgs
38 AUTOSTART_PROCESSES(&temperature_sensor_reading, &button_status_reading,
39                     &serial_port_sender);
40 */
41 PROCESS_THREAD(temperature_sensor_reading, ev, data) {
42     uint16_t temperature_register_value;
43
44     PROCESS_BEGIN();
45
46     while (1) {
47         /* Read temperature value from sensor*/
48         SENSORS_ACTIVATE(temperature_sensor);
49         temperature_register_value = temperature_sensor.value(0);
50         SENSORS_DEACTIVATE(temperature_sensor);
51
52         /* Print read value */
53         temperature_C.temperatureInt = (temperature_register_value >> 2);
54         temperature_C.temperatureDec = (temperature_register_value & 0x3) * 25U;
55         temperature_F.temperatureInt = (2U * temperature_C.temperatureInt + 32U);
56         temperature_F.temperatureDec = (2U * temperature_C.temperatureDec);
57         if (temperature_F.temperatureDec >= 100U) {
58             temperature_F.temperatureDec -= 100U;
59             temperature_F.temperatureInt++;
60         }
61
62         /* Wait to receive an event in order to read again */
63         PROCESS_WAIT_EVENT_UNTIL(ev == PROCESS_EVENT_REFRESH);
64     }
65
66     PROCESS_END();
67 }
68
69 PROCESS_THREAD(button_status_reading, ev, data) {
70     static button_hal_button_t *button_p;
71     PROCESS_BEGIN();
72
73     button_p = button_hal_get_by_id(BUTTON_HAL_ID_BUTTON_ZERO);
74
75     if (button_hal_get_state(button_p)) {
76         virtual_switch_state = CLOSE;
77     } else {
78         virtual_switch_state = OPEN;
79     }
80
81     while (1) {
82         /* Wait to receive a hardware interruption event caused by button */
83         PROCESS_WAIT_EVENT_UNTIL(ev == button_hal_press_event);
84         virtual_switch_state = (~virtual_switch_state & 0x3);

```

```

85    }
86
87    PROCESS_END();
88}
89
90PROCESS_THREAD(serial_port_sender, ev, data) {
91    static struct etimer timer;
92
93    PROCESS_BEGIN();
94
95    /* Setup a periodic timer that expires after 2 seconds. */
96    etimer_set(&timer, CLOCK_SECOND * 2);
97
98    while (1) {
99        /* Wait for the periodic timer to expire and then restart the timer. */
100       PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&timer));
101       etimer_reset(&timer);
102
103       process_post_synch(&temperature_sensor_reading, PROCESS_EVENT_REFRESH,
104                           NULL);
105
106       printf("%.2d %.2d;%.2d %.2d;%d\n", temperature_C.temperatureInt,
107              temperature_C.temperatureDec, temperature_F.temperatureInt,
108              temperature_F.temperatureDec, virtual_switch_state);
109    }
110
111    PROCESS_END();
112}
113/*-----*/

```

application.c

A su vez, también se han añadido nuevos gauges y tópicos en el script de Python de mqtt_exporter, reconstruyendo los contenedores de los servicios, para transmisir así los 3 datos que se obtienen ahora.

Finalmente, el resultado de cara al usuario final del producto es el mostrado en las siguientes imágenes:

```

localhost:9000      ×   Prometheus Time Series ×   Application Exercise 2 - ×   +
← → C           ⌂   localhost:9000

# HELP python_gc_objects_collected_total Objects collected during gc
# TYPE python_gc_objects_collected_total counter
python_gc_objects_collected_total{generation="0"} 3624.0
python_gc_objects_collected_total{generation="1"} 243.0
python_gc_objects_collected_total{generation="2"} 0.0
# HELP python_gc_objects_uncollectable_total Uncollectable objects found during GC
# TYPE python_gc_objects_uncollectable_total counter
python_gc_objects_uncollectable_total{generation="0"} 0.0
python_gc_objects_uncollectable_total{generation="1"} 0.0
python_gc_objects_uncollectable_total{generation="2"} 0.0
# HELP python_gc_collections_total Number of times this generation was collected
# TYPE python_gc_collections_total counter
python_gc_collections_total{generation="0"} 48.0
python_gc_collections_total{generation="1"} 4.0
python_gc_collections_total{generation="2"} 0.0
# HELP python_info Python platform information
# TYPE python_info gauge
python_info{implementation="CPython",major="3",minor="10",patchlevel="13",version="3.10.13"} 1.0
# HELP process_virtual_memory_bytes Virtual memory size in bytes.
# TYPE process_virtual_memory_bytes gauge
process_virtual_memory_bytes 2.7901952e+07
# HELP process_resident_memory_bytes Resident memory size in bytes.
# TYPE process_resident_memory_bytes gauge
process_resident_memory_bytes 2.0545536e+07
# HELP process_start_time_seconds Start time of the process since unix epoch in seconds.
# TYPE process_start_time_seconds gauge
process_start_time_seconds 1.7042304894e+09
# HELP process_cpu_seconds_total Total user and system CPU time spent in seconds.
# TYPE process_cpu_seconds_total counter
process_cpu_seconds_total 0.7300000000000001
# HELP process_open_fds Number of open file descriptors.
# TYPE process_open_fds gauge
process_open_fds 15.0
# HELP process_max_fds Maximum number of open file descriptors.
# TYPE process_max_fds gauge
process_max_fds 1.048576e+06
# HELP number_msgs_total Number of received messages
# TYPE number_msgs_total counter
number_msgs_total 717.0
# HELP number_msgs_created Number of received messages
# TYPE number_msgs_created gauge
number_msgs_created 1.7042304859506698e+09
# HELP tempCelsius Temperature [Celsius Degrees]
# TYPE tempCelsius gauge
tempCelsius 32.75
# HELP tempFahrenheit Temperature [Fahrenheit Degrees]
# TYPE tempFahrenheit gauge
tempFahrenheit 97.5
# HELP switchStatus Switch Status [Open "1", Close "2"]
# TYPE switchStatus gauge
switchStatus 2.0

```

Figura 4.5: Servicio de mqtt_exporter



Figura 4.6: Servicio de Prometheus, tópico de temperatura en grados Celsius

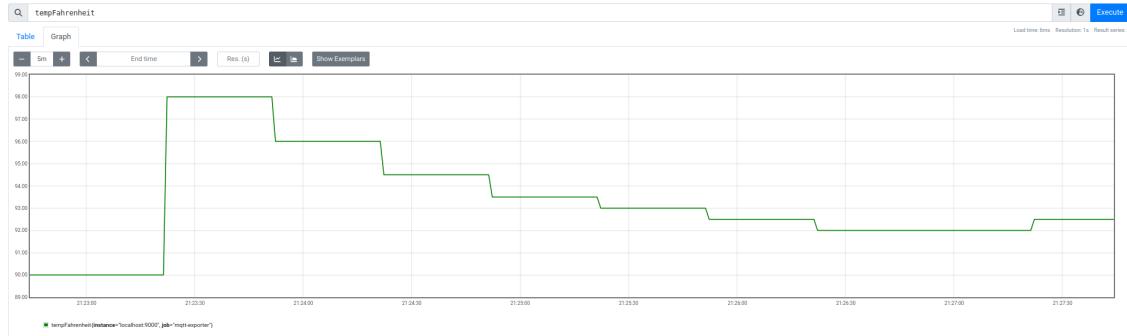


Figura 4.7: Servicio de Prometheus, tópico de temperatura en grados Fahrenheit

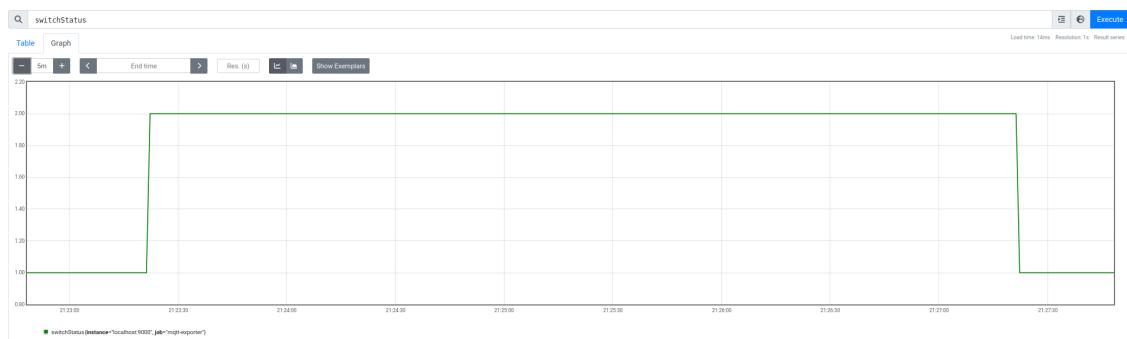


Figura 4.8: Servicio de Prometheus, tópico del estado del interruptor

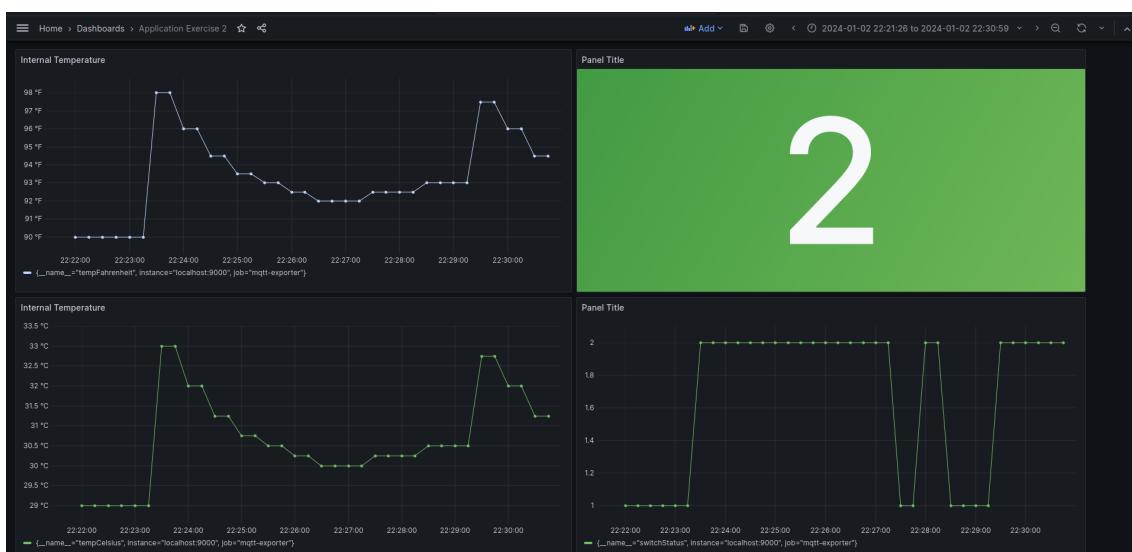


Figura 4.9: Servicio de Grafana mostrando todos los datos

Se adjunta junto a esta memoria un archivo comprimido que contiene el código fuente y simulaciones de todos los ejercicios que componente estas prácticas, siguiendo una estructura de directorios similar a la marcada por el índice de este documento

Bibliografía

- [1] Contiki-NG. *Multitasking and scheduling*. 2023. URL: <https://docs.contiki-ng.org/en/develop/doc/programming/Multitasking-and-scheduling.html#the-contiki-ng-scheduler-and-event-dispatch>.
- [2] Contiki-NG. *Processes and events*. 2023. URL: <https://docs.contiki-ng.org/en/develop/doc/programming/Processes-and-events.html>.