Laboratorio 3: Capa de Aplicación

1 Introducción

En este laboratorio desarrollaremos una aplicación usando el protocolo CoAP de capa de aplicación.

Las pruebas se deberán realizar en simulación con la herramienta Cooja o en hardware utilizando las plataformas CC1350/CC2650. Se requiere al menos dos nodos para realizar las actividades.

2 Objetivos

2.1 Objetivos generales

Implementar una aplicación en la cual se van a programar nodos con servidores CoAP para que puedan recibir datos desde y hacia un PC utilizando un cliente CoAP.

2.2 Objetivos específicos

- Conocer la arquitectura cliente-servidor y el modelo REST.
- Implementar un recurso CoAP en Contiki-NG.
- Emplear el cliente *Californium Browser* (Cf-Browser) de Eclipse para interactuar con los servidores, acceder a los recursos y descubrir los recursos que un servidor expone.
- Interpretar los mensajes CoAP intercambiados entre cliente y servidor.
- Utilizar el método GET para obtener datos de los sensores de los servidores en diferentes formatos.
- Implementar la funcionalidad para que un nodo pueda llevar la hora unix y exponerla como recurso CoAP para su configuración y lectura.

3 Fundamentos

CoAP es un protocolo en capa de aplicación que utiliza UDP como capa de transporte. Por más información puede consultarse la documentación oficial sobre la implementación de CoAP en Contiki-NG, y el tutorial de CoAP para arquitectura nativa.

Para crear una aplicación que use CoAP se debe incluir dicha implementación en el Makefile. Observar que el Makefile que se encuentra en el directorio `coap-example-server` contiene la siguiente línea:

```
MODULES += $(CONTIKI_NG_APP_LAYER_DIR)/coap
```

Adicionalmente se incluye un directorio con los recursos:

```
MODULES_REL += ./resources
```

La forma recomendada para implementar recursos es crear un archivo fuente para cada recurso, ubicado en el subdirectorio `resources`, donde se puede ver varios ejemplos.

Un recurso se define utilizando una estructura `coap_resource_t` que contiene un *string* con los atributos del recurso y punteros a funciones que implementan los distintos métodos soportados por el recurso.

Un recurso se puede definir utilizando el macro:

```
RESOURCE(name, attributes, get_handler, post_handler, put_handler, delete_handler)
```

que se expande creando una estructura `name` de tipo `coap_resource_t`, y asignando el resto de los parámetros del macro a los miembros de la estructura.

Si el recurso no responde a alguno de los métodos simplemente se asigna un puntero nulo (`NULL`) a la función correspondiente en lugar del nombre de la función. Los métodos no nulos se deberán implementar a continuación. Como ejemplo, la declaración de la función que implementa el método `get` podría ser:

```
static void res_get_handler(coap_message_t *request, coap_message_t *response, \
    uint8_t *buffer, uint16_t preferred_size, int32_t *offset) { ... }
```

Finalmente, se debe crear el archivo principal donde reside el proceso asociado a la aplicación. Al inicio del proceso se debe activar los recursos previamente creados llamando a la función:

```
void coap_activate_resource(resource_t *resource, char *path);
```

donde el primer parámetro es el nombre de la estructura declarada en el archivo correspondiente al recurso y el segundo es el *string* que define la *Universal Resource Identifier* (URI) donde se va a acceder al recurso.

Observar que para poder acceder correctamente a la estructura del recurso, la misma se debe declarar como `extern` en el archivo del proceso principal.

4 Actividades

Las primeras dos actividades apuntan a familiarizarse con el software y las configuraciones necesarias para realizar un despliegue CoAP tanto en simulación (Cooja) como en hardware.

Luego, las actividades pueden realizarse tanto en simulación como en hardware, recomendándose la prueba en ambos entornos.

4.1 CoAP en simulación

1. Abrir el simulador Cooja, crear una nueva simulación y agregar (en este orden)[1] los siguientes nodos de tipo cooja:
   a. Un nodo con `examples/rpl-border-router/border-router.c`.
   b. Un nodo con `examples/coap/coap-example-server/coap-example-server.c`.
2. Iniciar un conector serie en *Tools -> Serial Socket (SERVER) -> Contiki 1…* y haciendo click en *Start*.
3. Iniciar la simulación (velocidad 1x).
4. Abrir una nueva terminal (del mismo contenedor docker) y dentro del directorio `examples/rpl-border-router` ejecutar

   ```
   make TARGET=zoul connect-router-cooja
   ```

[1] El objetivo de agregarlos en este orden es que el router de borde quede en el nodo 1, lo que permite que se le abra una conexión serie para conectar la red IP de la simulación con la red IP del contenedor contiker.

Nota: No importa realmente que el *target* sea particularmente *zoul*, sino que no sea *native*. El comando inicia un programa llamado *tunslip6* que establece una conexión entre la red IPv6 de los nodos y la red del sistema operativo (contenedor docker en este caso). El *target* no debe ser *native* porque de ser así el comando *make* hace algo distinto (compila el firmware para la arquitectura nativa y no ejecuta el software *tunslip6*).

5. En una nueva terminal, pero siempre dentro del mismo contenedor, abrir el cliente coap (ejecutar `cf-browser` en el contenedor o `contiker cf-browser` fuera del mismo), ingresar la dirección

   `coap://[fd00::202:2:2:2]:5683`

   y hacer click en "DISCOVERY" para actualizar la lista de recursos disponibles a la izquierda.

6. Observar en Cooja que cada vez que se hace POST[2] al recurso con URI `actuators/toggle`, se conmuta un LED del nodo 2. Observar que si se hace GET, PUT o DELETE a ese mismo recurso, devuelve "4.04/NOT_FOUND" ya que dichos métodos no están implementados para ese recurso.

7. Hacer GET al recurso con URI `sensors/button`. Observar que si se hace POST, PUT o DELETE a ese mismo recurso, devuelve "4.04/NOT_FOUND" ya que dichos métodos no están implementados para ese recurso.

4.2 CoAP en hardware

1. Compilar el programa `examples/coap/coap-example-server/coap-example-server.c` de la siguiente manera:

   ```
   make -E "CFLAGS += -DPLATFORM_HAS_LEDS" TARGET=cc26x0-cc13x0 \
      BOARD=launchpad/cc2650 coap-example-server
   ```

   Esto es necesario ya que para los launchpads Contiki-NG omite definir `PLATFORM_HAS_LEDS`; de otro modo el recurso `actuators/toggle` del ejemplo se excluye de la compilación. Tener presente que puede ser necesario antes ejecutar `make distclean`.

2. Grabar el programa ejemplo en un launchpad, abrir el puerto serie y anotar la dirección que reporta como "Tentative link-local IPv6 address". Luego dejarlo corriendo (no es necesario que quede conectado a una computadora, podría estar conectado a un cargador de celular).

3. Grabar el programa `examples/rpl-border-router/border-router.c` en otro launchpad y dejarlo conectado a la computadora en la que se realizarán los siguientes pasos.

4. Dentro del directorio `examples/rpl-border-router` ejecutar

   `make TARGET=zoul PORT=/dev/ttyACM0 connect-router`

   cambiando el puerto por aquel que corresponda.

5. En una nueva terminal, pero dentro del mismo contenedor en que se ejecutó el comando anterior, abrir el cliente coap (ejecutar `cf-browser` en el contenedor o `contiker cf-browser` fuera del mismo), ingresar la dirección que reportó el primer nodo pero utilizando el prefijo `fd00` propagado por el router de borde, por ejemplo:

   `coap://[fd00::212:4b00:f8e:7e86]:5683`

   y luego hacer click en "DISCOVERY" para actualizar la lista de recursos disponibles a la izquierda.

[2] CF-Browser no prevé mensajes POST sin contenido; se debe escribir cualquier cosa en el Request para que la solicitud funcione.

6. Observar en Cooja que cada vez que se hace POST al recurso con URI `actuators/toggle`, se conmuta un LED del primer nodo[3]. Observar que si se hace GET, PUT o DELETE a ese mismo recurso, devuelve "4.04/NOT_FOUND" ya que dichos métodos no están implementados para ese recurso.
7. Hacer GET al recurso con URI `sensors/button`. Observar que si se hace POST, PUT o DELETE a ese mismo recurso, devuelve "4.04/NOT_FOUND" ya que dichos métodos no están implementados para ese recurso.

4.3 Recurso: medida de tensión

Implementar un nuevo recurso para obtener una medida de la tensión de alimentación del launchpad, basándose en las actividades del laboratorio 1 en que se midió la tensión de alimentación. Se recomienda copiar la carpeta `coap-example-server` a una carpeta propia para realizar las modificaciones pedidas. También se recomienda leer algunos de los archivos que implementan recursos, ubicados bajo el directorio `coap-example-server/resources`, para tomar como base para implementar el recurso nuevo; en particular ver `res-battery.c`, `res-sht11.c` y `res-temperature.c`.

4.4 Recurso: Configurar la hora

Se desea llevar la hora (en formato unix, llamado *tiempo unix*) en los nodos, por ejemplo para agregar un *timestamp* (marca de tiempo) a los datos enviados. Para ello el nodo debe ofrecer un recurso `node/time` que soporte los métodos `PUT` y `GET`, para configurar y leer la hora, respectivamente. El módulo `clock` de Contiki-NG lleva la cuenta de los segundos transcurridos desde el último reinicio, que pueden consultarse usando la función:

```
unsigned long clock_seconds(void);
```

El tiempo unix se define como la cantidad de segundos transcurridos desde el 1º de enero del año 1970 (a la hora 0 en el meridiano de Greenwich). Para llevar el tiempo unix en el nodo se utilizará una variable `offset_segundos` para guardar la diferencia en segundos del tiempo unix con los segundos del módulo clock de Contiki-NG. Para obtener el tiempo unix en el nodo se suma el `offset_segundos` a los segundos transcurridos desde el último reinicio.

Para la implementación se pueden basar en el ejemplo `coap-example-server`.

En particular se pide:

1. Crear un proyecto con el Makefile correspondiente que incluya las aplicaciones necesarias y todos los archivos fuente.
2. Utilizar para la comunicación el canal exclusivo del grupo (#grupo+10).
3. Implementar el recurso `node/time`, con la funcionalidad descrita.
   a. Se recomienda analizar los ejemplos disponibles y la descripción de las funciones definidas en `os/net/app-layer/coap/coap-engine.h` y `os/net/app-layer/coap/coap.h` para ver cómo se obtiene y cómo se escribe el payload en los handles de los métodos GET y PUT. Más concretamente para:
      i. método GET: basarse en cualquier ejemplo que lo implemente, en particular (`res-temperature.c`).
      ii. método PUT: utilizar la función `coap_get_payload` para obtener el payload con la información requerida.

[3] CF-Browser no prevé mensajes POST sin contenido; se debe escribir cualquier cosa en el Request para que la solicitud funcione.

   b. Para la conversión de `string` a `long int` se recomienda el uso de la función `atol` o `strtol` de la biblioteca `<stdlib.h>`.

4. Para verificar el correcto funcionamiento, configurar la hora utilizando el cliente Californium (`cf-browser`) y posteriormente leerla.
5. Incorporar un *timestamp*, a la medida de la tensión de alimentación. Para ello, modificar el recurso asociado a ese sensor (el formato es libre, a definir por el grupo).
6. Observar los mensajes intercambiados entre el cliente y el servidor identificando al menos el tipo (confiable, no confiable, etc), el código (clase y detalle, por ejemplo: petición, respuesta exitosa, etc.), Message ID y token.

4.5 Enviar la hora Unix al servidor CoAP

Para realizar esta tarea se puede usar el propio Californium para escribir directamente el recurso, u opcionalmente se puede automatizar con el siguiente procedimiento en una terminal Bash:

1. La hora Unix se puede obtener desde un terminal Bash con el comando:

   ```
   date +%s
   ```

   Esto da como salida una secuencia de caracteres con la hora, que vamos a poner en el payload de un mensaje PUT de CoAP.
2. Para enviar el PUT desde la consola Linux mediante un comando Bash usaremos la biblioteca `libcoap` disponible en la imagen Docker utilizada en el curso.
3. Experimentar con el comando `coap-client` para cambiar la hora a un valor dado, por ejemplo ejecutando:

   ```
   coap-client -N -m put coap://[fd00::212:7402:2:202]:5683/node/time -e 1627874580
   ```
4. Para mandar la hora unix se debe ejecutar el siguiente comando desde consola:

   ```
   date +%s | coap-client -N -m put coap://[fd00::212:7402:2:202]:5683/node/time -f -
   ```

   sustituyendo la dirección IP del nodo y la URI del recurso correspondiente.

5 Entregables

Se deberá entregar un archivo comprimido que contenga:

1. Los archivos modificados, en particular el Makefile y el `coap-example-server`.
2. Captura de pantalla donde aparezca el *timestamp* implementado evidenciando que funciona correctamente.

Las entregas se realizarán a través de la plataforma EVA del curso.

6 Referencias

- Programming Contiki-NG: CoAP
- Tutorial CoAP
- Colina, Antonio Linan, Alvaro Vives, Antoine Bagula, Marco Zennaro, y Ermanno Pietrosemoli. 2016. IoT in five Days. E-Book. https://github.com/marcozennaro/IPv6-WSN-book/releases/.
- libcoap: C-Implementation of CoAP Files

Los materiales de este curso fueron parcialmente financiados por:

Co-funded by the
Erasmus+ Programme
of the European Union