Práctica 3 - CoAP y Thread

Autor: Miguel Pérez García

Introducción

Para esta práctica, se tuvo que implementar un sistema de riego, en este sistema existen tres componentes: un controlador de aspersores que se comporta de igual manera como un gateway, un hub de sensores y una computadora. El hub de sensores reporta sus lecturas al controlador de aspersores, de modo que si se encuentra en modo automático y las lecturas indican que están sobre los límites considerados para encender o apagar, esta acción se debe realizar. La computadora a su vez funge como centro de control para el control de aspersores, desde este equipo se puede configurar y seleccionar los distintos modos de operación o cambiar los limites.

Código implementado

Como base del proyecto se utilizó

Para la parte del controlador de aspersores se implementaron las siguientes funciones además del código de configuración de CoAP.

```
// Aquí inicia lo necesario para configurar CoAP con las rutas adicionales.
#define APP_MODE_SET_PATH                       "/mode"
#define APP_IS_ON_PATH                          "/toggle"
#define APP_CONFIG_PATH                         "/config"
#define APP_HUMIDITY_PATH                       "/humidity"
#define APP_SensorREPORT_PATH                   "/report"
#define APP_STATUS_PATH                         "/status"


static void APP_CoapModeSet(coapSessionStatus_t sessionStatus, void *pData,
coapSession_t *pSession, uint32_t dataLen);
static void APP_CoapIsOn(coapSessionStatus_t sessionStatus, void *pData,
coapSession_t *pSession, uint32_t dataLen);
static void APP_CoapConfig(coapSessionStatus_t sessionStatus, void *pData,
coapSession_t *pSession, uint32_t dataLen);
static void APP_CoapHumidity(coapSessionStatus_t sessionStatus, void *pData,
coapSession_t *pSession, uint32_t dataLen);
static void APP_CoapSensorReport(coapSessionStatus_t sessionStatus, void *pData,
coapSession_t *pSession, uint32_t dataLen);
static void APP_CoapStatus(coapSessionStatus_t sessionStatus, void *pData,
coapSession_t *pSession, uint32_t dataLen);


const coapUriPath_t gAPP_MODE_SET_PATH = {SizeOfString(APP_MODE_SET_PATH),
(uint8_t *)APP_MODE_SET_PATH};
const coapUriPath_t gAPP_IS_ON_PATH    = {SizeOfString(APP_IS_ON_PATH), (uint8_t
*)APP_IS_ON_PATH};
const coapUriPath_t gAPP_CONFIG_PATH   = {SizeOfString(APP_CONFIG_PATH), (uint8_t
```

```
*)APP_CONFIG_PATH};
const coapUriPath_t gAPP_HUMIDITY_PATH = {SizeOfString(APP_HUMIDITY_PATH),
(uint8_t *)APP_HUMIDITY_PATH};
const coapUriPath_t gAPP_SENSORREPORT_PATH = {SizeOfString(APP_SensorREPORT_PATH),
(uint8_t *)APP_SensorREPORT_PATH};
const coapUriPath_t gAPP_STATUS_PATH = {SizeOfString(APP_STATUS_PATH), (uint8_t
*)APP_STATUS_PATH};

// A partir de aquí es la implementación que se hizo para las nuevas funcione
// Se utilizan variables globales para los distintos estados y configuraciones
static uint32_t config_temp = 0;
static uint32_t config_humidity = 0;
static bool_t isOn = false;
static bool_t isAuto = false;
// En esta función se reciben los reportes de los sensores
static void APP_CoapSensorReport
(
    coapSessionStatus_t sessionStatus,
    void *pData,
    coapSession_t *pSession,
    uint32_t dataLen
){
    // Se recibe el mensaje POST con los datos, pero solo se procesa si está en
modo automático
    if(gCoapPOST_c == pSession->code && NULL != pData && isAuto){
        uint8_t *data = (uint8_t *)pData;
        uint8_t humidity = data[0];
        uint8_t temp = data[1];
        if(humidity < config_humidity &&
           temp < config_temp &&
           !isOn){
            isOn = true;
            shell_write("Sprinkles are on");
        } else if(humidity > config_humidity &&
                  temp > config_temp &&
                  isOn)
        {
            isOn = false;
            shell_write("Sprinkles off");
        }
    }

    // Se regresa un Ack independientemente de si se proceso o no el mensaje
    if(gCoapConfirmable_c == pSession->msgType)
    {
        /* Send CoAP ACK */
        COAP_Send(pSession, gCoapMsgTypeAckSuccessChanged_c, NULL, 0);
    }
    shell_refresh();
}

// Con este método se obtiene el estado actual de los aspersores
static void APP_CoapStatus
(
```

```
    coapSessionStatus_t sessionStatus,
    void *pData,
    coapSession_t *pSession,
    uint32_t dataLen
){
    // Solo comprobamos que esté esperando respuesta y que el método por el que
llegó sea un GET
    if(gCoapConfirmable_c == pSession->msgType && gCoapGET_c == pSession->code)
    {
        uint8_t value[1];
        value[0] = isOn;
        /* Send CoAP ACK */
        COAP_Send(pSession, gCoapMsgTypeAckSuccessChanged_c, value, 1);
    }
}

// Esta función se utiliza para cambiar el modo a automático o a manual.
static void APP_CoapModeSet
(
    coapSessionStatus_t sessionStatus,
    void *pData,
    coapSession_t *pSession,
    uint32_t dataLen
){
    // El método debe ser POST
    if(gCoapPOST_c == pSession->code){
        // Comprobamos cual de los dos modos fue seleccionado
        // Si fue automático
        if(FLib_MemCmp(pData, "AUTO", 4)){
            // hacemos el cambio
            isAuto = true;
            shell_write("Mode is AUTO\n\r");
        // al igual que si fue manual
        } else if(FLib_MemCmp(pData, "MANUAL", 6)) {
            isAuto = false;
            shell_write("Mode is MANUAL\n\r");
        }
    }

    // Enviamos un ACK solo para tranquilidad del usuario
    if(gCoapConfirmable_c == pSession->msgType)
    {
        /* Send CoAP ACK */
        COAP_Send(pSession, gCoapMsgTypeAckSuccessChanged_c, NULL, 0);
    }
    shell_refresh();
}

// Esta función permite encender y apagar los aspersores
static void APP_CoapIsOn
(
    coapSessionStatus_t sessionStatus,
    void *pData,
    coapSession_t *pSession,
```

```
    uint32_t dataLen
){
    // El método debe ser POST y además estar en modo manual
    if(gCoapPOST_c == pSession->code && !isAuto){
        uint8_t* data = (uint8_t *)pData;
        // En caso de que el byte sea '0', apagamos aspersores
        if(pData != NULL && 0 == data[0] ){
            isOn = false;
            shell_write("TURN OFF\n\r");
        // En caso de que el byte sea '1', encendemos aspersores
        } else if(pData != NULL && 1 == data[0]) {
            isOn = true;
            shell_write("TURN ON\n\r");
        }
    }

    // Confirmamos de recibido con un ACK
    if(gCoapConfirmable_c == pSession->msgType)
    {
        /* Send CoAP ACK */
        COAP_Send(pSession, gCoapMsgTypeAckSuccessChanged_c, NULL, 0);
    }
    shell_refresh();
}

//Este método permite cambiar la configuración del límite de humedad y de
temperatura
static void APP_CoapConfig
(
    coapSessionStatus_t sessionStatus,
    void *pData,
    coapSession_t *pSession,
    uint32_t dataLen
){
    // Se recibe por el método POST
    if(gCoapPOST_c == pSession->code && NULL != pData){
        uint8_t* data = (uint8_t *)pData;
        // Se toman los datos
        config_temp = (uint32_t)data[0];
        config_humidity = (uint32_t)data[1];
        shell_printf("Temp %d, Humidity %d\n\r", config_temp, config_humidity);
    }

    // Se confirma de recibido con un ACK.
    if(gCoapConfirmable_c == pSession->msgType)
    {
        /* Send CoAP ACK */
        COAP_Send(pSession, gCoapMsgTypeAckSuccessChanged_c, NULL, 0);
    }
    shell_refresh();
}
```

Para la parte del hub de sensores, además de las funciones adicionales para la humedad y para configurar el periodo, se creó un timer para poder enviar el reporte de los sensores cada cierto tiempo, ese código será omitido puesto que es configuración de FreeRTOS y no de CoAP.

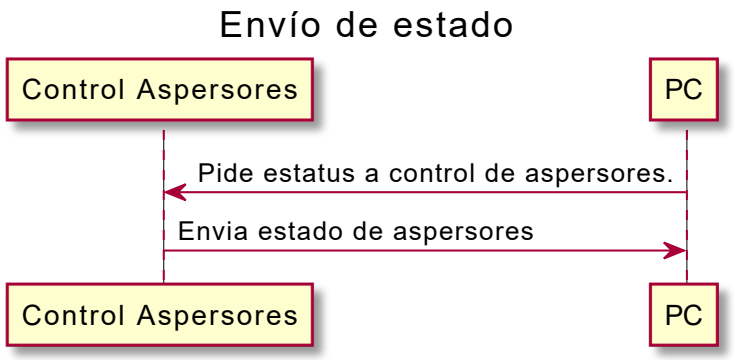
```
// Esto es la parte de la configuración de CoAP. Se omite cuando se crea el
arreglo de estructuras.
#define APP_HUNIDITY_PATH                       "/humidity"
#define APP_PERIOD_CONFIG_PATH                  "/period_config"

static void APP_CoapHumidity(coapSessionStatus_t sessionStatus, void *pData,
coapSession_t *pSession, uint32_t dataLen);
static void APP_CoapPeriodConfig(coapSessionStatus_t sessionStatus, void *pData,
coapSession_t *pSession, uint32_t dataLen);

const coapUriPath_t gAPP_HUMIDITY_PATH = {SizeOfString(APP_HUNIDITY_PATH),
(uint8_t *)APP_HUNIDITY_PATH};
const coapUriPath_t gAPP_PERIOD_CONFIG_PATH =
{SizeOfString(APP_PERIOD_CONFIG_PATH), (uint8_t *)APP_PERIOD_CONFIG_PATH};

// En esta función se procesa el cambio de periodo del timer
static void APP_CoapPeriodConfig
(
    coapSessionStatus_t sessionStatus,
    void *pData,
    coapSession_t *pSession,
    uint32_t dataLen
) {
    if(gCoapPOST_c == pSession->code && NULL != pData){
        // Se obtiene el número que está encodeado en el payload
        period = atoi((uint8_t *)pData);
        // Eliminamos el timer anterior
        xTimerDelete(xTimerHandle, 0);
        // Creamos un timer nuevo con el periodo modificado.
        xTimerHandle = xTimerCreate("timer", period / portTICK_PERIOD_MS, pdTRUE,
(void *)0, vTimerCallback);
    }

    // Confirmamos con un ACK al cliente
    if(gCoapConfirmable_c == pSession->msgType && gCoapGET_c == pSession->code)
    {
        COAP_Send(pSession, gCoapMsgTypeAckSuccessChanged_c, NULL, 0);
    }
}

bool lastWasIncrement = false;
uint8_t steps[7] = { 10, 5, 20, 15, 8, 3, 2 };
uint8_t i = 0;
uint32_t humidity = 50;
#define MAX_INDEX 7
// Por alguna razón random no estaba funcionando, entonces realicé un poors-man
random.
int32_t get_simulated_sensor()
```

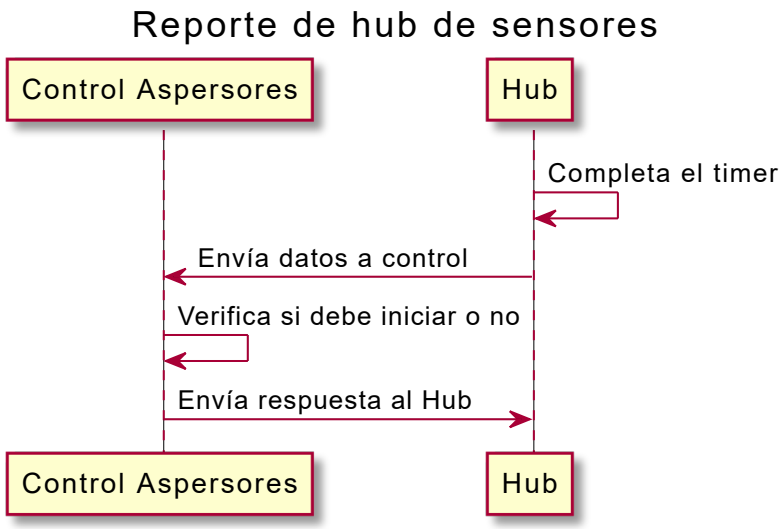
```
{
    if (MAX_INDEX == i) i = 0;
    uint8_t step = steps[i];
    uint32_t current_value = humidity;
    if(lastWasIncrement) current_value -= step;
    else current_value += step;
    i++;
    lastWasIncrement = !lastWasIncrement;

    return current_value;
}

// En esta se obtiene la humedad del sensor
static void APP_CoapHumidity
(
    coapSessionStatus_t sessionStatus,
    void *pData,
    coapSession_t *pSession,
    uint32_t dataLen
){
    uint8_t message[3];
    // Se busca que el método sea GET
    if(gCoapGET_c == pSession->code){
        // obtenemos la humedad y la encodeamos al string
        humidity = get_simulated_sensor();
        sprintf(message, "%d", humidity);
    }

    //Enviamos el ACK y con ello el payload
    if(gCoapConfirmable_c == pSession->msgType && gCoapGET_c == pSession->code)
    {
        COAP_Send(pSession, gCoapMsgTypeAckSuccessChanged_c, message, 3);
    }
}

// Este es el callback del timer
void vTimerCallback( TimerHandle_t xTimer ){
    // obtenemos la información de los sensores
    uint8_t temp = (uint8_t)APP_GetCurrentTempValue();
    humidity = get_simulated_sensor();

    // Creamos una sesión de CoAP
    coapSession_t *pSession = COAP_OpenSession(mAppCoapInstId);

    if(NULL != pSession)
    {
        coapMsgTypesAndCodes_t coapMessageType = gCoapMsgTypeNonPost_c;
        // Indicamos que no requerimos procesar la respuesta
        pSession->pCallback = NULL;
        // Copiamos la dirección de destino de la configuración a la sesión
        FLib_MemCpy(&pSession->remoteAddr, &gCoapDestAddress, sizeof(ipAddr_t));
        // Asignamos el path a la sesión
        COAP_SetUriPath(pSession, (coapUriPath_t *)&gAPP_TEMP_URI_PATH);
```

```
            // Si fuera multicast, cambiamos el tipo de mensaje
            // y asignamos un callback a la sesión
            if(!IP6_IsMulticastAddr(&gCoapDestAddress))
            {
                coapMessageType = gCoapMsgTypeConPost_c;
                pSession->pCallback = APP_CoapGenericCallback;
            }

            // creamos el buffer y asignamos los valores
            uint8_t data[2];
            data[0] = temp;
            data[1] = humidity;
            // Enviamos la información al control de aspersores
            COAP_Send(pSession, coapMessageType, data, 2);
        }
    }
```

La aplicación de escritorio fue escrita usando C# y la plataforma de UWP, con la librería de CoAP, el código puede encontrarse en el siguiente repositorio.

Diagramas





Problemas

Principalmente mi problema fue que la función `random` me daba un hardfault siempre que la intentaba usar, mi solución mas eficiente fue hacer una función que simulara la variabilidad de la humedad sin ser un random completamente y sin ser demasiado sofisticada.

Conclusiones

Me parece verdaderamente interesante el funcionamiento de Thread, si bien anteriormente en clases de redes había entendido que pues TCP y UDP son simples protocolos de transporte, nunca había visto una implementación distinta de los stacks de comunicación a este nivel, sin lugar a duda me deja intrigado sobre que tan complejos pueden llegar a ser a veces este tipo de redes que no están basada en infraestructura. Sin lugar a duda las redes mesh son bastante poderosas para las aplicaciones que se han presentado en la clase y en este caso, en la práctica, solo faltaría añadir un pequeño mDNS para poder hacer mas efectiva la comunicación con el gateway principal.