# Xarxes

Pràctica 3: La Capa d'enllaç

Blai Ras i José Peña

## 1. ***Objetivos***

Después de conocer nuestra mota y su envío de tramas, en esta práctica se nos pide implementar un algoritmo de capa de enlace. Concretamente nuestro objetivo será establecer un nodo central que gestionará una serie de nodos secundarios para realizar una serie de acciones, como su conexión y desconexión, el consumo energético de ellos y el uso de su sensor de temperatura.

En este algoritmo de capa de enlace las conexiones son punto a punto, tal y como hemos visto en clase. Nuestro grupo es uno de estos nodos secundarios, es decir, nuestro objetivo es comunicarnos con el nodo central junto con otros nodos secundarios siguiendo una tipología de estrella. En el apartado de procedimiento veremos cómo.

## 2. ***Material***

a. La mota
b. Code Composer
c. Connector JTAG
d. Programador MSP-FET430UIF
e. Sniffer

## 3. ***Procedimiento***

El algoritmo que se nos pide pasa por los siguientes puntos:

a. Conexión

En primer lugar debemos indicar al nodo central que queremos entrar en su red. Para hacerlo, este envía durante un minuto tramas de *broadcast*, es decir, tramas en que se indica su *status* de coordinador. Nuestro grupo llegó a la conclusión que este tipo de tramas deben ir precedidas de FF (hexadecimal) en la primera posición de su campo *payload*.

Para comunicar a dicho coordinador que queríamos formar parte de su red debemos enviar una trama a este indicando nuestra dirección, de manera que podrá "guardar-nos" como nodo secundario junto a los otros grupos de la clase. Este tipo de trama debe tener en la primera posición de su campo *payload* el número AB en hexadecimal, y en su campo de dirección el número 0x999, es decir, nuestra dirección.

Si el envío de esta trama era correcto, a continuación recibíamos una respuesta por parte del coordinador, un *ACK* que identificábamos gracias al número CC en la primera posición de su campo *payload*. De esta forma, nuestra entrada a su red sabíamos que había sido satisfactoria.

b. Dormir

Una vez formábamos parte de la conexión el enunciado nos pedía implementar la función de "dormir", es decir, estar en un estado inactivo hasta que el coordinador lo indicaba.

El estado de "dormir" lo entendíamos como "desconectar el *transceiver*" i pasar a un nivel de consumo mínimo. La primera acción se llevaba a cabo con la instrucción (ya implementada) *basicRfReceiveOff()*, mientras que el paso a un nivel de consumo inferior se realizaba con el *set* de instrucciones *LPM (Lower Power Mode)*. Concretamente, usábamos las instrucciones *LPM3* (nivel 3) y *LPM3_EXIT* para salir de dicho modo.

Sin embargo, a pesar de estar en un estado de "dormir" debemos comprobar si nuestro coordinador nos envía algún tipo de instrucción, como de despertar. Para hacerlo, se nos pide un *Duty Cicle* del 10%, es decir, estar 9 segundos dormidos y 1 despierto. Esto requiere realizar una interrupción cada segundo. Tal y como se indica en el enunciado, hemos modificado el temporizador interno de la mota para que cuente hasta 1 segundo y su rutina de interrupción.

En esta rutina debemos comprobar dos cosas. La primera, si estamos dormidos o no, ya que salta cada segundo estemos en el estado que estemos. En consecuencia, tenemos un booleano llamado *dd* que se activa al recibir una trama de dormir, la cual viene con el número DD en su campo *payload*. La segunda comprobación es necesaria para saber cuántos segundos han transcurrido. De esta forma, sabemos que si se trata del "segundo 10" hemos de "despertarnos". Para hacerlo, hemos declarado un *int* global llamado *count*. Este, se incrementa cada vez que entramos en la rutina.

Resumiendo, al entrar en la interrupción miramos si este booleano está en *True* o *False*. Si está "activado", debemos mirar la variable *count.* Si es la décima vez que entramos, nos despertamos. Para hacerlo, tenemos la instrucción *LPM3_EXIT* y a continuación reactivamos el *transceiver* con *basicRfReceiveOn();*, en este orden. Por último, *reseteamos* la variable *count* a cero.

c. Temperatura

El algoritmo finalmente dice que el coordinador despierta a sus nodos para realizar una petición de su temperatura. Esta instrucción viene precedida con el número DD en la primera posición de su campo *payload*. Para diferenciar si tenemos que dormirnos o despertarnos, miramos si actualmente estamos despiertos o dormidos. Sin embargo, al realizarse el envío de la trama de "despertar" nuestro nodo puede que esté dormido, es decir, que vaya por el segundo 2 del *Duty Cicle*. En consecuencia, debemos tener en cuenta que se puede tardar un poco en despertar.

Una vez despiertos los nodos secundarios deben hacer un envío de su temperatura. Para hacerlo, contamos con nuestro sensor de temperatura. La gestión de este se realiza con la librería *sht11.* Concretamente, de esta librería usamos la instrucción *sht11_temp();*, la cual lee el registro de temperatura de nuestra mota. Guardamos dicho número en una variable entera de 16 bits, pero antes de ser enviada debe ser transformada a grados Celsius. Su conversión se realiza multiplicando por 0,01 (dividiendo por 100) con *D2_14BIT* y sumándole al resultado 39,6 con *D1_CEL30.*

Una vez la temperatura es la correcta, decidimos guardarla en la segunda posición del campo *payload*, ya que en el primero poníamos el número hexadecimal BB, el cual indica que se trata de una trama "de temperatura". Se realiza el envío de la trama al coordinador y a continuación "apagamos" el sensor con la instrucción *sht11_off();*.

**Atención:** para realizar la conexión establecimos una condición que miraba si habíamos recibido "un CC". De así ser, dejábamos de enviar tramas de incorporación a la red (tramas "AB"). Sin embargo, para que todo funcionara y para realizar una comprobación más ágil de todas las funciones del algoritmo, decidimos realizar un *while* con una condición que siempre se cumple en el cual enviamos una única trama "AB".

Para realizar las funciones de "dormir" y "temperatura", decidimos realizar un *switch*. La condición de este era *rfSettings.pRxInfo->pPayload[0]*, es decir, miraba qué había en la primera posición del campo *payload* que habíamos recibido. No obstante, dicha comprobación no se hacía en tiempo real y por lo tanto no siempre funcionaba (no entraba en el *switch*). En consecuencia, la comprobación de la función de temperatura la hicimos sin dicho *switch* (y sí que funcionaba).

## 4. ***Resultados***

La conexión con el coordinador se realizaba de manera satisfactoria, es decir, nosotros entrábamos en su red tras su petición y ellos sabían nuestra dirección tras realizar la conexión. Sin embargo, tal y como hemos mencionado antes, la solicitud de formar parte de su red solo se hace una sola vez.
La función de dormir por desgracia no se llegó a comprobar, pero está implementada y revisada por nuestro profesor.
La función de temperatura sí que se realiza tal y como se pide, es decir, tras recibir una solicitud por parte del coordinador (trama BB) se hace una lectura de nuestro sensor y se envía a dicho coordinador. El valor que experimentamos en clase fue de 29 grados Celsius.
Por desgracia, no disponemos de una lectura por parte del *sniffer* de este algoritmo.

## 5. ***Conclusiones***

Mediante la implementación de este algoritmo hemos podido ver "con nuestros ojos" cómo funciona una conexión de tipo estrella mediante el protocolo IEEE 802.15.4
Concretamente, hemos visto cómo funciona en sí un protocolo, definiendo nosotros mismos cómo debe ser el campo *payload* para identificar las distintas tramas y en consecuencia realizar una u otra acción.
Nos hubiera gustado arreglar la condición de nuestro *switch* para comprobar a tiempo real qué está enviando el coordinador (es decir, no usar *rfSettings.pRxInfo->pPayload[0]*) y de la misma forma poder realizar la acción de dormir y despertar.
No obstante, hemos visto cómo nuestra mota se conectaba a otra a tiempo real y cómo funcionaba la interacción nodo coordinador-nodo secundario.