Ejercicio *tracking*.

Monitorización de elementos móviles.

SUIA 2018-19

Erik Fustes, Ainhoa Havelka e Iñigo Olaso

Índice

1. Programación de balizas (sin nodos ni sink)

Se ha definido una estructura en la que guardar los datos a enviar por las balizas (**broadcast_message**). En ella se almacena:

- El ID del nodo en el escenario general (beaconID)
- El tipo de nodo (nodeType → si es sink == 0, baliza == 1 o nodo == 2)

Esta estructura se envía por la red como *anonymous broadcast* por el canal 126. Cada baliza envía este mensaje con su propia información cada 20 segundos.

2. Programación de nodos (integración con balizas)

2.1 Implementar recepción y cálculo de RSSI.

Los nodos deben ser capaces de recibir de las balizas la estructura definida en el apartado 1. Una vez recibida calculan también el RSSI asociado al mensaje recibido.

Esta información se almacena en una estructura llamada **Beacon**. Los parámetros de la estructura son:

- ID de la baliza
- RSSI
- tipo del nodo (nodeType → si es sink == 0, baliza == 1 o nodo == 2)

```
//Almacenar baliza
struct beacon {
    char beaconId;
    int rssi;
    char type;
};
```

2.2 Comparación la mejor baliza de 3.

Una vez implementada la estructura **Beacon**, los nodos deben ser capaces de almacenar las tres mejores balizas recibidas en un período de tiempo de 60 segundos.

Para ello se ha definido una estructura **Beacon_3** que almacena dichas tres balizas. La estructura contiene los siguientes datos:

- Array de tres elementos tipo **Beacon**.
- Entero **“n_balizas”** para indicar cuántas balizas hay guardadas en la estructura Beacon.
- Entero **“my_id”** para indicar el ID del nodo que está enviando esas balizas.

```
//Array 3 balizas
struct beacon_3 {
    struct beacon beacons[3];
    int n_balizas;
    int my_id;
} beacon_Temp_3;
```

La decisión de cuál es la mejor baliza se realiza cada vez que recibimos un mensaje de una baliza y se realiza en una función con los siguientes pasos:

- Si la baliza leída ya está registrada, se comparan los RSSI de ambas y nos quedamos con la baliza de menor RSSI (valor absoluto)
- Si la baliza leída no está registrada y todavía no tenemos tres balizas registradas, la nueva baliza se guarda en la siguiente posición
- Si la baliza leída ya está registrada y ya tenemos tres balizas registradas se debe compara la nueva lectura con las tres almacenadas. Se comparan y se escribirá la nueva en la primera de peor RSSI que la leída.

2.3 Histórico de 20 registros (tipo Beacon_3)

Cada 60 segundos se guardan la estructura **Beacon_3** con las tres balizas más cercanas al nodo. El histórico de balizas soporta 20 registros de estructuras Beacon_3. Se ha implementado el histórico a modo de cola circular (se sobreescribe el registro más antiguo en caso de que la cola esté llena).

```
#define MAX_ITEMS    20
typedef struct Queue
{
    int     first;
    int     last;
    int     validItems;
    struct beacon_3     data[MAX_ITEMS];
};
struct Queue circularQueue_t; //Histórico balizas
```

3. Programación de sinks (integración con balizas y nodos) - comunicación directa

Los nodos sink son los encargados de recibir los registros históricos de los nodos. Para que los nodos sepan dónde se encuentra el sink (o más bien, si es que están a su alcance), el nodo sink envía mensajes tipo “broadcast_message” (como las balizas, pero con nodeType == 0) para promocionarse. Este envío se realiza por el mismo canal (126).

3.1 Definición de un canal adicional (canal 127) para la recepción del histórico

Como ahora los nodos sink tiene que ser capaces de recibir los registros tipo **Beacon_3** es necesario definir un nuevo canal de comunicación para enviar este tipo de estructuras. Se envían estas estructuras en forma de *anonymous broadcast* por el canal 127.

4. Enrutamiento dinámico de los paquetes

El funcionamiento del enrutamiento dinámico de los paquetes se divide en dos casuísticas que se explicarán a continuación.

4.1 Formación de la red

La estructura "broadcast_message" se ha sustituido por la estructura "multihop_message", que esencialmente sigue siendo igual, salvo que se le ha incluido el parámetro "saltos", que indica a cuántos saltos está ese nodo/baliza del sink.

```
struct multihop_Message {
    int beaconId;
    int nodeType;
    int salto;
};
```

Las balizas y el nodo sink envían este mensaje cada 20 segundos.

Al inicio de la formación de la red todas las balizas contienen '-1' en el campo de "salto". No envían la estructura "multihop_message" hasta que no sepan a cuántos saltos están (*salto == -1*). El nodo sink es el que comienza la cadena enviando el primer "multihop_message" con un número de saltos conocido.

Cuando una baliza recibe un "multihop_message" ya puede saber a cuántos saltos está del nodo sink y comenzar a enviar ella misma este tipo de mensajes. De esta forma se propaga por la red la distancia (en saltos) al nodo sink.

Las balizas y los nodos a su vez, cuando reciben mensajes multihop, deciden a qué baliza o sink están "enlazados" (que también comienza siendo '-1'). Se "enlazan" a la baliza o sink que a menos saltos esté. En caso de empate, se elige también la más cercana (la de menor RSSI).

Se guardan como variables locales (al nodo/baliza) el ID del sink/baliza al que están enlazados, el número de saltos al que se encuentra su enlace y lo cerca que están de su enlace.

```
//Variables multihop
char id_enlace=-1;
char salto_enlace=-1;
int  rssi_enlace=-88;
//--------------------
```

4.2 Envío de los paquetes

La gestión de envío de paquetes (*de registros tipo Beacon_3*) sigue siendo la misma que en los anteriores apartados, con la salvedad de que ahora (a pesar de seguir siendo mensajes tipo *anonymous broadcast*), van dirigidos (con un ID) a la baliza o sink al que están enlazados. Las balizas reenvían los mensajes recibidos a su propio nodo de enlace, permitiendo de esta forma que los mensajes lleguen al nodo sink.

5. Frecuencia de envío de mensajes / actualización de enrutamiento

En este apartado se listará la frecuencia de envío de cada uno de los mensajes. Además, se explicará una implementación adicional al enrutamiento dinámico de los paquetes.

5.1 Frecuencia de envío de mensajes

5.1.1 Sink

- Envío de "multihop_message" cada 20 segundos (facilita el enrutamiento).

5.1.2 Baliza

- Envío de "multihop_message" cada 20 segundos (facilita el enrutamiento)
- Envío de "Beacon_3" cada vez que se recibe un registro (enrutamiento directo al siguiente enlace, sin espera).

5.1.3 Nodo

- Envío de un registro de la cola circular cada segundo (si está vacía, no hace nada).

5.2 Actualización de enrutamiento

Cada 370 segundos todas las balizas reinician sus enlaces ('-1' en los campos correspondientes). Hacen una espera de 30 segundos para sincronizar el reinicio entre todas las balizas, y vuelven a iniciar el protocolo de enrutamiento. De esta forma, si hubiera cambiado la estructura de la red, cada 400 segundos se volverían a encaminar los paquetes de la forma adecuada.

Los nodos sink reinician su "enlace" cada 180 segundos.