Laboratorio 3: Capa de Aplicación

Redes de Sensores Inalámbricos
IIE, Facultad de Ingeniería, UDELAR

Tabla de Contenidos.

1 Introducción

En este laboratorio desarrollaremos una aplicación usando la capa de aplicación CoAP.
Las pruebas se podrán realizar en simulación con la herramienta Cooja o en hardware utilizando la plataforma RE-Mote. Se requieren al menos dos nodos para realizar las actividades.
Se aceptarán entregas tanto en hardware (RE-Mote) como en simulación (z1).

2 Objetivo

Se implementará 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.

2.1 Objetivos de aprendizaje

- Conocer la arquitectura cliente-servidor y el modelo REST.

- Implementar un recurso CoAP en Contiki.
- Emplear el plugin Copper de Firefox para interactuar con los servidores y 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, incluyendo la opción de observable.
- Implementar la funcionalidad para que un nodo pueda llevar la hora unix y exponerla como recurso CoAP para su configuración y lectura.

3 Lecturas y actividades previas

Nota: para realizar las actividades en simulación tener en cuenta las consideraciones señaladas en el Laboratorio 1.

Actividad previa 1

Familiarizarse con la implementación de CoAP en Contiki, llamada `erbium`, teniendo como referencia los pasos indicados en [1], y siguiendo los pasos siguientes para verificar el correcto funcionamiento:

1. Abrir el simulador Cooja y cargar el archivo de simulación suministrado: `server-only-z1.csc`
2. En caso de dar un error de compilación en relación al tamaño de buffer `REST_MAX_CHUNK_SIZE`, cambiar la siguiente línea del `project-conf.h` que está dentro del directorio `examples/er-rest-example/`:

   ```
   #define REST_MAX_CHUNK_SIZE          16
   ```
3. Abrir una nueva terminal y dentro del directorio `examples/ipv6/rpl-border-router` hacer `make connect-router-cooja`
4. Abrir el navegador Firefox (que tiene instalado el plugin Copper para CoAP) y entrar a la dirección `coap://[fd00::c30c:0:0:2]:5683/` (asociada al nodo 2)
5. Hacer click en “Discover” y se actualizará la lista de recursos disponibles a la izquierda
6. Observar en Cooja que cada vez que se hace POST 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 “4.05 Method Not Allowed” ya que dichos métodos no están implementados para ese recurso
7. Hacer GET al recurso con URI `sensors/battery`. Observar que si se hace POST, PUT o DELETE a ese mismo recurso, devuelve “4.05 Method Not Allowed” ya que dichos métodos no están implementados para ese recurso

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

Actividad previa 2

Leer la sección 5.2 de [2], y la siguiente guía breve sobre la implementación de `erbium` para facilitar su comprensión y posterior realización de la actividad.

Guía de implementación de CoAP en Contiki

La implementación de CoAP en Contiki, llamada `erbium`, está implementada como una aplicación de Contiki, es decir los archivos fuentes están en la carpeta de aplicaciones (`app/er-coap`). Esta aplicación a su vez se basa en una aplicación que implementa un motor de REST (`rest-engine`).
Está estructurada en muchos módulos, siendo los principales: *`er-coap`* y *`er-coap-engine`*.

Para crear una aplicación que use CoAP se deben incluir dichas aplicaciones en el `Makefile`. Observar en el `Makefile` que se encuentra en el directorio `er-rest-example` las siguientes líneas:

```
APPS += er-coap
APPS += rest-engine
```

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

Un recurso se define utilizando una estructura `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 `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(void *request, void *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 deben activar los recursos previamente creados llamando a la función:

```
void rest_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 url 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 Procedimientos y Tareas

4.1 Tarea 1: Configurar la hora

Se desea llevar la hora (en formato unix, llamado tiempo unix) en los nodos, por ejemplo para agregar un `timestamp` a los datos enviados. Para ello el nodo debe ofrecer un recurso "`node/time`" que soporte los métodos `put` y `get`, para setear y leer la hora, respectivamente. El módulo `clock` de Contiki lleva la cuenta de los segundos transcurridos desde el último reset, 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. Para obtener tiempo unix en el nodo se suma el `offset_segundos` a los segundos transcurridos desde el reset.

Para la implementación se pueden basar en el ejemplo `er-rest-example`.
En particular se pide:

1. Crear una 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)[1].
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 `rest-engine.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 `get_request_payload` para obtener el payload con la información requerida.
    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 plugin Copper de Firefox 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.

4.2 Tarea 2: Enviar la hora Unix al servidor CoAP.

Para realizar ésta tarea se puede usar el propio Copper 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 [4]).

[1] Este punto no es relevante en la edición de 2021, al no compartir el espacio de laboratorio con otros grupos.

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 `er-rest-example`.
2. Captura de pantalla donde aparezca el time-stamp implementado evidenciando que funciona correctamente.

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

6 Referencias

[1] https://github.com/contiki-os/contiki/tree/master/examples/er-rest-example
[2] A. L. Colina, A. Vives, A. Bagula, M. Zennaro, and E. Pietrosemoli, IoT in five Days. EBook, Jun. 2016, rev 1.1. [Online]. Available: https://github.com/marcozennaro/IPv6-WSN-book/blob/master/Releases
[3] Crontab. https://help.ubuntu.com/community/CronHowto
[4] https://libcoap.net/install.html
[5] http://sourceforge.net/projects/libcoap/files/