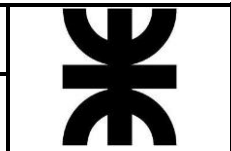


Laboratorio Remoto para Dispositivos IIoT

Manual Usuario



2024



Contenido

1.	Introducción	3
2.	Ejecución de reserva.....	4
2.1.	Reserva mediante apache VCL	4
2.2.	Uso de reserva	5
2.3.	Acceso al recurso	7
2.4.	Verificación del entorno.....	8
3.	Programación y carga de firmware.....	9
3.1.	Programa hello-world	9
3.2.	Ejemplo uso de sensor de temperatura.....	13
3.3.	Ejemplo de realización de comunicación CoAP	15
3.4.	Ejemplo de realización de comunicación MQTT	21
3.4.1.	Breve explicación y funcionamiento de MQTT	21
3.4.2.	Programación de ejemplo MQTT	23
3.5.	Sniffer de red	25
	Anexo A: sistema contiki-NG y organización de directorios	29
	Anexo B: Protohilos de Contiki-NG	33

1. Introducción

El LRDIIoT (Laboratorio Remoto para Dispositivos IIoT) es una plataforma que permite el acceso remoto a entornos de desarrollo placas IIoT, concretamente el modelo openmote-cc2538. Vale aclarar que a diferencia de muchos dispositivos IIoT, estas placas pueden ser programadas y comunicarse entre ellas, lo que permite realizar diversos experimentos desde el punto de vista de tráfico, protocolos, arquitectura y comunicación.

El entorno de desarrollo de cada uno de los usuarios está compuesto por cuatro placas, una de ellas conectada a una placa raspberry PI y las tres placas restantes están conectadas directamente a un contenedor LxC.

La primera de ellas en conjunto con la placa raspberry PI se identificará como nodo coordinador, mientras que las tres restantes se las identifica como nodos sensores. A estos últimos se conectan múltiples sensores o se utilizan los sensores embebidos en la placa.

El LRDIIoT está conformado por varios módulos. Desde el punto de vista del usuario, cuatro módulos son relevantes:

- Gestión de reservas (Apache VCL)
- Interfaz de acceso remoto (Apache Guacamole)
- Interfaz de base de datos (MySQL)
- Interfaz de visualización (Grafana)

Para poder acceder a los recursos computacionales (línea de comando lab-iot con sus correspondientes placas openmote) el usuario debe realizar la reserva de un recurso a través de Apache VCL. Existen dos tipos de reserva, una que puede ser utilizada en el momento y otra que puede ser utilizada en el futuro.

En el caso de querer usarla en el momento, se debe constatar que existe disponibilidad para usar la misma ya que otro usuario podría estar utilizándola. En caso de que sea la reserva sea a futuro, en el día y la hora solicitado se debe reclamar nuevamente la reserva a través de Apache VCL. Para acceder a la terminal de programación se deberá hacer uso de Guacamole, donde se dispondrán de hasta 10 consolas de línea de comando para utilizar el entorno.

Como requisito fundamental es que el usuario disponga de un navegador y conexión a internet para utilizar el laboratorio remoto.

2. Ejecución de reserva

2.1. Reserva mediante apache VCL

Para realizar la reserva de un entorno de desarrollo, primero se debe verificar de utilizar el método de autenticación “lab-iot” y posteriormente seleccionar “Proceed to Login”.

En caso de querer cambiar el idioma, se puede realizar desde el menú desplegable ubicado en el extremo superior derecho.

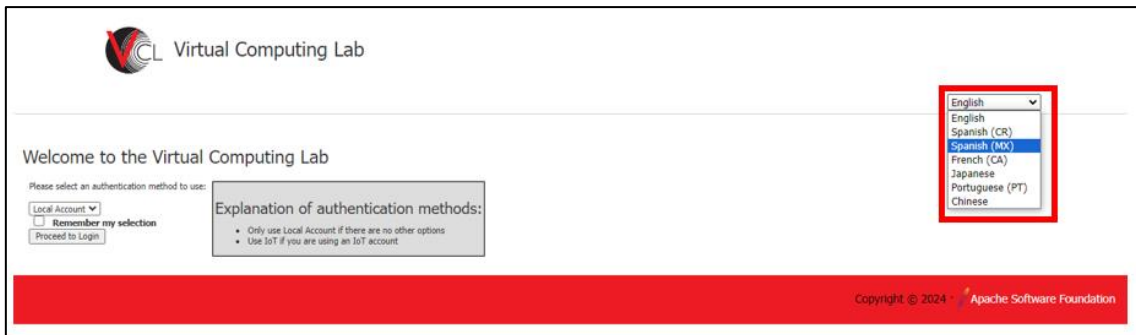


Figura 1: Interfaz VCL

Se debe ingresar las credenciales otorgadas por el administrador del laboratorio remoto con el objetivo de acceder a la página de inicio de usuario. En caso no poder ingresar con las credenciales otorgadas, consultar al soporte del laboratorio.

Conectarse con lab-iot

Usuario:

Contraseña:

[Select a different authentication method](#)

Figura 2: Logueo en VCL

El siguiente paso es seleccionar en el menú superior la opción “Reservaciones” y posteriormente en “Nueva reservación”. En caso de no visualizar el botón, consultar al soporte del laboratorio.

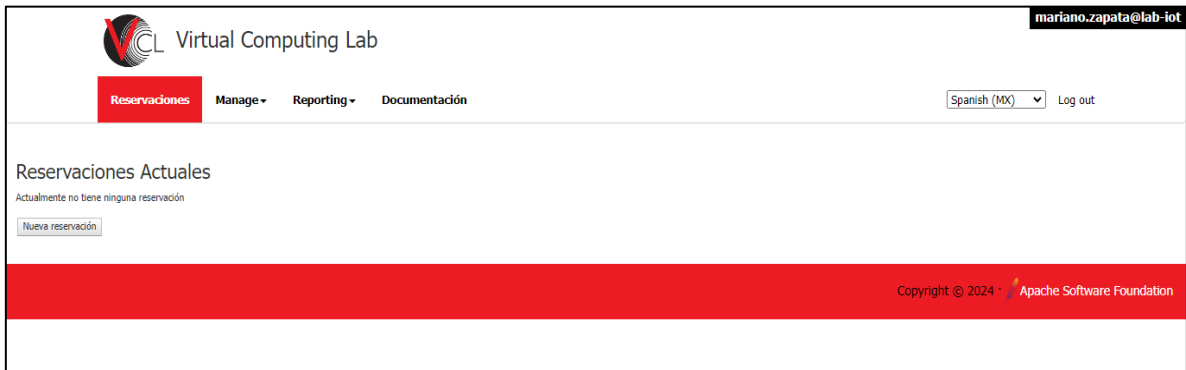


Figura 3: Interfaz reserva

En el cuadro emergente seleccionar lab-iot, hora y día para utilizar el recurso y la duración deseada. Finalmente se debe hacer clic en “Crear reservación”. El cuadro emergente se cerrará y se podrá visualizar el proceso de reserva.

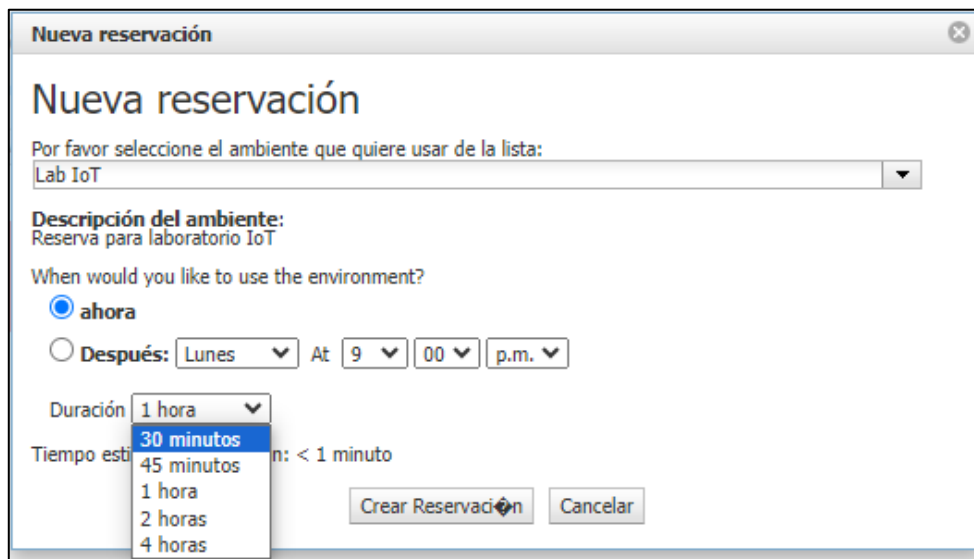


Figura 4: Creación de reserva

En caso de observar algún error, consultar al soporte del laboratorio.

2.2. Uso de reserva

En caso de realizar la reserva para ser utilizada en el momento, se debe hacer click en “Crear Reservación”. Por otro lado, si la reserva se hace para otro día, se debe nuevamente ingresar en la fecha y día elegido, seleccionar Reservaciones en el menú superior y seleccionar “Crear Reservación”.



Figura 5: Reserva exitosa

Si la reserva no es reclamada dentro de los 15 minutos de inicio, se pierde y se deberá realizar nuevamente el procedimiento de la reserva.

Posterior a realizar clic en “¡Conectar!” se abrirá un cuadro emergente con información sobre la reserva donde se debe tomar nota de la computadora Remota (Remote Computer) y posteriormente cerrar el cuadro. En caso de no recordar la computadora, se puede volver a realizar clic sobre el botón “¡Conectar!” para observar la identificación de la computadora.



Figura 6: Información de reserva

2.3. Acceso al recurso

Para acceder a apache Guacamole se debe hacer a través de la dirección:

<https://liderar-iot.utn.frn.edu.ar/guacamole>


Para ingresar se hace uso de las mismas credenciales que en Apache VCL.



Figura 7: Logueo en Apache Guacamole

Haciendo clic en el símbolo + a la izquierda de *lab-iot* se despliegan todas las conexiones disponibles, posteriormente se debe seleccionar el recurso reservado. En esta instancia se abrirá el entorno de desarrollo sin necesidad de ingresar nuevamente las credenciales ya que estas son gestionadas de forma automática por Apache Guacamole.

CONEXIONES RECIENTES

 mariano.zapata ▾

Sin conexiones recientes.

TODAS LAS CONEXIONES

 Filtros

```

└─ lab-iot
   ├── ide-iot-01 (10.0.0.101)
   └── ide-iot-02 (10.0.0.102)
  
```

Figura 8: Interfaz Apache Guacamole

Para abrir más de una consola en simultáneo (el límite es diez), se recomienda hacer clic derecho sobre la computadora y luego hacer clic en “*duplicar*”.

2.4. Verificación del entorno

Al realizar el primer ingreso al entorno, se debería observar una interfaz como la siguiente:

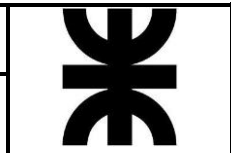
```

Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 6.5.11-8-pve x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/pro
New release '22.04.3 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Sat Apr 20 21:35:23 2024 from 10.0.0.150
mariano.zapata@ide-iot-01:~$ ls
contiki-ng
mariano.zapata@ide-iot-01:~$ █
  
```

Figura 9: Entorno de desarrollo



Mediante el comando `lsusb` se observarán las placas openmote conectadas:

```
mariano.zapata@ide-iot-01:~$ lsusb
Bus 006 Device 002: ID 05e3:0626 Genesys Logic, Inc. USB3.1 Hub
Bus 006 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 005 Device 003: ID 0403:6001 Future Technology Devices International, Ltd FT232 Serial (UART) IC
Bus 005 Device 004: ID 0403:6001 Future Technology Devices International, Ltd FT232 Serial (UART) IC
Bus 005 Device 002: ID 05e3:0610 Genesys Logic, Inc. 4-port hub
Bus 005 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 004 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 003 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 003: ID 0403:6001 Future Technology Devices International, Ltd FT232 Serial (UART) IC
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

Figura 10: Placas conectadas

Ingresando al directorio Contiki-NG se listan los archivos que forman parte del sistema operativo.

```
mariano.zapata@ide-iot-01:~$ cd contiki-ng
mariano.zapata@ide-iot-01:~/contiki-ng$ ls
CODE OF CONDUCT.md  Makefile.dir-variables  Makefile.identify-target  README.md  doc  tests
CONTRIBUTING.md    Makefile.embedded      Makefile.include          SECURITY.md  examples  tools
LICENSE.md          Makefile.help          Makefile.tools            arch        os
```

Figura 11: Contiki-NG

Nota: Para más información sobre la funcionalidad de los archivos que forman parte de Contiki-NG consultar el Anexo A.

Se pueden listar los puertos series utilizados por los dispositivos conectados al entorno de programación (deberían ser tres dispositivos seriales utilizados).

```
mariano.zapata@ide-iot-01:~/contiki-ng$ ls /dev/ttyUSB*
/dev/ttyUSB0 /dev/ttyUSB1 /dev/ttyUSB2
```

Figura 12: Puertos utilizados

3. Programación y carga de firmware

3.1. Programa hello-world

Dentro del directorio Contiki-NG se encuentra el directorio `examples` en el cual se encuentran múltiples ejemplos de uso de la placa `openmote-cc2538`:

```
mariano.zapata@ide-iot-01:~/contiki-ng/examples$ ls
6tisch  coap  dev  ip64-router  lwm2m-ipso-objects  multicast  platform-specific  rpl-udp  slip-radio  storage
benchmarks  cplusplus  hello-world  libs  mqtt-client  nullnet  rpl-border-router  sensniff  snmp-server  websocket
```

Figura 13: Directorio examples

Con objetivos didácticos se utilizará el directorio hello-world.

```
mariano.zapata@ide-iot-01:~/contiki-ng/examples/hello-world$ ls
Makefile README.md build hello-world.c
mariano.zapata@ide-iot-01:~/contiki-ng/examples/hello-world$ █
```

Figura 14: Directorio hello-world

El archivo a programar es el que tiene extensión .c. Para su modificación se puede usar un editor como *nano* o *vim*.

```
mariano.zapata@ide-iot-01:~/contiki-ng/examples/hello-world$ nano hello-world.c █
```

Figura 15: Edición de archivo .c

```
#include "contiki.h"
#include <stdio.h> /* For printf() */
/*-----*/
PROCESS(hello world process, "Hello world process");
AUTOSTART PROCESSES(&hello world process);
/*-----*/
PROCESS_THREAD(hello world process, ev, data)
{
  static struct etimer timer;

  PROCESS BEGIN();

  /* Setup a periodic timer that expires after 10 seconds. */
  etimer set(&timer, CLOCK_SECOND * 10);

  while(1) {
    printf("Hello, world\n");

    /* Wait for the periodic timer to expire and then restart the timer. */
    PROCESS_WAIT_EVENT_UNTIL(etimer expired(&timer));
    etimer reset(&timer);
  }

  PROCESS END();
}
/*-----*/
```

Figura 16: Programa hello-world

El programa realiza simplemente una impresión cada 10 segundos por el puerto serie del mensaje “Hello, world”.

Nota: Para más información sobre el formato de programación y código consultar el Anexo B.

Para realizar la compilación del programa y carga del firmware en las placas se debe utilizar la siguiente instrucción:

```
make TARGET=openmote BOARD=openmote-cc2538 BOARD_REVISION=REV_A1
PORT=/dev/ttyUSB* hello-world.upload
```

- **make:** es un comando utilizado en sistemas Unix/Linux para compilar y construir proyectos de software. En este contexto se utiliza para compilar, construir y cargar firmware en dispositivos IoT.
- **TARGET=openmote:** especifica el tipo de dispositivo de destino para el cual se está compilando el firmware. En este caso se trata de la placa openmote, pero podría ser otras que soporten Contiki-NG.
- **BOARD=openmote-cc2538:** indica la placa específica del dispositivo OpenMote que se está utilizando, en este caso es la placa con el chip CC2538.
- **BOARD_REVISION=REV_A1:** especifica la revisión de la placa que se está utilizando. (Esta especificación es necesaria en placas versión A1, ya que son dispositivos con menos memoria).
- **PORT=/dev/ttyUSB*:** indica el puerto al que está conectado el dispositivo OpenMote para la carga del firmware. Se debe colocar el número del puerto serie.
- **hello-world.upload:** especifica que la acción que se realizará con 'make'. En este caso se está realizando la carga ('upload') del firmware llamado 'hello-world' en el dispositivo OpenMote.

Posterior a esto se deberían compilar todas las librerías y cargar el firmware en la placa que se encuentra en el puerto serial ttyUSB*

```

CC      ../os/sys/mutex.c
CC      ../os/sys/stimer.c
CC      ../os/sys/compower.c
CC      ../os/sys/atomic.c
CC      ../os/sys/rtimer.c
CC      ../os/sys/node-id.c
CC      ../os/sys/process.c
CC      ../os/sys/timer.c
CC      ../os/sys/stack-check.c
LD      build/openmote/openmote-cc2538/hello-world.openmote
OBJCOPY build/openmote/openmote-cc2538/hello-world.openmote -> build/openmote/openmote-cc2538/hello-world.bin
../tools/cc2538-bsl/cc2538-bsl.py --bootloader-invert-lines -e -w -v -b 450000 -p /dev/ttyUSB0 -a 0x00202000 build/openmote/openmote-cc2538/hello-world.bin
Opening port /dev/ttyUSB0, baud 450000
Reading data from build/openmote/openmote-cc2538/hello-world.bin
Cannot auto-detect firmware filetype: Assuming .bin
Connecting to target...
CC2538 PG2.0: 256KB Flash, 32KB SRAM, CCFG at 0x0023FFD4
Primary IEEE Address: 00:12:4B:00:04:30:53:14
Performing mass erase
Erasing 262144 bytes starting at address 0x00200000
Erase done
Writing 253952 bytes starting at address 0x00202000
Write 248 bytes at 0x0023FF080
Write done
Verifying by comparing CRC32 calculations.
Verified (match: 0x29e2c71d)
mariano.zapata@ide-iot-01:~/contiki-ng/examples/hello-worlds

```

Figura 17: Compilación de firmware

Luego de realizar la compilación se creará un nuevo directorio llamado build en el cual se almacena el binario compilado, un archivo con extensión *.openmote*, los archivos *.o* y el header generado y utilizado.

Para poder visualizar la salida del firmware se debe usar la misma línea de comandos anterior con la diferencia de colocar login al final de la misma, esto permitirá ejecutar una terminal serial conectada con el openmote:

```
make TARGET=openmote BOARD=openmote-cc2538 BOARD_REVISION=REV_A1
PORT=/dev/ttyUSB* login
```

```

mariano.zapata@ide-iot-01:~/contiki-ng/examples/hello-world$ make TARGET=openmote BOARD=openmote-cc2538 BOARD_REVISION=REV_A1 PORT=/dev
/ttyUSB0 login
flwrp ../../tools/serial-io/serialedump -b115200 /dev/ttyUSB0
connecting to /dev/ttyUSB0 [OK]
Hello, world
Hello, world
  
```

Figura 18: Puerto serial

Los programas también pueden ser compilados de forma nativa, lo que permite ejecutarlos en el entorno de programación y no en la placa openmote.

Primero se debe ejecutar el siguiente comando:

Make TARGET=native

```

CC      ../../os/sys/etimer.c
CC      ../../os/sys/log.c
CC      ../../os/sys/mutex.c
CC      ../../os/sys/stimer.c
CC      ../../os/sys/compower.c
CC      ../../os/sys/atomic.c
CC      ../../os/sys/rtimer.c
CC      ../../os/sys/node-id.c
CC      ../../os/sys/process.c
CC      ../../os/sys/timer.c
LD      build/native/hello-world.native
CP      build/native/hello-world.native --> hello-world.native
  
```

Figura 19: Compilación nativa

Lo que generará un archivo `.native` que debe ser ejecutado como un programa dentro de Linux.

```

mariano.zapata@ide-iot-01:~/contiki-ng/examples/hello-world$ ./hello-world.native
[WARN: Tun6      ] Failed to open tun device (you may be lacking permission). Running without network.
[INFO: Main     ] Starting Contiki-NG-develop/v4.9-590-g38957608f-dirty
[INFO: Main     ] - Routing: RPL Lite
[INFO: Main     ] - Net: tun6
[INFO: Main     ] - MAC: nullmac
[INFO: Main     ] - 802.15.4 PANID: 0xabcd
[INFO: Main     ] - 802.15.4 Default channel: 26
[INFO: Main     ] Node ID: 1800
[INFO: Main     ] Link-layer address: 0102.0304.0506.0708
[INFO: Main     ] Tentative link-local IPv6 address: fe80::302:304:506:708
[INFO: Native   ] Added global IPv6 address fd00::302:304:506:708
Hello, world
Hello, world
  
```

Figura 20: Ejecución nativa

La conexión a la placa raspberryPi se debe realizar mediante ssh, utilizando las mismas credenciales para VCL y Guacamole.

```

mariano.zapata@ide-iot-01:~$ ssh mariano.zapata@10.0.0.10
mariano.zapata@10.0.0.10's password:
Linux raspberrypi 6.1.21-v8+ #1642 SMP PREEMPT Mon Apr  3 17:24:16
BST 2023 aarch64

The programs included with the Debian GNU/Linux system are free so
ftware;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sat Apr 20 21:39:17 2024 from 10.0.0.101
mariano.zapata@raspberrypi:~$ █

```

Figura 21: Conexión a Raspberry Pi

Nota: para realizar la carga y compilación del firmware en la placa raspberryPi se debe utilizar el siguiente comando: `make TARGET=openmote BOARD=openmote-cc2538 PORT=/dev/ttyUSB* <nombre del programa>.upload`

En este caso no es necesario el parámetro `BOARD_REVISION=REV_A1` ya que la placa utilizada utiliza una diferente versión con mayor memoria.

3.2. Ejemplo uso de sensor de temperatura

Nota: Para este ejemplo se utilizará el sensor embebido en la misma placa `openmote-cc2538`. En versiones posteriores de desarrollo del laboratorio se hará uso de conexión de sensores externos.

Dentro del directorio `/contiki-ng/arch/platform/openmote/dev` se encuentran el archivo `.c` y el archivo `.h` utilizados para los sensores embebidos en la placa, en este caso solamente se hará uso de los correspondientes al sensor `sht21`.

En el ejemplo se copian ambos a un nuevo directorio dentro del directorio `examples` y se crean los siguientes archivos:

`my_sensor_app.c`: utilizado como archivo main para el uso del sensor y el archivo `Makefile`.

```

mariano.zapata@ide-iot-01:~/contiki-ng/examples/sht21$ ls
Makefile build my sensor app.c sht21.c sht21.h
mariano.zapata@ide-iot-01:~/contiki-ng/examples/sht21$ █

```

Figura 22: Directorio sht21



Dentro del archivo *my_sensor_app* en una primera instancia se define una variable de tiempo y se le coloca de nombre *timer*, posterior a eso se inicia el proceso y se activa el sensor.

El siguiente paso es un bucle *while* en el cual cada 3 segundos se consulta al sensor el estado de la temperatura y humedad y se almacenan en una variable de tipo entero, posteriormente esas variables son recortadas a dos dígitos después del punto y se realiza la impresión por pantalla.

```

#include "contiki.h"
#include "lib/sensors.h"
#include "sht21.h"
#include <stdio.h>

PROCESS(my sensor app process, "My Sensor App Process");
AUTOSTART PROCESSES(&my sensor app process);

PROCESS_THREAD(my sensor app process, ev, data)
{
    static struct etimer timer;

    PROCESS_BEGIN();

    printf("Initializing SHT21 sensor...\n");
    SENSORS_ACTIVATE(sht21);
    while(1){
        // Espera durante 3 segundos
        etimer set(&timer, CLOCK_SECOND * 3);
        PROCESS_WAIT_EVENT_UNTIL(etimer expired(&timer));

        // Lee valores del sensor
        int temperature = sht21.value(SHT21_READ_TEMP);
        int humidity = sht21.value(SHT21_READ_RHUM);

        printf("Temperature: %d.%02d degrees Celsius\n", temperature/100, temperature % 100);
        printf("Humidity: %d.%02d percent\n", humidity/100, humidity % 100);
    }
    // Desactiva el sensor
    //SENSORS_DEACTIVATE(sht21);
  
```

Figura 23: Programa temperatura

La compilación se realiza de la misma forma que los ejemplos anteriores, y mediante el comando login se puede ver la temperatura y humedad que está detectando el sensor:

```

connecting to /dev/ttyUSB0 [OK]
Temperature: 30.03 degrees Celsius
Humidity: 98.49 percent
Temperature: 30.05 degrees Celsius
Humidity: 98.53 percent
Temperature: 30.08 degrees Celsius
Humidity: 98.56 percent
  
```

Figura 24: Impresión de temperatura por puerto serial

3.3. Ejemplo de realización de comunicación CoAP

CoAP es un protocolo de software que se encuentra en el nivel de capa de aplicación del modelo OSI y está orientado a correr en dispositivos que tienen recursos restringidos, como son los dispositivos IoT.

Permite el intercambio de mensajes asincrónicos, es decir que un mensaje puede llegar en cualquier momento. Si bien CoAP es similar en algunos casos a HTTP, la diferencia principal reside en que mientras HTTP utiliza como capa de transporte a TCP, CoAP hace uso de UDP.

CoAP utiliza una arquitectura de nodos, en el cual todos pueden intercambiar información con cualquier nodo mediante un protocolo de ruteo, con la finalidad de encontrar el nodo que funciona como Gateway (en caso de ser necesario).

Dentro de contiki-ng existen múltiples formas de programar los openmote para hacer uso de CoAP, en este caso se explicará solamente una posibilidad.

En la arquitectura de laboratorio existe además de los tres openmote conectados al entorno de desarrollo, un cuarto mote conectado a una placa raspberry PI para emular mejor un entorno real de desarrollo en campo o complejos industriales.

Nota: Para más información de CoAP y de la arquitectura IoT desarrollada, consultar el documento “*manual-arquitectura-lab-iot*”.

Como primer paso se debe realizar la conexión a la placa raspberryPI mediante una segunda terminal en Guacamole.

Para el correcto funcionamiento del sistema es necesario que un mote funcione como un router de borde, es decir que permita a la red IPv6 comunicarse con las redes IPv4, además de otras características. No es necesario realizar estos pasos en la placa raspberryPI ya que se puede programar en las placas dentro del mismo entorno de desarrollo que tiene el nombre de host “lab-iot-<número>”. Se recomienda realizarlo de esta manera porque se asemeja a un sistema en campo.

```

mariano.zapata@raspberrypi:~/contiki-ng/examples/rpl-border-router$ make TARGET=openmote BOARD=openmote-cc2538 BOARD_REVISION=REV_A1 PORT=/dev/ttyUSB0 border-router.upload
./../tools/cc2538-bsl/cc2538-bsl.py --bootloader-invert-lines -e -w -v -b 450000 -p /dev/ttyUSB0 -a 0x00202000 build/openmote/openmote-cc2538/border-router.bin
Opening port /dev/ttyUSB0, baud 450000
Reading data from build/openmote/openmote-cc2538/border-router.bin
Cannot auto-detect firmware filetype: Assuming .bin
Connecting to target...
CC2538 PG2.0: 256KB Flash, 32KB SRAM, CCFG at 0x0023FFD4
Primary IEEE Address: 00:12:4B:00:04:30:53:81
Performing mass erase
Erasing 262144 bytes starting at address 0x00200000
Erase done
Writing 253952 bytes starting at address 0x00202000
Write 248 bytes at 0x0023FF088
Write done
Verifying by comparing CRC32 calculations.
Verified (match: 0x470864ba)
mariano.zapata@raspberrypi:~/contiki-ng/examples/rpl-border-router$

```

Figura 25: Carga de firmware border-router

Debido a problemas entre la diferencia de arquitecturas entre el entorno de desarrollo y la placa raspberry, para poder visualizar el terminal serial no se debe usar la palabra login como en los ejemplos anteriores, ya que surgirán errores. Para poder utilizarlo se debe ejecutar el siguiente comando:

```
rlwrap ../../tools/serial-io/serialedump.raspi -b115200 /dev/ttyUSB0
```

rlwrap es un programa que proporciona una interfaz de línea de comandos mejorada alrededor de otros programas. El primer parámetro del programa es la ruta ejecutable al binario serialdump.raspi, se utiliza una velocidad de transmisión de 115200 baudios a través del puerto serie ttyUSB0.

Nota: en el caso de usar el mote colocado en la placa raspberryPI, este estará colocado siempre en el serial ttyUSB0.

```

mariano.zapata@raspberrypi:~/contiki-ng/examples/rpl-border-router$ rlwrap ../../tools/serial-io/serialedump.raspi -b115200 /dev/ttyUSB0
connecting to /dev/ttyUSB0 [OK]
[INFO: BR      ] Waiting for prefix
[INFO: BR      ] Waiting for prefix
[INFO: BR      ] Waiting for prefix
[INFO: BR      ] Waiting for prefix
?p
mariano.zapata@raspberrypi:~/contiki-ng/examples/rpl-border-router$ █

```

Figura 26: Visualización border-router

El siguiente paso es ejecutar el túnel SLIP para permitir la comunicación entre el openmote y el sistema operativo Linux, para ello se debe ejecutar la instrucción *make TARGET=openmote connect-router*

SLIP (serial línea internet protocol) es una técnica de encapsulamiento que se utiliza para transmitir paquetes de datos a través de una conexión serial entre dos dispositivos o sistemas que no tienen una red de datos directa entre sí. Si bien el nombre no especifica que tipos de datos se convertirán, en este caso se transforma de IPv6 a serial para posteriormente ser utilizado en redes IPv4.


```

mariano.zapata@raspberrypi:~/contiki-ng/examples/rpl-border-router$ make TARGET=openmote connect-router
make -C ../../tools/serial-io tunslip6
make[1]: Entering directory '/mnt/nfs/home/mariano.zapata/contiki-ng/tools/serial-io'
cc -Wall -Werror -O2 tunslip6.c tools-utils.c -o tunslip6
make[1]: Leaving directory '/mnt/nfs/home/mariano.zapata/contiki-ng/tools/serial-io'
sudo ../../tools/serial-io/tunslip6 fd00::1/64
*****SLIP started on `/dev/ttyUSB0'
opened tun device `/dev/tun0'
ifconfig tun0 inet `hostname` mtu 1500 up
ifconfig tun0 add fd00::1/64
ifconfig tun0 add fe80::0:0:0:1/64
ifconfig tun0

tun0: flags=4305<UP,POINTOPOINT,RUNNING,NOARP,MULTICAST> mtu 1500
inet 127.0.1.1 netmask 255.255.255.255 destination 127.0.1.1
inet6 fe80::1 prefixlen 64 scopeid 0x20<link>
inet6 fd00::1 prefixlen 64 scopeid 0x0<global>
inet6 fe80::b1b2:1416:ba9f:d835 prefixlen 64 scopeid 0x20<link>
unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00 txqueuelen 500 (UNSPEC)
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

[INFO: BR ] Waiting for prefix
*** Address:fd00::1 => fd00:0000:0000:0000
[INFO: BR ] Waiting for prefix
[INFO: BR ] Server IPv6 addresses:
[INFO: BR ] fd00::212:4b00:430:5381
[INFO: BR ] fe80::212:4b00:430:5381

```

Figura 27: Túnel SLIP

Se ve que el túnel otorga dos direcciones IPv6 que se observan al final de la información mostrada. Si bien son similares, se utilizan mediante conceptos diferentes, la IP que comienza con fe80 es para ser utilizada en enlaces locales de los motes, es decir que a esta dirección IP no se puede alcanzar desde redes externas. Mientras que la IP que comienza con fd80 es para ser utilizada en redes que comparten el mismo segmento de red, motivo por el cual es posible realizar el comando ping para recibir respuesta de dicha conexión.

El siguiente paso consiste en configurar CoAP del lado de los nodos sensores, en nuestro caso se hizo uso del sensor de temperatura para poder enviar la información cada un tiempo definido. El programa main se creará dentro del directorio *contiki-ng/examples/coap/coap-example-server* y se usará como plantilla el programa *coap-example-server.c*



```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "contiki.h"
#include "coap-engine.h"

#if PLATFORM SUPPORTS BUTTON HAL
#include "dev/button-hal.h"
#else
#include "dev/button-sensor.h"
#endif

/* Log configuration */
#include "sys/log.h"
#define LOG_MODULE "App"
#define LOG_LEVEL LOG_LEVEL_APP
/*
 * Resources to be activated need to be imported through the extern keyword.
 * The build system automatically compiles the resources in the corresponding sub-directory.
 */
extern coap_resource_t
  res hello,
  res temp sensor;

PROCESS(er example server, "Erbium Example Server");
AUTOSTART_PROCESSES(&er example server);

PROCESS_THREAD(er example server, ev, data)
{
  PROCESS_BEGIN();

```

Figura 28: Programa CoAP - Parte 1

```

PROCESS PAUSE();

LOG_INFO("Starting Erbium Example Server\n");

/*
 * Bind the resources to their Uri-Path.
 * WARNING: Activating twice only means alternate path, not two instances!
 * All static variables are the same for each URI path.
 */
coap activate resource(&res hello, "test/hello");
coap activate resource(&res temp sensor, "sensor/temperatura");
/* Define application-specific events here. */
while(1) {
  PROCESS_WAIT_EVENT();
#if PLATFORM HAS BUTTON
#if PLATFORM SUPPORTS BUTTON HAL
  if(ev == button hal release event) {
#else
  if(ev == sensors event && data == &button sensor) {
#endif
  LOG_DBG("*****BUTTON*****\n");

  /* Call the event handler for this application-specific event. */
  /* Also call the separate response example handler. */
  }
#endif /* PLATFORM HAS BUTTON */
} /* while (1) */

PROCESS_END();
}

```

Figura 29: Programa CoAP - Parte 2

Nota: Para información general de la programación consultar el anexo B.

En este código resalta el uso de la línea
coap_activate_resource(&res_temp_sensor, "sensor/temperatura");

`Coap_activate_resource()` es la función que se utiliza para activar un recurso en el servidor CoAP. `&res_temp_sensor` es la dirección de memoria del recurso que contiene la variable `'res_temp_sensor'` y debe ser una instancia válida de una estructura que representa el recurso CoAP, en otras palabras, es un puntero que apunta a la dirección de memoria en la que se encuentra el programa correspondiente a `'res_temp_sensor'` y este debe existir.

`"sensor/temperatura"` es la ruta del recurso dentro del servidor CoAP. Esto significa que estará disponible para los clientes CoAP que deseen acceder a esta ruta.

En el caso de CoAP se necesita tener una programación que pueda crear el formato de los paquetes pedidos desde el router de borde, para ello siguiendo el criterio de modularidad de contiki, se creó un nuevo archivo dentro del directorio resources. Dicho programa en el caso de este ejemplo se lo llamó `res-temp-sensor.c`

```
#include "contiki.h"
#include "lib/sensors.h"
#include <stdio.h>
#include "coap-engine.h"
#include "sht21.h"

static void temperature_resource_get_handler(coap_message_t *request, coap_message_t *response, uint8_t *buffer, uint16_t preferred_size)
static void temperature_resource_periodic_handler(void);

PERIODIC_RESOURCE(res_temp_sensor,
                  "title=\"Temperature\";rt=\"TemperatureSensor\"",
                  temperature_resource_get_handler,
                  NULL,
                  NULL,
                  NULL,
                  3 * CLOCK_SECOND, // Actualiza cada 3 segundos
                  temperature_resource_periodic_handler);

static void temperature_resource_get_handler(coap_message_t *request, coap_message_t *response, uint8_t *buffer, uint16_t preferred_size)
{
  SENSORS_ACTIVATE(sht21);
  int temperature = sht21.value(SHT21_READ_TEMP);

  // Formatea la temperatura en la cadena de respuesta
  sprintf((char *)buffer, COAP_MAX_CHUNK_SIZE, "%d.%02d", temperature / 100, temperature % 100);

  // Configura los encabezados de la respuesta CoAP
  coap_set_header_content_format(response, TEXT_PLAIN);
  coap_set_payload(response, buffer, strlen((char *)buffer));
}
```

Figura 30: Lectura de temperatura - Parte 1

```
static void temperature_resource_periodic_handler(void)
{
  // Notificar a los observadores sobre cambios en la temperatura
  coap_notify_observers(&res_temp_sensor);
}
```

Figura 31: Lectura de temperatura - Parte 2

Las funciones `'temperature_resource_get_handler'` y `'temperature_resource_periodic_handler'` se encargan de manejar las solicitudes GET del recurso de temperatura y de realizar actualizaciones periódicas del recurso, respectivamente.

En el caso de la primera función, se le pasa como parámetros punteros a la estructura `coap_message_t` para las solicitudes y respuestas del servidor CoAP. Posteriormente se pasa un puntero a un búfer de bytes donde se

colocará la carga útil de la respuesta. El siguiente parámetro indica el tamaño preferido para la carga útil de la respuesta.

En la siguiente línea se define el recurso CoAP *'res_temp_sensor'*, que representa el sensor de temperatura definido anteriormente en el programa main. Se especifica el título y el tipo y para este recurso se asigna el manejador *'temperature_resource_get_handler'* para las solicitudes GET del recurso y el manejador *'temperature_resource_periodic_handler'* para las actualizaciones periódicas. El recurso se configura para actualizarse cada 3 segundos.

Posteriormente se define el manejador GET del recurso de temperatura (función *temperatura_resource_get_handler*): se activa el sensor de temperatura y lee la temperatura actual del sensor. Después formatea esta temperatura en una cadena de respuesta y configura los encabezados de respuesta CoAP para indicar que el contenido es de tipo texto plano.

La última función que se ejecuta periódicamente se encarga de actualizar el recurso de temperatura.

Posterior a esto se debe compilar y cargar el firmware en las placas mediante los comandos anteriormente definidos.

```

connecting to /dev/ttyUSB0 [OK]
[INFO: Main      ] Starting Contiki-NG-develop/v4.9-590-g38957608f-dirty
[INFO: Main      ] - Routing: RPL Lite
[INFO: Main      ] - Net: sicslowpan
[INFO: Main      ] - MAC: CSMA
[INFO: Main      ] - 802.15.4 PANID: 0xabcd
[INFO: Main      ] - 802.15.4 Default channel: 26
[INFO: Main      ] Node ID: 21268
[INFO: Main      ] Link-layer address: 0012.4b00.0430.5314
[INFO: Main      ] Tentative link-local IPv6 address: fe80::212:4b00:430:5314
[INFO: OpenMote  ] OpenMote-CC2538
[INFO: App       ] Starting Erbium Example Server

```

Figura 32: Compilación de firmware

Para poder realizar la petición de los recursos mediante coap se hará uso del siguiente script de Python:

```

import logging
import asyncio
import influxdb client
from aiocoap import *
from influxdb import InfluxDBClient

logging.basicConfig(level=logging.INFO)

async def main():
    protocol = await Context.create_client_context()

    request = Message(code=GET, uri='coap://[fd00::212:4b00:430:5314]/sensor/temperatura')

    try:
        response = await protocol.request(request).response
    except Exception as e:
        print('Failed to fetch resource:')
        print(e)
    else:
        print('Result: %s\n%s'%(response.code, response.payload))
        temp = (response.payload.decode('utf-8'))
        print(temp)
        write to influxdb(temp)

if name == " main ":
    asyncio.run(main())

```

Figura 33: Programa Python

Se observa que se utiliza la dirección IPv6 asignada al programa de lectura de temperatura. Y se hace uso de la ruta definida en el script *main* del programa que mide la temperatura para poder obtener la misma.

Posterior a esto se debe ejecutar el programa Python:

```

mariano.zapata@raspberrypi:~$ python3 conexion.py
Result: 2.05 Content
b'30.82'
30.82
mariano.zapata@raspberrypi:~$ python3 conexion.py
Result: 2.05 Content
b'30.81'
30.81
mariano.zapata@raspberrypi:~$ █

```

Figura 34: Ejecución Python

Se observa que se obtienen los resultados de la temperatura que se está midiendo en el sensor colocado en el segundo mote.

3.4. Ejemplo de realización de comunicación MQTT

3.4.1. Breve explicación y funcionamiento de MQTT

Si bien mqtt es un protocolo que se utiliza en capa de aplicación, tiene varias diferencias frente a CoAP ya que utiliza una arquitectura diferente de cliente-broker, además de que el mismo puede correr tanto sobre TCP como UDP.

Para el uso de mqtt en openmote mediante contiki-ng se hará uso del bróker Mosquitto, el cual se encuentra instalado y ejecutándose por defecto en el puerto 1885 tanto del entorno de desarrollo como de la raspberry.

El primer paso consiste en cargar nuevamente el programa border-router en el mote que funcionará como router de borde y posteriormente “levantar” el túnel slip.



Figura 35: Logo Mosquitto

Como en este caso el bróker Mosquitto ya se encuentra disponible, no será necesario configurar el mismo.

Sin embargo, si es necesario suscribirse al tópico deseado para poder obtener los mensajes, para ello se debe abrir una nueva terminal ya sea en el entorno de desarrollo o en la placa raspberry y ejecutar el siguiente comando, *mosquitto_sub -h localhost -p 1885 -t "#"*

```
mariano.zapata@raspberrypi:~$ mosquitto sub -h localhost -p 1885 -t "#"
```

Figura 36: Suscripción MQTT

En una primera instancia no se observará la recepción de ningún mensaje, sin embargo es posible realizar la publicación desde el mismo entorno en el que se ha suscripto al servidor mosquitto o desde el otro servidor (esto sin realizar la publicación mediante las placas openmote).

En las siguientes imágenes se observa la publicación de un mensaje desde la misma placa raspberry y su posterior visualización en la pantalla en la que se ha realizado la suscripción a todos los tópicos:

```
mariano.zapata@raspberrypi:~$ mosquitto_pub -h localhost -p 1885 -t "/prueba" -m "Hola, mundo"
mariano.zapata@raspberrypi:~$
```

Figura 37: Publicación MQTT

```
mariano.zapata@raspberrypi:~$ mosquitto_sub -h localhost -p 1885 -t "#"
Hola, mundo
```

Figura 38: Resultado suscripción

Sin embargo, también es posible realizar la publicación desde el mismo entorno de desarrollo, colocando como host la dirección IPv4 de la placa raspberryPI:

```
mariano.zapata@ide-iot-01:~$ mosquito_pub -h 10.0.0.10 -p 1885 -t "/prueba" -m "Soy el entorno de desarrollo"
mariano.zapata@ide-iot-01:~$
```

Figura 39: Publicación desde entorno de desarrollo

```
mariano.zapata@raspberrypi:~$ mosquito_sub -h localhost -p 1885 -t "/"
Hola, mundo
Soy el entorno de desarrollo
```

Figura 40: Resultado suscriptor

3.4.2. Programación de ejemplo MQTT

Como se mencionó en el punto anterior, la aplicación mosquito se encuentra corriendo en la placa raspberryPi y en el entorno de desarrollo. Por lo tanto se puede realizar la comunicación MQTT utilizando solamente el entorno de desarrollo o utilizarlos combinados.

En este caso se utilizará la placa raspberryPi como el bróker MQTT y el objetivo será traer información general desde un cliente MQTT.

El primer paso consiste en cargar el programa rpl-border-router en el nodo anexo a la placa RaspberryPI:

```
mariano.zapata@raspberrypi:~/contiki-ng/examples/rpl-border-routers$ make TARGET=openmote BOARD=openmote-cc2538 PORT=/dev/ttyUSB0 border-router.upload
make: git: No such file or directory
./../tools/cc2538-bsl/cc2538-bsl.py --bootloader-invert-lines -e -w -v -b 450000 -p /dev/ttyUSB0 -a 0x00202000 build/openmote/openmote-cc2538/border
-router.bin
Opening port /dev/ttyUSB0, baud 450000
Reading data from build/openmote/openmote-cc2538/border-router.bin
Cannot auto-detect firmware filetype: Assuming .bin
Connecting to target...
CC2538 PG2.0: 512KB Flash, 32KB SRAM, CCFG at 0x0027FFD4
Primary IEEE Address: 00:12:48:00:00:00:7F:DE
Performing mass erase
Erasing 524288 bytes starting at address 0x00200000
Erase done
Writing 516096 bytes starting at address 0x00202000
Write 8 bytes at 0x0027FFF8F00
Write done
Verifying by comparing CRC32 calculations.
Verified (match: 0x44415132)
```

Figura 41: Carga border-router

El siguiente paso es activar el túnel slip:

```
[INFO: BR      ] Waiting for prefix
*** Address:fd00::1 => fd00:0000:0000:0000
[INFO: BR      ] Waiting for prefix
[INFO: BR      ] Server IPv6 addresses:
[INFO: BR      ]   fd00::212:4b00:60d:7fde
[INFO: BR      ]   fe80::212:4b00:60d:7fde
```

Figura 42: Túnel SLIP

En este punto se observan dos direcciones IP, correspondiente a IP routeables y la IP del enlace punto a punto de los motes respectivamente.

La primera de estas direcciones debe ser utilizada para especificar el bróker mosquito en el cliente MQTT. Sin embargo, en este caso no es necesario configurar la ip del bróker ya que como solamente existe un solo bróker, el

router de borde entiende que debe reenviar la información que recibe de forma inalámbrica al server mosquito. Si se debe tener en cuenta de cambiar el puerto del cliente mqtt al 1885.

```

/*-----*/
/* MQTT broker address. Ignored in Watson mode */
#ifdef MQTT_CLIENT_CONF_BROKER_IP_ADDR
#define MQTT_CLIENT_BROKER_IP_ADDR MQTT_CLIENT_CONF_BROKER_IP_ADDR
#else
#define MQTT_CLIENT_BROKER_IP_ADDR "fd00::1"
#endif
/*-----*/

```

Figura 43: Programa MQTT - Parte 1

```

/*-----*/
/* Default configuration values */
#define DEFAULT_TYPE_ID "mqtt-client"
#define DEFAULT_EVENT_TYPE_ID "status"
#define DEFAULT_SUBSCRIBE_CMD_TYPE "+"
#define DEFAULT_BROKER_PORT 1885
#define DEFAULT_PUBLISH_INTERVAL (30 * CLOCK_SECOND)
#define DEFAULT_KEEP_ALIVE_TIMER 60
#define DEFAULT_RSSI_MEAS_INTERVAL (CLOCK_SECOND * 30)
/*-----*/

```

Figura 44: Programa MQTT - Parte 2

Se hará un pequeño cambio en la información recibida en el bróker MQTT con el objetivo de poder identificar que efectivamente se está realizando la comunicación.

```

len = sprintf(buf_ptr, remaining,
              "\Def Route (soy el ttyUSB1) \": \"%s\", \"RSSI (dBm) \": %d",
              def_rt_str, def_rt_rssi);

```

Figura 45: Modificación en programa

Posterior a esto se debe cargar y compilar el programa en el mote deseado (en este caso se hizo uso del mote conectado a la interfaz serial ttyUSB1).

Mediante mqtt el cliente enviará la información del estado de la placa openmote, esto deberá verse en la pestaña en la que se realizó la suscripción al tópico "#", realizada en la placa raspberryPi.

```

bariano.zapata@raspberrypi:~$ mosquitto_sub -h localhost -p 1885 -t "#"
{"d":{"Platform":"openmote","Board":"openmote-cc2538","Seq #":1,"Uptime (sec)":666,"Def Route (soy el ttyUSB1)": "fe80::212:4b00:60d:7fde", "RSSI (dBm)": -54, "VDD3 (mV)": 2478, "On-Chip Temp (mC)": 16667}}
{"d":{"Platform":"openmote","Board":"openmote-cc2538","Seq #":2,"Uptime (sec)":696,"Def Route (soy el ttyUSB1)": "fe80::212:4b00:60d:7fde", "RSSI (dBm)": -54, "VDD3 (mV)": 2479, "On-Chip Temp (mC)": 16667}}
{"d":{"Platform":"openmote","Board":"openmote-cc2538","Seq #":3,"Uptime (sec)":726,"Def Route (soy el ttyUSB1)": "fe80::212:4b00:60d:7fde", "RSSI (dBm)": -54, "VDD3 (mV)": 2478, "On-Chip Temp (mC)": 16905}}
{"d":{"Platform":"openmote","Board":"openmote-cc2538","Seq #":4,"Uptime (sec)":756,"Def Route (soy el ttyUSB1)": "fe80::212:4b00:60d:7fde", "RSSI (dBm)": -54, "VDD3 (mV)": 2478, "On-Chip Temp (mC)": 16667}}
{"d":{"Platform":"openmote","Board":"openmote-cc2538","Seq #":5,"Uptime (sec)":786,"Def Route (soy el ttyUSB1)": "fe80::212:4b00:60d:7fde", "RSSI (dBm)": -54, "VDD3 (mV)": 2478, "On-Chip Temp (mC)": 16667}}
{"d":{"Platform":"openmote","Board":"openmote-cc2538","Seq #":6,"Uptime (sec)":816,"Def Route (soy el ttyUSB1)": "fe80::212:4b00:60d:7fde", "RSSI (dBm)": -54, "VDD3 (mV)": 2479, "On-Chip Temp (mC)": 16667}}
{"d":{"Platform":"openmote","Board":"openmote-cc2538","Seq #":7,"Uptime (sec)":846,"Def Route (soy el ttyUSB1)": "fe80::212:4b00:60d:7fde", "RSSI (dBm)": -54, "VDD3 (mV)": 2479, "On-Chip Temp (mC)": 16667}}
{"d":{"Platform":"openmote","Board":"openmote-cc2538","Seq #":8,"Uptime (sec)":876,"Def Route (soy el ttyUSB1)": "fe80::212:4b00:60d:7fde", "RSSI (dBm)": -54, "VDD3 (mV)": 2479, "On-Chip Temp (mC)": 16667}}
{"d":{"Platform":"openmote","Board":"openmote-cc2538","Seq #":9,"Uptime (sec)":906,"Def Route (soy el ttyUSB1)": "fe80::212:4b00:60d:7fde", "RSSI (dBm)": -54, "VDD3 (mV)": 2479, "On-Chip Temp (mC)": 16667}}

```

Figura 46: Resultado de suscripción

En esta puede observarse el mensaje configurado para identificar que efectivamente se está realizando la comunicación.

3.5. Sniffer de red

Un sniffer de red es un programa que hace uso de la placa de red para capturar todo el tráfico que pasa a través del mismo, como la comunicación entre los motes es inminentemente Wireless, un sniffer podría obtener prácticamente todos los paquetes que se intercambian para realizar la comunicación entre los motes.

Contiki-NG ofrece la posibilidad de usar un sniffer de red que contiene ciertas modificaciones para permitir la captura y entendimiento de protocolos IoT, este programa debe cargarse en un mote diferente (se recomienda utilizar uno de los tres disponibles en el entorno de desarrollo) y posteriormente realizar la ejecución para comenzar el “snifteo” es decir, la captura de los paquetes.

Una vez que se cancela el proceso, se generará un archivo .pcap que es un formato conocido por múltiples analizadores de protocolos, en este caso particular se usará wireshark para abrir el mismo y observar los paquetes capturados.

Para ejecutar el programa de carga de firmware se debe ejecutar el programa sniff.sh que se encuentra dentro del directorio sensniff, mediante el siguiente comando:

```

mariano.zapata@ide-iot-01:~/contiki-ng/examples/sensniff$ ls
Makefile  build  cc26x0-cc13x0  openmote  project-conf.h  sensniff.c  simplelink  zl
README.md  cc2538dk  nrf52840  pool  sensniff-mac.c  sensniff.h  sniff.sh  zoul
mariano.zapata@ide-iot-01:~/contiki-ng/examples/sensniff$ █

```

Figura 47: Directorio sensniff

```

Make TARGET=openmote BOARD=openmote-cc2538 PORT=/dev/ttyUSB*
senniff.upload

```

```

mariano.zapata@ide-iot-01:~/contiki-ng/examples/sensniff$ ./sniff.sh /dev/ttyUSB0
Opening port /dev/ttyUSB0, baud 450000
Reading data from build/openmote/openmote-cc2538/sensniff.bin
Cannot auto-detect firmware filetype: Assuming .bin
Connecting to target...
CC2538 PG2.0: 256KB Flash, 32KB SRAM, CCFG at 0x0023FFD4
Primary IEEE Address: 00:12:4B:00:04:30:53:14
Performing mass erase
Erasing 262144 bytes starting at address 0x00200000
Erase done
Writing 253952 bytes starting at address 0x00202000
Write 248 bytes at 0x0023FF080
Write done
Verifying by comparing CRC32 calculations.
Verified (match: 0x3bc8335a)
mariano.zapata@ide-iot-01:~/contiki-ng/examples/sensniff$ █

```

Figura 48: Compilación sensniff

Posteriormente para la ejecución del sniffer de red se debe mover al directorio *contiki-ng/tools/sensniff* y ejecutar el archivo *sensniff.py* mediante el comando *python3 sensniff.py -d <puerto serial del mote> -D INFO -p <nombre deseado del pcap>*

```

mariano.zapata@ide-iot-01:~/contiki-ng/tools/sensniff$ python3 sensniff.py -d /dev/ttyUSB2 -D INFO -p nuevo.pcap
Started logging
Serial port /dev/ttyUSB2 opened
FIFO /tmp/sensniff exists. Using it
Dumping PCAP to nuevo.pcap
Commands:
c: Print current RF Channel
m: Print Min RF Channel
M: Print Max RF Channel
n: Trigger new pcap header before the next frame
h,?: Print this message
<number>: Change RF channel.
q: Quit
Error decoding peripheral output: 'utf-8' codec can't decode byte 0xec in position 1: invalid continuation byte
m
User input: "m"
Min channel: 11
Read a frame of size 27
Remote end not reading
PcapDumpOutHandler: Dumped a frame of size 27 bytes
Read a frame of size 27
Remote end not reading
PcapDumpOutHandler: Dumped a frame of size 27 bytes

```

Figura 49: Resultado captura

Se abrirá una pequeña terminal en la que se podrá elegir entre algunas opciones como visualizar el canal RF que se está utilizando o imprimir en pantalla la frecuencia mínima y máxima de los canales disponibles.

En el momento que otros motes estén realizando algún tipo de intercambio de información, el mote que está funcionando como sniffer de red comenzará a almacenar paquetes en un *archivo .pcap*, cuando se cancele la captura con las teclas *ctrl+c* aparecerá un resumen de la cantidad de paquetes capturados:

```

Frame Stats:
  Captured: 54
  Non-Frame: 0
  Piped: 0
  Not Piped: 54
  Dumped to PCAP: 54
mariano.zapata@ide-iot-01:~/contiki-ng/tools/sensniff$

```

Figura 50: Paquetes capturados

```

mariano.zapata@ide-iot-01:~/contiki-ng/tools/sensniff$ ls
LICENSE README.md nuevo.pcap sensniff.pcap sensniff.py
mariano.zapata@ide-iot-01:~/contiki-ng/tools/sensniff$ █

```

Figura 51: Resultado archivo .pcap

Este *archivo .pcap* no puede ser leído en el emulador de terminal ya que no se cuenta con la interfaz gráfica de algún programa que permita leer este tipo de archivo.

Para ello se debe descargar el archivo mediante un cliente de transferencia de archivos SFTP como winscp o putty.

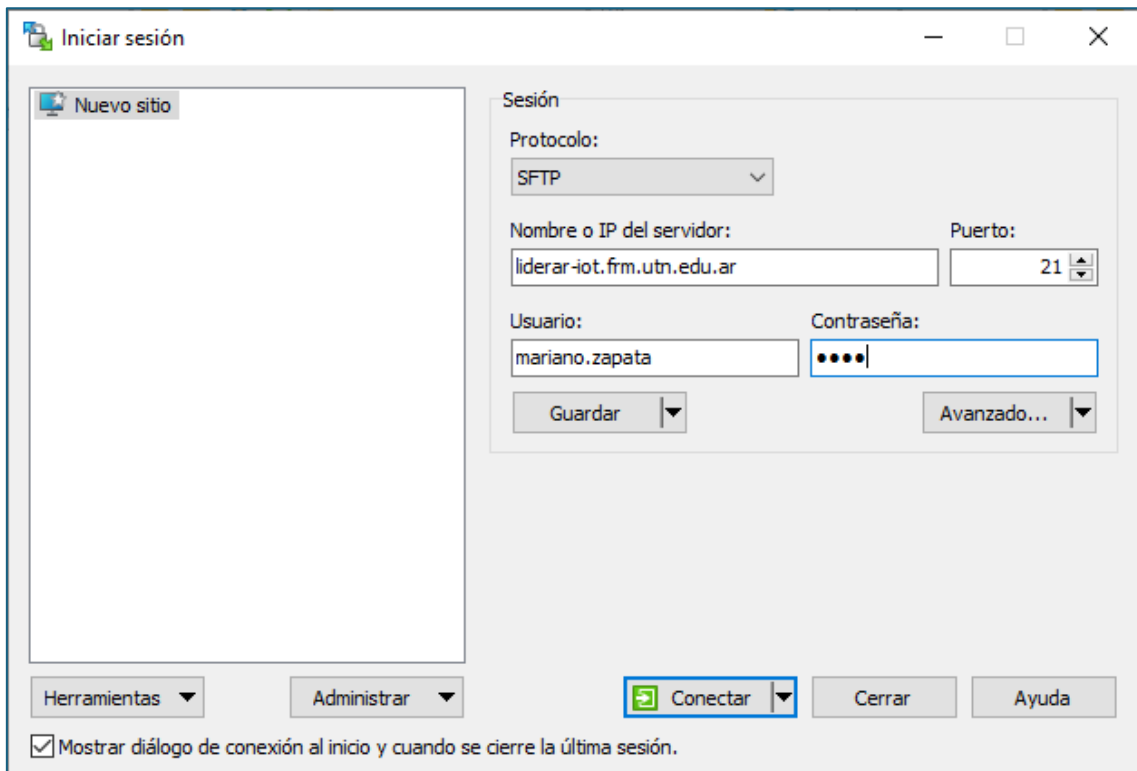
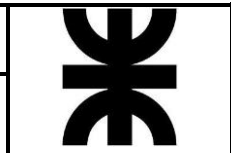


Figura 52: WinSCP






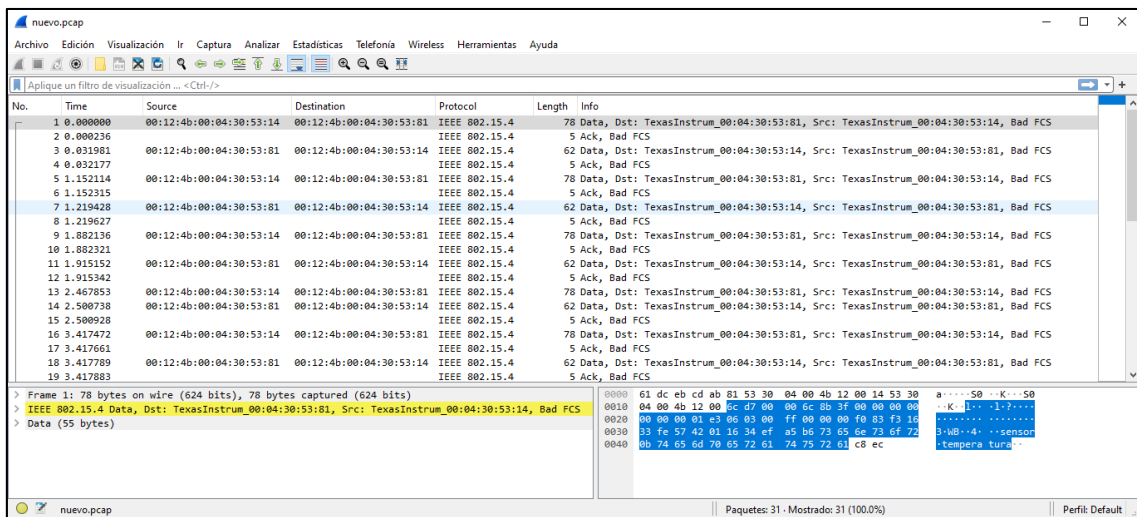
/srv/home/mariano.zapata/				
Nombre	Tamaño	Modificado	Permisos	Propiet...
		10/4/2024 11:16:03	rwxr-xr-x	root
 contiki-ng		17/4/2024 09:39:58	rwxr-xr-x	marian...
 nuevo	1 KB	22/4/2024 20:44:20	rw-r--r--	marian...

Figura 53: Directorio Usuario

Posteriormente se podrá visualizar los paquetes capturados para su posterior análisis.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	00:12:4b:00:04:30:53:14	00:12:4b:00:04:30:53:81	IEEE 802.15.4	78	Data, Dst: TexasInstrum_00:04:30:53:81, Src: TexasInstrum_00:04:30:53:14, Bad FCS
2	0.000236			IEEE 802.15.4	5	Ack, Bad FCS
3	0.031901	00:12:4b:00:04:30:53:81	00:12:4b:00:04:30:53:14	IEEE 802.15.4	62	Data, Dst: TexasInstrum_00:04:30:53:14, Src: TexasInstrum_00:04:30:53:81, Bad FCS
4	0.032177			IEEE 802.15.4	5	Ack, Bad FCS
5	1.152114	00:12:4b:00:04:30:53:14	00:12:4b:00:04:30:53:81	IEEE 802.15.4	78	Data, Dst: TexasInstrum_00:04:30:53:81, Src: TexasInstrum_00:04:30:53:14, Bad FCS
6	1.152315			IEEE 802.15.4	5	Ack, Bad FCS
7	1.219428	00:12:4b:00:04:30:53:81	00:12:4b:00:04:30:53:14	IEEE 802.15.4	62	Data, Dst: TexasInstrum_00:04:30:53:14, Src: TexasInstrum_00:04:30:53:81, Bad FCS
8	1.219627			IEEE 802.15.4	5	Ack, Bad FCS
9	1.882136	00:12:4b:00:04:30:53:14	00:12:4b:00:04:30:53:81	IEEE 802.15.4	78	Data, Dst: TexasInstrum_00:04:30:53:81, Src: TexasInstrum_00:04:30:53:14, Bad FCS
10	1.882321			IEEE 802.15.4	5	Ack, Bad FCS
11	1.915152	00:12:4b:00:04:30:53:81	00:12:4b:00:04:30:53:14	IEEE 802.15.4	62	Data, Dst: TexasInstrum_00:04:30:53:14, Src: TexasInstrum_00:04:30:53:81, Bad FCS
12	1.915342			IEEE 802.15.4	5	Ack, Bad FCS
13	2.467853	00:12:4b:00:04:30:53:14	00:12:4b:00:04:30:53:81	IEEE 802.15.4	78	Data, Dst: TexasInstrum_00:04:30:53:81, Src: TexasInstrum_00:04:30:53:14, Bad FCS
14	2.500938	00:12:4b:00:04:30:53:81	00:12:4b:00:04:30:53:14	IEEE 802.15.4	62	Data, Dst: TexasInstrum_00:04:30:53:14, Src: TexasInstrum_00:04:30:53:81, Bad FCS
15	2.500938			IEEE 802.15.4	5	Ack, Bad FCS
16	3.417472	00:12:4b:00:04:30:53:14	00:12:4b:00:04:30:53:81	IEEE 802.15.4	78	Data, Dst: TexasInstrum_00:04:30:53:81, Src: TexasInstrum_00:04:30:53:14, Bad FCS
17	3.417661			IEEE 802.15.4	5	Ack, Bad FCS
18	3.417789	00:12:4b:00:04:30:53:81	00:12:4b:00:04:30:53:14	IEEE 802.15.4	62	Data, Dst: TexasInstrum_00:04:30:53:14, Src: TexasInstrum_00:04:30:53:81, Bad FCS
19	3.417883			IEEE 802.15.4	5	Ack, Bad FCS

Frame 1: 78 bytes on wire (624 bits), 78 bytes captured (624 bits)		0000	61 dc eb cd ab 81 53 30 04 00 4b 12 00 14 53 30	a.....50...K...50
IEEE 802.15.4 Data, Dst: TexasInstrum_00:04:30:53:81, Src: TexasInstrum_00:04:30:53:14, Bad FCS		0010	04 00 4b 12 00 5c 07 00 09 62 60 5f 03 00 00 08	k.....1172777
Data (55 bytes)		0020	00 00 00 01 e3 96 03 00 ff 00 00 00 f0 83 f3 16
		0030	33 fe 57 42 01 16 34 ef a5 b6 73 65 6e 73 6f 72	3WB:4:..sensor
		0040	0b 74 65 6d 70 65 72 61 74 75 72 61 c8 ec	+tempera tura

Figura 54: Resultado captura de paquetes

Anexo A: sistema contiki-NG y organización de directorios

Contiki-ng es un sistema operativo de código abierto y una plataforma de desarrollo diseñada específicamente para aplicaciones de Internet de las cosas (IoT) y sistemas embebidos de baja potencia. Este sistema operativo está optimizado para dispositivos de baja potencia y ofrece características que ayudan a minimizar el consumo de energía, como el manejo eficiente de la radio y el “sueño profundo” de los nodos cuando no están activos.

Al ser un sistema operativo en sí mismo que puede ser utilizado y es soportado por una gran cantidad de placas IoT, *contiki-ng* cuenta con una cantidad considerable de directorios, componentes y archivos que ayudan a su correcto funcionamiento, a continuación, se explican en alto nivel cual es la funcionalidad de cada uno de ellos:

```

mariano.zapata@ide-iot-01:~/contiki-ng$ ls
CODE_OF_CONDUCT.md  Makefile.dir-variables  Makefile.identify-target  README.md  doc  tests
CONTRIBUTING.md    Makefile.embedded      Makefile.include          SECURITY.md  examples  tools
LICENSE.md          Makefile.help          Makefile.tools            arch       os
  
```

Figura 55: Contiki-NG

- **CODE_OF_CONDUCT.md:** contiene las reglas y pautas de conducta que se espera que los contribuyentes y usuarios sigan al participar en el proyecto.
- **CONTRIBUTING.md:** en este archivo se proporcionan instrucciones y directrices para aquellos que deseen contribuir al proyecto, incluyendo información sobre como informar errores, enviar parches, etc.
- **LICENSE.md:** Contiene la licencia bajo la cual se distribuye el software Contiki-NG. Por lo general, Contiki-NG utiliza licencias de código abierto como la Licencia Pública General de GNU (GPL) u otras licencias permisivas.
- **README.md:** Es un archivo de texto que proporciona información básica sobre el proyecto Contiki-NG, como una descripción general, requisitos de instalación, ejemplos de uso y enlaces útiles.
- **SECURITY.md:** Aquí se detallan las políticas y procedimientos de seguridad relacionados con el proyecto, incluyendo cómo informar sobre vulnerabilidades de seguridad de forma responsable.
- **doc:** Es un directorio que contiene documentación adicional, como guías de usuario, manuales técnicos, especificaciones de API, tutoriales, etc.

- **examples:** Contiene ejemplos de código y proyectos de demostración que muestran cómo usar las funcionalidades de Contiki-NG en aplicaciones concretas.
- **os:** En este directorio se encuentran los archivos principales del sistema operativo Contiki-NG, como los componentes del kernel, los controladores de dispositivos y otras partes fundamentales del sistema.
- **tools:** Contiene herramientas adicionales relacionadas con el desarrollo y la depuración de aplicaciones Contiki-NG, como compiladores, depuradores, scripts de automatización, etc.
- **tests:** Aquí se encuentran los archivos de pruebas y casos de prueba que se utilizan para validar y verificar el funcionamiento correcto del sistema Contiki-NG.
- **arch:** Contiene la implementación específica de arquitectura para diferentes plataformas de hardware compatibles con Contiki-NG.
- **Makefile:** Estos archivos son scripts de Makefile que se utilizan para automatizar y gestionar el proceso de compilación, configuración y distribución de los proyectos Contiki-NG.
 - **Makefile.identify-target:** Se utiliza para identificar y seleccionar la plataforma de destino (target platform) en la que se va a compilar el código. Contiene reglas y configuraciones para definir la plataforma objetivo, como el tipo de procesador, el sistema operativo subyacente, la arquitectura, etc.
 - **Makefile.include:** Contiene las inclusiones de archivos de configuración y definiciones de variables globales que se utilizan en el proceso de compilación y configuración del proyecto Contiki-NG.
 - **Makefile.tools:** Define las herramientas y utilidades necesarias para compilar y depurar el código Contiki-NG, como el compilador cruzado, el depurador, el simulador, etc. También se encarga de gestionar las dependencias de herramientas externas necesarias para el desarrollo.
 - **Makefile.dir-variables:** Define las variables de directorio utilizadas en el proyecto, como la ubicación de los directorios de código fuente, los directorios de salida para los archivos compilados, los directorios de bibliotecas, etc.
 - **Makefile.embedded:** Se utiliza para compilar y construir aplicaciones embebidas específicas para la plataforma objetivo, incluyendo la configuración de opciones de compilación, enlace de bibliotecas, generación de imágenes binarias, etc.
 - **Makefile.help:** Contiene reglas y comandos relacionados con la generación de ayuda y documentación para el proyecto Contiki-

NG, como la generación de páginas de manual, ayuda en línea, etc.

Nota: en general, los archivos Makefile que se encuentran en el directorio principal no se modifican, no así el Makefile que se encuentra dentro de los proyectos en el directorio examples.

Dentro del directorio de los proyectos existe variedad de archivos de acuerdo a la complejidad del mismo, sin embargo, todos comparten el archivo Makefile local, que en líneas generales estará compuesto por las siguientes líneas (se toma como ejemplo el *Makefile* del directorio *rpl-border-router*):

```

CONTIKI_PROJECT = border-router
all: $(CONTIKI_PROJECT)
CONTIKI = ../..

# The BR is either native or embedded, and in the latter case must support SLIP
PLATFORMS_EXCLUDE = z1

# Include RPL BR module
include $(CONTIKI)/Makefile.dir-variables
MODULES += $(CONTIKI_NG_SERVICES_DIR)/rpl-border-router
# Include webserver module
MODULES_REL += webserver

include $(CONTIKI)/Makefile.include

```

Figura 56: Archivo Makefile

- **CONTIKI_PROJECT = border-router:** Define la variable CONTIKI_PROJECT con el nombre del proyecto, en este caso, "border-router".
- **all: \$(CONTIKI_PROJECT):** Establece la regla "all" que compila el proyecto especificado en CONTIKI_PROJECT (línea superior) cuando se ejecuta el comando make.
- **CONTIKI = ../..:** Establece la variable CONTIKI con la ruta al directorio principal de Contiki-NG.
- **PLATFORMS_EXCLUDE = z1:** Excluye la plataforma "z1" del proceso de compilación. Esto significa que el proyecto no se compilará para la plataforma "z1".
- **include \$(CONTIKI)/Makefile.dir-variables:** Incluye el archivo Makefile Makefile.dir-variables ubicado en el directorio principal de Contiki-NG. Este archivo define variables de directorio utilizadas en el proyecto.
- **MODULES += \$(CONTIKI_NG_SERVICES_DIR)/rpl-border-router:** Agrega el módulo RPL Border Router al proyecto. Este módulo proporciona soporte para el Protocolo de Enrutamiento RPL en el enrutador de borde.

- **MODULES_REL += webserver:** Agrega el módulo "webserver" al proyecto. Este módulo proporciona funcionalidad de servidor web.
- **include \$(CONTIKI)/Makefile.include:** Incluye el archivo Makefile Makefile.include ubicado en el directorio principal de Contiki-NG. Este archivo contiene reglas y configuraciones estándar para compilar proyectos Contiki-NG, incluyendo opciones de compilación, enlace de bibliotecas, etc.

Nota: si bien el nombre del directorio en el que se está almacenando el proyecto no debe tener ningún nombre en especial más que el deseado por el programador, si es necesario que el nombre del archivo main del proyecto coincida con el colocado en la variable CONTIKI_PROJECT, de lo contrario el programa no compilará.



Anexo B: Protohilos de Contiki-NG

Contiki-NG utiliza lo que se conoce como protohilos o protothreads, lo que permite desarrollar aplicaciones concurrentes y eficientes con recursos limitados sin la sobrecarga de un sistema operativo de tiempo compartido completo.

El objetivo de utilizar protohilos es poder realizar multitareas de forma cooperativa, además no son preemptivos, es decir que estos no se interrumpen entre sí, sino que cooperan mediante puntos de sincronización explícitos. Algunos de estos puntos de sincronización pueden ser las instrucciones “PROCESS_WAIT_EVENT_UNTIL()” y “PROCESS_YIELD()”.

Se hará uso del programa hello-world para explicar en forma general la estructura de programación en contiki-ng usando la lógica de los protohilos.

```

#include "contiki.h"

#include <stdio.h> /* For printf() */
/*-----*/
PROCESS(hello_world_process, "Hello world process");
AUTOSTART_PROCESSES(&hello_world_process);
/*-----*/
PROCESS_THREAD(hello_world_process, ev, data)
{
    static struct etimer timer;

    PROCESS_BEGIN();

    /* Setup a periodic timer that expires after 10 seconds. */
    etimer_set(&timer, CLOCK_SECOND * 10);

    while(1) {
        printf("Hello, world\n");

        /* Wait for the periodic timer to expire and then restart the timer. */
        PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&timer));
        etimer_reset(&timer);
    }

    PROCESS_END();
}
/*-----*/
  
```

Figura 57: Programa hello-world

La línea *PROCESS(hello_world_process, "Hello world process");* se utiliza para definir un nuevo proceso llamado “hello_world_process”, y se le da una descripción de texto asociada.

La siguiente línea indica que el proceso cuyo nombre es *“hello_world_process”* se iniciará de forma automática al arrancar el sistema Contiki-NG (esto sucede en el mote en el que se carga el firmware programado).

PROCESS_THREAD(hello_world_process, ev, data): Define el hilo de ejecución del proceso *hello_world_process*, con parámetros para eventos (*ev*) y datos (*data*). Estos parámetros se encuentran en la macro *“PROCESS_THREAD”*.

En este caso no se hace uso de los parámetros *ev* y *data*, sin embargo son fundamentales en procesos más complejos donde se necesita manejar eventos y datos específicos para la lógica de la aplicación. En general los manejos de eventos y datos se codifican en bloques de código diferentes para darle mayor modularidad y orden al proyecto. Ejemplo de esto son los programas: *coap*, *mqtt-client*, *rpl-border-router*, entre otros.

static struct etimer timer;: Declara una variable estática *timer* de tipo estructura *etimer*, que se utiliza para gestionar el temporizador.

PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&timer)); Espera hasta que el temporizador *timer* expire, lo que indica que han pasado 10 segundos.

Como el proceso espera a que pase el tiempo, durante dicho tiempo se podría realizar una tarea diferente, de manera tal de no tener al procesador del dispositivo IoT consumiendo ciclos de reloj de forma ociosa.