UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

SOPORTE PARA LA EDUCACIÓN DE LABORATORIO EN EL PROCESAMIENTO DIGITAL DE SEÑALES. COMPARACIÓN BASADA EN EL CONSUMO DE ENERGÍA ENTRE LOS PROTOCOLOS DE ENRUTAMIENTO PARA REDES DE SENSORES INALÁMBRICOS MÓVILES EN MATLAB

Rubén Diab Martínez

Tutor: Pavel Zahradník

Cotutor: Ignacio Bosch Roig

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2017-18

Valencia, 23 de junio de 2018

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universitat Politècnica de València
Edificio 4D. Camino de Vera, s/n, 46022 Valencia
Tel. +34 96 387 71 90, ext. 77190
www.etsit.upv.es

VLC/ CAMPUS VALENCIA, INTERNATIONAL CAMPUS OF EXCELLENCE

EMAS Gestión ambiental verificada REG.NO. ES-CV- 000030

Resumen

Este proyecto consiste en 2 partes, las cuales son realizadas 100% mediante *MatLab*:

1. Soporte para la educación de laboratorio en el procesamiento digital de señales en *MatLab*:

- Se centra en el procesamiento de señales de audio en tiempo real utilizando las opciones recientes de *MatLab*. El procesamiento de señal incluye filtrado de señal y análisis de señal.

2. Comparación basada en el consumo de energía entre los protocolos de enrutamiento para redes de sensores inalámbricos *(WSN)*:

- Implementación de diferentes protocolos de enrutamiento de eficiencia energética en *WSN*.

- Comparar estos protocolos en el caso de movilidad, donde la red es una red de sensores inalámbricos móviles *(MWSN).*

Resum

Aquest projecte consisteix en 2 parts, les quals són realitzades 100% mitjançant *MatLab*:

1. Suport per a l'educació de laboratori en el processament digital de senyals en *MatLab*:

- Se centra en el processament de senyals d'àudio en temps real utilitzant les opcions recents de *MatLab*. El processament del senyal inclou el filtratge de senyals i l'anàlisi del senyal.

2. Comparació basada en el consum d'energia entre els protocols d'enrutament per reds de sensors inalàmbrics *(WSN)*:

- Implementació de diferents protocols d'enrutament d'eficiència energètica en *WSN*.

- Comparar aquests protocols en el cas de la mobilitat, on la red es una red de sensors mòbils *(MWSN).*

Abstract

This project consists of two parts, which are both 100% realized in *MatLab*:

1. Support for laboratory education in digital signal processing:

- It is focused on processing of audio signals in real time using recent *MatLab* options. The signal processing includes signal filtering and signal analysis.

2. Energy consumption-based comparison between routing protocols in Wireless Sensor Networks *(WSN)*:

- Implementing different energy efficient routing protocols in *WSN*.

- Compare these protocols in the mobility case, where the network is a mobile wireless sensor networks *(MWSN)*.

Índice

CAPÍTULO 1. SOPORTE PARA LA EDUCACIÓN DE LABORATORIO EN EL PROCESAMIENTO DIGITAL DE SEÑALES EN MATLAB

1.1 Objetivo del procesamiento de señales digitales en *MatLab*

En primera instancia, el objetivo principal de la parte del *TFG* centrada en el Soporte para la educación de laboratorio en el procesamiento digital de señales en *MatLab*, consiste en la búsqueda de las opciones más recientes y eficientes para el tratamiento de señal de audio en tiempo real.

Se requiere de unas funciones *MatLab* que permitan guardar y procesar la entrada de señal de audio, ya sea introducido por el canal Jack, por el micrófono o mediante un archivo del ordenador, y que, de manera simultánea, esa señal de audio sea reproducida.

Finalmente, las funciones elegidas corresponden a "*The Audio System Toolbox"* [1], ya que proporcionan procesamiento de las señales de audio en tiempo real y con baja latencia utilizando objetos del sistema como *audioDeviceReader* y *audioDeviceWriter*.

1.2 Códigos *MatLab* principales

Código *MatLab* principal utilizado para realizar la lectura de un archivo y la escritura al dispositivo de audio:

```
fileReader = dsp.AudioFileReader('speech_dft.mp3');
fileInfo = audioinfo('speech_dft.mp3');

deviceWriter = audioDeviceWriter(...
    'SampleRate',fileInfo.SampleRate);
setup(deviceWriter,...
    zeros(fileReader.SamplesPerFrame,fileInfo.NumChannels));

while ~isDone(fileReader)
    audioData = fileReader();
    deviceWriter(audioData);
end

release(fileReader);
release(deviceWriter);
```

Código *MatLab* principal utilizado para realizar la lectura desde el micrófono y la escritura al dispositivo de audio:

```
deviceReader = audioDeviceReader;
setup(deviceReader);

fileWriter = dsp.AudioFileWriter(...
    'mySpeech.wav',...
    'FileFormat','WAV');

disp('Speak into microphone now.')
tic;
while toc < 10
    acquiredAudio = deviceReader(); % Graba 10 segundos de audio
    fileWriter(acquiredAudio);
end
disp('Recording complete.')

release(deviceReader);
release(fileWriter);
```

1.3 Optimización de las funciones correspondientes a "*The Audio System Toolbox*"

Para optimizar y conseguir el mayor rendimiento de las funciones correspondientes a "*The Audio System Toolbox"* [1], es necesario:

1) Encontrar el tiempo de demora límite, introducido entre cada bloque de la señal de audio procesada en tiempo real, a partir de donde se producen pérdidas.
2) Tener el control de la elección del tamaño de cada bloque de la señal de audio procesada en tiempo real, tanto de entrada como de salida.
3) Aplicar un algoritmo para filtrar por bloques. Principalmente existen dos métodos:
   - *Solapamiento y suma*
   - *Solapamiento y almacenamiento*

   En este caso se ha optado por aplicar el método de *Solapamiento y suma* [2].

1.3.1 Pérdidas de la señal procesada en tiempo real

Este apartado se centra en la búsqueda de pérdidas cuando se procesa la señal de audio en tiempo real y cuál es el límite, medido en tiempo, a partir de donde se empiezan a producir.

Estas pérdidas se producen en las fronteras de cada bloque o segmento de señal de audio de entrada y salida.

Para ello, se introduce una demora en el código *MatLab* utilizando la función *pause()*. Para averiguar exactamente cuál es el límite, se realizan pruebas introduciendo diferentes tiempos en segundos dentro del paréntesis de la función previamente comentada.

Introduciendo diferentes retardos empezando por 1 ms hasta 1 segundo se halla que en el abanico comprendido entre 60 y 70 ms se empieza a percibir pérdidas audibles.

Tanteando diferentes posibilidades en el abanico previamente mencionado, se descubre que el límite, donde se puede procesar señal de audio en tiempo real sin que se produzcan pérdidas audibles, ocurre a partir de una demora de 63 ms.

Código *MatLab* donde se aplica la demora (subrayado en amarillo):

```
fileReader = dsp.AudioFileReader('speech_dft.mp3');
fileInfo = audioinfo('speech_dft.mp3');

deviceWriter = audioDeviceWriter(...
    'SampleRate',fileInfo.SampleRate);
setup(deviceWriter,...
    zeros(fileReader.SamplesPerFrame,fileInfo.NumChannels));

while ~isDone(fileReader)
    audioData = fileReader();
    deviceWriter(audioData);
    pause(63e-3) %Delay of 63 ms is the limit for real time audio
processing without audible drop outs. When introducing a delay above
63 ms, the drop outs are audible. (When introducing 1024 samples per
frame).
end

release(fileReader);
release(deviceWriter);
```

1.3.2 Elección del tamaño de cada bloque de la señal de entrada y de salida

Para eliminar las pérdidas que se producen procesando la señal de audio en tiempo real es necesario tener el control del tamaño de cada bloque de la señal de audio, tanto de entrada como de salida.

Esto se puede conseguir cuando creamos los objetos de entrada y de salida.

El código *MatLab* correspondiente a esta configuración para la entrada de señal de audio por el micrófono es el siguiente (se puede observar subrayado en amarillo):

```
%% Real-Time Audio Stream Processing
%% Create input and output objects
fs = 16e3; %audioDeviceReader will operate at 16 kHz sample rate

sampPerFrameValue = 1024;
deviceReader = audioDeviceReader(fs,sampPerFrameValue);
deviceWriter =
audioDeviceWriter('SampleRate',deviceReader.SampleRate);
```

Y el código *MatLab* correspondiente para la entrada de señal de audio proveniente de un archivo es el que se muestra a continuación (se puede observar subrayado en amarillo):

```
%% Real-Time Audio Stream Processing
%% Create input and output objects
fs = 16e3; %audioDeviceReader will operate at 16 kHz sample rate

sampPerFrameValue = 1024;
fileReader =
dsp.AudioFileReader('Rueda.mp3','SamplesPerFrame',sampPerFrameValue);
deviceWriter = audioDeviceWriter('SampleRate',fileReader.SampleRate);
```

El tamaño del bloque es igual a 1024 debido a que es el predeterminado y recomendado para las funciones *audioDeviceReader* y *audioDeviceWriter*.

El control del tamaño del bloque de la señal de audio de entrada y salida nos sirve de ayuda para la eliminación de las pérdidas que se producen en las fronteras de cada bloque.

1.3.3 Algoritmo de Solapamiento y suma

Para conseguir eliminar las pérdidas introducidas por la aplicación de los diferentes filtros *FIR*, se utiliza el método de *Solapamiento y suma* [2].

Este algoritmo se denomina así por el hecho de que las secciones filtradas se solapan y se suman al construir la señal de salida.

Este método es una manera eficiente de evaluar la convolución discreta de una señal muy larga *x(n)* con un filtro *h(n)* de respuesta de impulso finita (*FIR*):

$$\boldsymbol{y[n] = x[n] * h[n] = \sum_{k=1}^{M} x[n-k] \cdot h[k]} \qquad \textbf{(1)}$$

Si la longitud de *h[n]* es *L* y la longitud del bloque *x[n]* es *M*, entonces la convolución resultante *y[n]* tendrá una longitud igual a *L+M-1*.

El concepto es dividir el problema en múltiples convoluciones de *h[n]* con pequeños segmentos de *x[n]* y procesarlas en tiempo real.

El solapamiento se produce debido a que la convolución lineal de cada sección, con la respuesta al impulso del filtro, es en general mayor que la longitud del segmento.

Básicamente, la convolución resultante es la suma de los resultados de la convolución de cada sección.

A continuación, se muestra en la figura 1 la representación del concepto de este algoritmo:

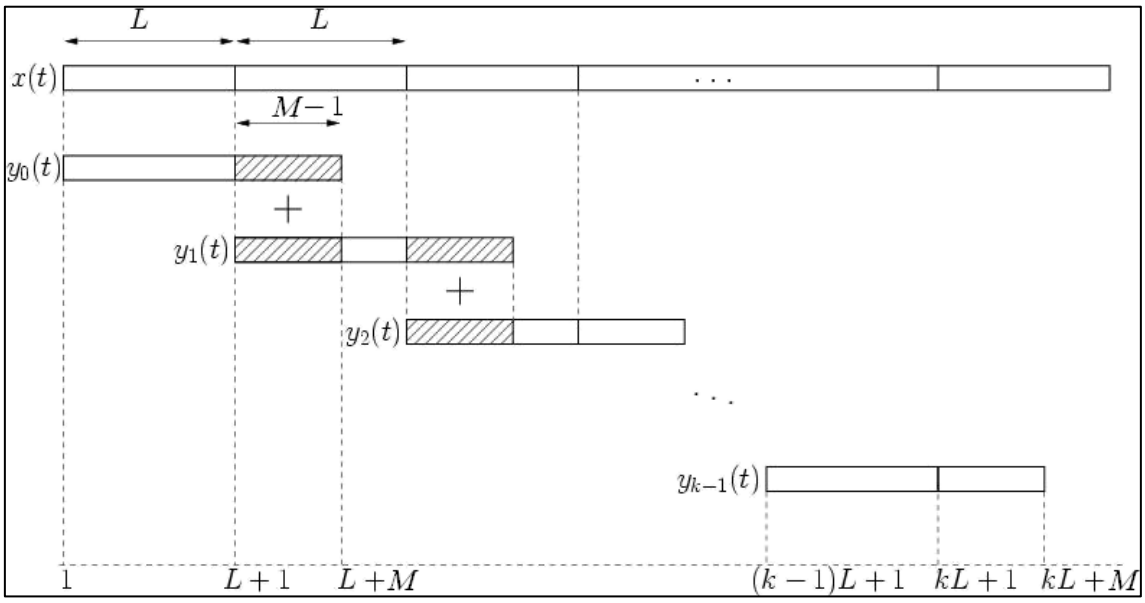

Figura 1. Método de Solapamiento y suma

El código *MatLab* realizado para implementar lo explicado previamente, y correspondiente a la entrada de señal de audio por el micrófono (se puede observar subrayado en amarillo), es el siguiente:

```
%% Code for stream processing (for device reader and FIR filters)

time=input('Introduce time wanted for real time audio stream
processing = ');

disp('Begin Signal Input...')

SizeQueue = M; % Size of the convolution queue
Queue = zeros(1,SizeQueue); % Buffer to store the queue

tic
while toc<time
    mySignal = deviceReader();

    filteredSig = myconv(blp,mySignal);

    out = filteredSig(1:sampPerFrameValue);
    out(1:SizeQueue) = out(1:SizeQueue) + Queue;
    Queue = filteredSig(sampPerFrameValue+1:end);
```

```
    deviceWriter(out');
    end
disp('End Signal Input')

release(deviceReader)
release(deviceWriter)
```

Y el código *MatLab* correspondiente a la entrada de señal de audio proveniente de la lectura de un archivo (se puede observar subrayado en amarillo) es:

```
%% Code for stream processing (for file reader and FIR filters)

disp('Begin Signal Input...')

SizeQueue = M; % Size of the convolution queue
Queue = zeros(1,SizeQueue); % Buffer to store the queue

while ~isDone(fileReader)
    mySignal = fileReader();
    mySignal = mySignal(:,1);

    filteredSig = myconv(blp,mySignal);

    out = filteredSig(1:sampPerFrameValue);
    out(1:SizeQueue) = out(1:SizeQueue) + Queue;
    Queue = filteredSig(sampPerFrameValue+1:end);

    deviceWriter(out');

end
disp('End Signal Input')

release(fileReader)
release(deviceWriter)
```

Para la convolución se ha creado una función propia para procesar y entender mejor los problemas que sucedían en las fronteras de las secciones de *x[n]*. Se ha implementado esta función, en lugar de usar la función de convolución de *MatLab*, con el fin de hacer frente a problemas ya comentados como las pérdidas de la señal procesada en tiempo real o facilitar la implementación del algoritmo de *Solapamiento y suma*.

El código *MatLab* correspondiente a la función de la convolución es:

```
%% Function of Convolution
function y = myconv(x,h)
% INPUTS:
% x ... input signal
% h ... impulse response signal

[rx,cx] = size(x);
[rh,ch] = size(h);
if rx>1
    x = x';
end
if rh>1
    h = h';
end
```

```
m = length(x);
n = length(h);
X = [x,zeros(1,n)];
H = [h,zeros(1,m)];
Y = zeros(1,n+m-1);
for i = 1:n+m-1
    for j = 1:n
        if (i-j+1>0)
            Y(i) = Y(i) + X(i-j+1)*H(j);
        else
        end
    end
end
[rY,cY] = size(Y);
if rY>1
    y = Y';
else
    y = Y;
end
```

1.4 Aplicación de filtros FIR

Para modificar la señal procesada en tiempo real se han implementado filtros *FIR* [3] paso banda, paso bajo y paso alto.

El objetivo de la implementación de estos filtros es tratar la señal de audio en tiempo real y comprobar que el algoritmo de *Solapamiento y suma* eliminan las pérdidas a la hora de realizar la convolución con la aplicación de estos filtros. Efectivamente, las pérdidas son eliminadas y en consecuencia se cumple el objetivo deseado.

Los códigos *MatLab* correspondientes a los filtros *FIR* implementados son:

```
%% FIR LP filter
M = 512; % Order of the filter
Wc = 0.5; %Normalized frequency
blp = fir1(M,Wc,'low');

%% FIR BP filter
M = 50;
fc1 = 300;
fc2 = 3400;
wc1 = fc1/(fs/2);
wc2 = fc2/(fs/2);
Wc = [wc1 wc2];
bbp = fir1(M,Wc,'bandpass');

%% FIR HP filter
M = 4;
Wc = 0.5;
bhp = fir1(M,Wc,'high');
```

1.5 Posibles aplicaciones

A parte de la aplicación de los filtros *FIR*, las funciones recientes en *MatLab* que permiten el procesamiento de señal de audio en tiempo real podrían ser aprovechadas de muchas maneras, ofreciendo así, un abanico muy grande de posibles aplicaciones.

Por ejemplo, la aplicación de:

- Filtros *IIR*
- Cálculo del espectrograma de la señal
- Efectos
- Sistemas de síntesis de señales de audio
- Análisis deseado de las señales de audio
- Etc.

CAPÍTULO 2. COMPARACIÓN BASADA EN EL CONSUMO DE ENERGÍA ENTRE LOS PROTOCOLOS DE ENRUTAMIENTO PARA REDES DE SENSORES INALÁMBRICOS MÓVILES

2.1 Objetivo

Las redes de sensores inalámbricos móviles (*MWSN – Mobile Wireless Sensor Network*) han aparecido y evolucionado durante estos últimos años. Han cambiado el enfoque inicial, desde las típicas redes de sensores inalámbricos estáticos hasta las redes de sensores o nodos inalámbricos móviles que son capaces de detectar, monitorizar y estudiar condiciones físicas o ambientales, como puede ser la humedad, temperatura, velocidad del aire, etc. Todos estos datos son recogidos por los sensores y posteriormente, de manera cooperativa son distribuidos y enviados hasta una estación base con diferentes posibles fines, como por ejemplo el control de procesos industriales, la monitorización de la salud, el estudio del ejercicio realizado por una persona en función de la cantidad de pasos y distancia realizados, etc. El envío de los datos obtenidos por los nodos se consigue siguiendo algún tipo de algoritmo.

Actualmente existen muchos algoritmos, y cabe la posibilidad de la creación e implementación de muchos más para el enrutamiento de la información analizada por los sensores. En las redes de dispositivos inalámbricos, para el intercambio de datos entre los sensores, es necesario mantener y descubrir de manera continua la topología de la red debido a que los nodos no disponen del conocimiento necesario.

Existen diferentes tipos de modelos de enrutamiento para conseguir la comunicación entre los nodos y la estación base, las cuales serán explicadas más adelante.

Por lo tanto, el objetivo principal de esta segunda parte es la implementación de diferentes protocolos de enrutamiento, en las redes de sensores inalámbricos móviles, y llevar a cabo una comparación de la eficiencia energética proporcionada por cada algoritmo. Tener este conocimiento permite poder seleccionar los protocolos que tienen un menor consumo de energía y, que, en consecuencia, maximizan el tiempo de vida de la red.

Todo esto se lleva a cabo utilizando el software *MatLab*.

2.2 Redes de sensores inalámbricos móviles

Las redes de sensores inalámbricos móviles consisten en un conjunto de dispositivos, que pueden ser denominados sensores o nodos, de bajo coste y consumo que recogen datos (ya sean del medio ambiente, físicos, etc.), la procesan y se distribuyen de manera cooperativa e inalámbrica por los sensores siguiendo una topología de red hasta llegar a un nodo central, también conocido como estación base, o hasta diferentes destinos. Estos últimos recogen toda la información proporcionada por la red para monitorizarla y utilizarla en función de sus necesidades.

Los nodos pueden ser fijos o móviles y funcionan como elementos de la infraestructura de comunicaciones ya que reenvían la información, acorde a un algoritmo, hasta que llega al destino deseado.

Cada sensor puede actuar como fuente, retransmisor de datos o como destino.

Casi siempre, para las redes de sensores inalámbricos móviles, se utiliza una topología de red denominada *ad-hoc*. Este tipo de red se caracteriza principalmente por lo siguiente:

- La comunicación se realiza entre dos puntos.
- Se elimina el punto central y, de esta manera, cualquier dispositivo puede comunicarse directamente con otro dispositivo que se encuentre dentro de su alcance de comunicación.
- Proporciona un gran dinamismo.

En la figura 2, podemos ver un ejemplo de una red de sensores inalámbricos móviles siguiendo un protocolo de enrutamiento y transmitiendo información hasta el nodo central:

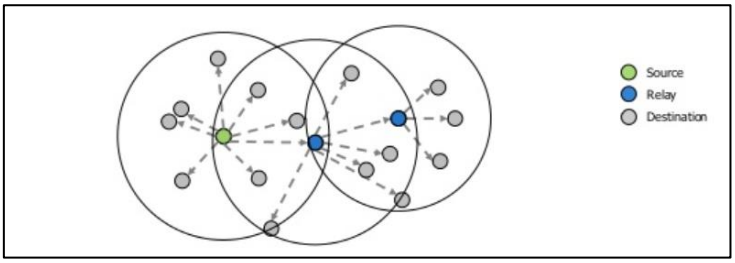

Figura 2. Protocolo de Inundación de red.

Se ha utilizado como ejemplo el protocolo de Inundación de red, conocido en inglés como *Flooding protocol* [4]. El funcionamiento de este algoritmo consiste en un continuo reenvío de la información de los sensores a todos los demás nodos, excepto de dónde provino, hasta llegar al destino.

Debido a las limitaciones que existen en la construcción de los sensores, el tiempo de vida de estos se ve limitado y lo mismo sucede con el rango del alcance de comunicación. Por tanto, el diseño de los sensores se realiza teniendo en cuenta la conservación de energía.

Las redes de sensores inalámbricos se caracterizan por ser independientes, es decir, por no tener intervención humana. Esto provoca una alta probabilidad de fallo, pero estas redes, con el paso del tiempo, se han ido desarrollando y evolucionando hasta el punto de disponer de capacidades de auto diagnóstico, configuración, organización, reparación y restauración. Por ejemplo, tanto si un sensor pierde potencia, muere, o entra como nuevo nodo a la red, puede ser solventado gracias a las capacidades previamente comentadas [5].

2.3 Elementos de los sensores inalámbricos móviles

Los componentes básicos que constituyen el diseño principal de los dispositivos inalámbricos móviles son [5,4]:

- Microcontrolador
- Fuente de energía (suele ser una batería)
- Radio-transceptor
- Sensor analógico
- Sensor digital

En la figura 3, se muestra, mediante un esquema, el diseño básico de un dispositivo inalámbrico móvil:

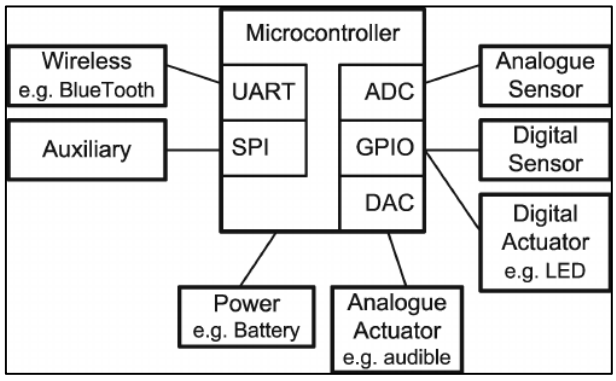

Figura 3. Esquema básico de un nodo sensor inalámbrico.

2.4 Parámetros de una red de sensores inalámbricos móviles

Los principales parámetros que definen y caracterizan una red de sensores inalámbricos móviles se muestran a continuación [6,7]:

- Tiempo de vida: Este parámetro se verá afectado mayoritariamente por el consumo de energía de la red. Por eso es muy importante un buen diseño del sensor, una correcta elección del algoritmo de enrutamiento, etc.

- Tolerancia a errores: La red debe ser capaz de solventar de alguna manera los errores que vayan surgiendo, para así poder seguir funcionando sin problemas.

- Eficiencia energética: Éste es realmente importante ya que dependiendo de él se consigue alargar el tiempo de vida de la red, tal y como se ha comentado anteriormente [8].

- Limitaciones hardware: Para conseguir una reducción notable del consumo energético es muy importante simplificar al máximo el hardware de los sensores.

- Dinamismo de la topología: La topología de la red debe de ser dinámica para poder hacer frente a los errores y poder seguir funcionando.

- Seguridad: Es importante constar de una seguridad robusta y fiable para evitar que la información recogida por la red llegue a manos indeseadas.

- Tamaño y coste de producción: Los dispositivos deben ser lo más pequeños posibles y debido a que su propia naturaleza requiere de una gran cantidad de sensores, si estos son construidos en masa, los costes de producción son económicos.

2.5 Protocolos de enrutamiento

Las redes de sensores inalámbricos móviles pueden contener miles de nodos. Como ya se ha explicado, los nodos no tienen conocimiento de la topología de la red. Para obtener esa información es necesario que los nodos realicen una petición a los demás nodos. De esta manera, se anuncian y obtienen mutuamente y cooperativamente información sobre su posición y como alcanzarlo. Pasado un tiempo, el resto de los nodos habrá enviado un mensaje de respuesta con toda la información requerida para que todos los dispositivos de la red tengan un conocimiento total sobre la topología de la red y el protocolo de enrutamiento a utilizar para que los datos sean transmitidos al destino deseado.

2.5.1 Proceso de enrutamiento

El proceso de enrutamiento, tal y como se representa en la figura 4 consiste principalmente en tres pasos:

- Descubrimiento de posibles rutas: Este paso consiste en encontrar todas las rutas adecuadas desde el nodo fuente hasta el nodo destino. Se incluye también el coste de cada ruta descubierta. Sólo se descubren las rutas que cumplan los requerimientos y condiciones correspondientes al protocolo de encaminamiento aplicado, como, por ejemplo, que haya un máximo número de saltos entre nodos.

  Proceso para llevar a cabo el descubrimiento todas las posibles rutas:

  - Paquete de solicitud de ruta enviado por el nodo origen a los nodos vecinos.

- Los nodos vecinos remiten el paquete de solicitud de ruta a sus vecinos. Este paso se repite hasta que la petición de la ruta llegue al nodo destino.
- El cómo se realice el encaminamiento de los paquetes de petición de la ruta depende del protocolo de enrutamiento específico que se esté utilizando.

- El paquete de respuesta que contiene la ruta solicitada es enviado por el nodo destino hasta el nodo fuente y por la misma ruta que ha seguido el paquete de petición de ruta.

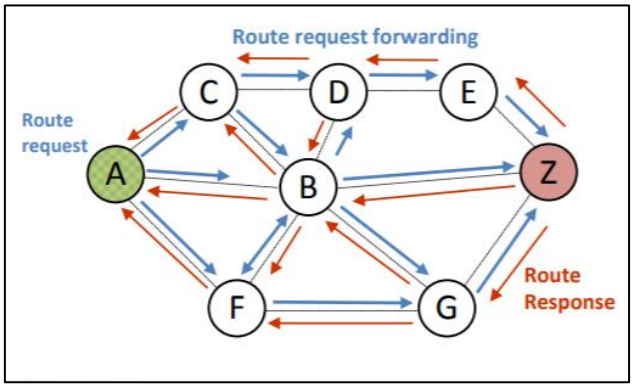

Figura 4. Esquema del proceso de descubrimiento de rutas.

- Selección de la ruta: Una vez encontradas todas las diferentes posibles rutas, se ha de seleccionar la ruta más apropiada. Esta elección depende del algoritmo de enrutamiento que se esté llevando a cabo y de su complejidad, ya que cada protocolo sigue unas condiciones y limitaciones de encaminamiento distintas.

  Para seleccionar la ruta se tiene en cuenta características de la red, entre ellos:

  - Número de saltos
  - Consumo de energía
  - Tiempo de vida de la red
  - Demora: Tiempo necesario para transmitir la información desde el nodo origen hasta el destino.
  - Confiabilidad: Proporción de paquetes entregados, desde el nodo fuente al destino, correspondientes a todos los paquetes enviados en un límite de tiempo.
  - Estabilidad de la ruta: Se revisa el estado de la ruta teniendo en cuenta la calidad del canal, la energía de los nodos, etc.
  - Gastos generales: Cantidad de información redundante necesaria para entregar todos los paquetes.
  - Rendimiento – velocidad de datos – capacidad: Cantidad de información (bits, paquetes, ...) entregada durante un período de tiempo.

- Mantenimiento de la ruta: Cuando la ruta más idónea ha sido seleccionada, es muy importante mantener actualizada la información sobre el estado de la ruta. Así se consigue

tener continuo conocimiento sobre la incorporación de un nuevo nodo, la existente interferencia, la movilidad de la red, el estado de la batería de cada sensor, la muerte de un nodo, etc.
Además, permite una selección dinámica de la ruta más adecuada en cada momento.

2.5.2 Clasificación de los protocolos de enrutamiento

Los protocolos se pueden clasificar [7,9] de dos maneras, en función de:

- El modo de operación del protocolo:

  - Proactivo: En este caso la tabla de enrutamiento se mantiene actualizada incluso si no se solicita comunicación y se puede llevar a cabo de manera periódica o cuando se realice una petición. Por lo tanto, la ruta está preparada de manera continua para cuando se requiera.

    El inconveniente de este modo de operación es que al tener que mantener continuamente actualizada la tabla de encaminamiento, el consumo de energía incrementa.

  - Reactivo: En contraposición al caso anterior, aquí se descubre una ruta mediante una petición realizada a priori.

    Este modo de funcionamiento conlleva las desventajas de producir demora porque se deben llevar a cabo los procesos de descubrimiento y elección de ruta.

    Pero presenta la ventaja de tener una mayor eficiencia energética, ya que, si no hay una comunicación en progreso, el estado de la red pasa a estar en reposo a la espera de una petición.

  - Híbrido: Éste se presenta como una combinación de los modos de operación proactivo y reactivo explicados previamente.

- La topología de la red [9]:

  - Plana: Todos los nodos constan, tanto de una importancia y un tipo de operación equitativos.

  - Protocolo basado en localización: Se explota la posición de los dispositivos para encaminar los datos en la red.

  - Jerárquica: En contraposición a los protocolos de enrutamiento planos, en este tipo, los nodos tienen una importancia y un tipo de operación desiguales. Un ejemplo de este caso es el modelo basado en el uso de nodos como *clústeres* que se encargan de, en primera instancia, recoger los datos de los demás nodos para, posteriormente, transmitirlos a la estación base o nodo destino.

2.6 Algoritmos de enrutamiento implementados en *MatLab*

Existen una gran cantidad de diferentes algoritmos de encaminamiento para las redes de sensores inalámbricas móviles. Cada protocolo tiene un modo de funcionamiento y características diferentes, y, en consecuencia, constan de una eficiencia energética desigual. Pero no siempre consiste en elegir aquel protocolo que proporcione un menor consumo de energía, sino aquel que cumpla mejor con las necesidades requeridas a la vez que se tiene en cuenta la eficiencia energética.

Se ha limitado la implementación a cuatro protocolos de enrutamiento debido a la carga de trabajo necesario para su implementación en *MatLab*.

Los algoritmos seleccionados son los que se muestran a continuación, y que más adelante serán explicados individualmente:

- *Comunicación Directa*
- *LEACH*
- *PEGASIS*
- *DREAM*

La elección de estos algoritmos no tiene ninguna explicación, ya que se podrían haber seleccionado otros totalmente diferentes. Básicamente, se tuvo la idea de implementar el protocolo de *Comunicación Directa*, la implementación del cual es sencilla y se conoce que su gasto energético es alto para unas condiciones específicas, con el fin de buscar otros protocolos que teóricamente tuvieran un menor consumo de energía y poder realizar así una comparación de la eficiencia energética que proporcionara buenos resultados.

2.6.1 Protocolo de Comunicación Directa

Este es el protocolo más simple y consiste en la transmisión de datos directa desde los nodos fuente hasta la estación base [4,7].

Presenta varios problemas que hacen que sea un protocolo inviable para las comunicaciones inalámbricas en la red de sensores, sobre todo si las distancias entre los nodos y la estación base son amplias, que principalmente son:

- El alcance de comunicación de los sensores es limitado debido a que su potencia también lo es. Por lo tanto, muchos nodos no podrían realizar una transmisión directa al destino debido a la distancia entre ellos.

- El consumo energético producido por este protocolo es muy elevado.

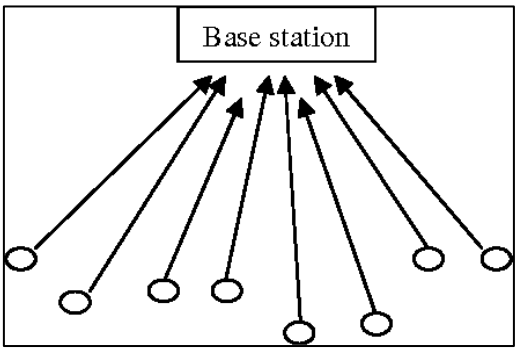

Figura 5. Esquema representativo del funcionamiento del protocolo de *Comunicación Directa*.

2.6.2 Protocolo LEACH

LEACH (Low Energy Adaptative Clustering Hierarchy) es un protocolo centralizado y de acceso múltiple por división de tiempo *(TDMA – Time Division Multiple Acces)*. Ésta última es una técnica que permite la transmisión de señales digitales, provenientes de distintas fuentes, compartiendo un mismo canal de frecuencia, lo cual se consigue dividiendo la señal en ranuras alternas de tiempo, y reduciendo así, las colisiones entre los paquetes de datos.

El funcionamiento de este protocolo se explica, de manera ordenada, siguiendo los siguientes pasos:

1) Se seleccionan un número de nodos de la red de sensores inalámbricos móviles para que funcionen como clústeres.

2) Los nodos de la red que no han sido elegidos como clústeres, en primera instancia, enviarán los datos que han recogido al clúster que tengan más cercano de ellos.

3) Una vez los clústeres han recolectado toda la información de los sensores de la red, la enviarán directamente a la estación base o nodo destino.

Se han de realizar las siguientes consideraciones al respecto:

El número de nodos elegidos que funcionan como clústeres, se ha implementado en *MatLab* para que ronde entre un 3% y un 10% con respecto al número de nodos total de la red. Este rango puede ser variado en función de lo que se desee conseguir. En este caso, utilizando el rango mencionado se han conseguido buenos resultados de eficiencia energética.

Después de un intervalo de tiempo, se cambian los clústeres. En este caso, se ha implementado en *MatLab* que este cambio se produzca cada segundo.

Los clústeres se eligen de manera aleatoria siguiendo la ecuación 2 [4,7]:

$$T(n) = \begin{cases} \dfrac{P}{1 - P \times [r \times mod(P^{-1})]} & \forall\, n \in G \\ 0 & \forall\, n \notin G \end{cases} \qquad (2)$$

Donde:

- *n* es el nodo seleccionado.
- *P* es la probabilidad de un nodo de ser clúster
- *r* es la ronda actual.
- *G* es el conjunto de nodos que no fueron elegidos como clústeres en la anterior ronda.
- *T(n)* es el límite.

Siguiendo la ecuación (2), el nodo se convierte en clúster para la ronda actual si el número es menor que el umbral *T(n)*. Y una vez, un nodo ha sido seleccionada como clúster, éste no puede volver a serlo hasta que lo hayan sido todos los demás nodos, por lo menos una vez.

En la figura 6, se representa el funcionamiento del protocolo *LEACH*:

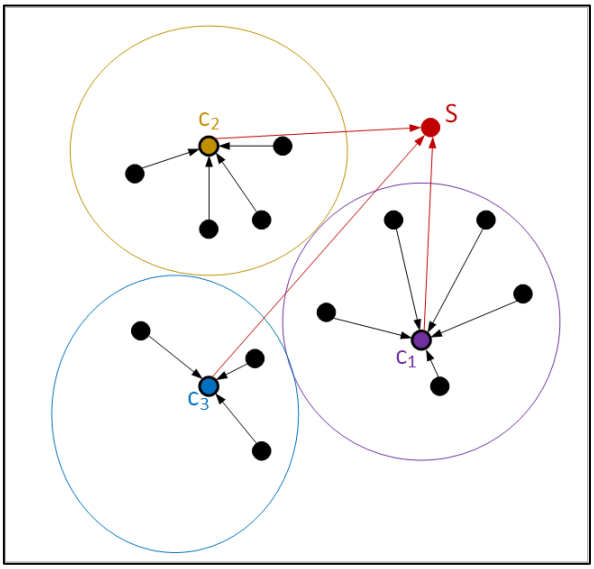

Figura 6. Esquema representativo del funcionamiento del protocolo *LEACH*.

Con la implementación y ejecución en *MatLab* del protocolo, se consigue con cada simulación un caso distinto al anterior debido a que tanto el posicionamiento de los nodos como la selección de los clústeres es aleatoria. A continuación, se muestra en la figura 7 una de las simulaciones:

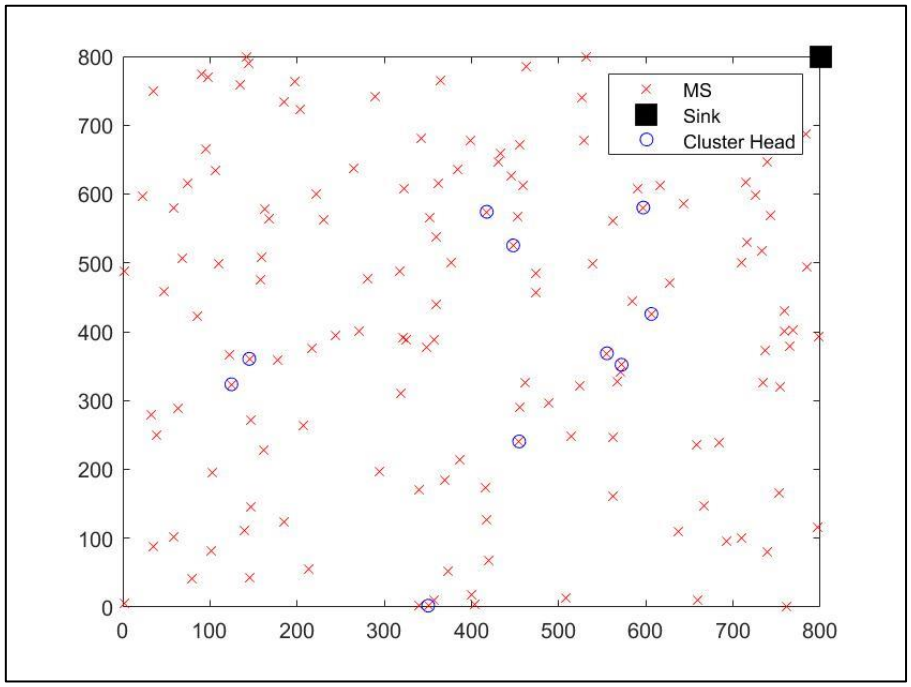

Figura 7. Simulación en *MatLab* del protocolo *LEACH*.

La simulación de la figura 7 se ha obtenido mediante la implementación de diferentes funciones:

- Generación del posicionamiento de las estaciones móviles (ANEXO C – 3.1).
- Elección de los clústeres.

Las funciones previamente mencionadas se explicarán detalladamente más adelante.

Para la simulación mostrada en la figura 7 se ha utilizado un escenario, cuya área tiene unas dimensiones de 800 x 800 m^2.

El número de estaciones móviles es igual a 150 y el número de clústeres igual a 10.

Como se puede observar, se ha posicionado, en todos los casos, la estación base *(sink)* en la esquina superior derecha del escenario utilizado.

Al principio se probó situando la estación base en el centro, pero se comprobó que, de esta manera, el protocolo no era viable ya que el consumo de energía era incluso mayor que el protocolo de *Comunicación Directa*. Esto se debe a que la función de los clústeres es ayudar a equilibrar la carga de datos a lo largo de la red y si el *sink* está en el centro, la distancia de los nodos en la red con el *sink* es relativamente cercana y, por eso, es mejor el algoritmo de *Comunicación Directa* ya que es mejor que los nodos transmitan de manera directa. En cambio, situando la estación base lejos de la mayoría de los nodos, se consigue una reducción bastante alta en la potencia de transmisión de los dispositivos, debido a las grandes distancias entre nodos y el *sink*, y, en consecuencia, una mejor distribución de la carga de energía en la red y una mayor eficiencia energética.

El protocolo *LEACH* presenta ventajas, por una parte, e inconvenientes por otra.

Ventajas:

- Los clústeres recogen todos los datos de los nodos de la red de sensores inalámbricos móviles, lo cual ayuda en equilibrar la carga en toda la red.

- Incrementa el tiempo de vida de los sensores en la red.

Desventajas:

- Cuando un clúster muere, éste se volverá inservible y, en consecuencia, la información transmitida por los nodos hasta el clúster nunca alcanzará la estación base. Actualmente, se han implementado mejoras para solventar este problema, en caso de que se presente.

- Los clústeres son elegidos de manera aleatoria cada ronda, y esto conlleva a que la distribución de carga en la red sea desigual en la red ya que habrá clústeres que tengan más nodos enviándoles datos y otros que tengan menos nodos. Este fenómeno puede producir un aumento en el consumo de energía.

El protocolo *LEACH* está implementado teniendo en cuenta que todos los nodos tienen la potencia necesaria para transmitir los datos directamente a la estación base. Por lo tanto, este no es un algoritmo viable para escenarios donde el área sea muy amplia, ya que la distancia de algunos nodos con el *sink* será tan grande que la potencia a consumir será muy elevada, lo que lleva a un gran aumento del consumo energético, o, directamente, habrá algunos nodos que sean incapaces de entregar los datos a la estación base [6].

2.6.3 Protocolo PEGASIS

PEGASIS (Power-Efficient Gathering in Sensor Information Systems) es básicamente una mejora del protocolo *LEACH* [6,7].

El funcionamiento de este protocolo, a diferencia del algoritmo *LEACH* que selecciona varios nodos como clústeres, elige un único nodo de la red de sensores inalámbricos móviles, el cual puede ser denominado nodo principal, "líder" o "maestro".

A partir de la selección del nodo maestro, se crea una cadena formada por los nodos de la red. El nodo líder no se selecciona de manera aleatoria tal y como sucedía en el protocolo *LEACH*, sino que se selecciona aquel más cercano a la estación base. Esto presenta una gran mejora en la eficiencia energética.

La cadena se crea en función de los siguientes pasos:

1) Se elige el nodo maestro de todos los nodos que forman la red, que será el más cercano a la estación base.

2) Se eligen cuantas cadenas se desean formar, y los nodos fuente u origen que empezarán la transmisión de datos. Los nodos fuente son aquellos más lejanos de la estación base.

3) Los nodos origen transmiten la información al nodo vecino más cercano, y el siguiente nodo entrega los datos al que tenga también más cerca y así sucesivamente.

4) Los nodos retransmiten los datos hasta que las diferentes cadenas llegan al único nodo maestro de la red.

5) Después de que el nodo maestro recolecte todos los datos de la red, es el encargado de transmitirla hasta la estación base.

Hay que tener diferentes cosas en cuenta sobre el algoritmo *PEGASIS*:

- *PEGASIS* utiliza una técnica de agregación de datos que permite que ir añadiendo toda la información entregada por los nodos anteriores, y enviar esa información, más la del nodo actual, al siguiente nodo.

- Para conseguir que los nodos transmitan al nodo vecino más cercano, estos detectan la potencia de sus nodos vecinos y reajustan su potencia para que sea igual a la necesaria para entregar los datos al nodo que se halle más cerca.

A continuación, en la figura 8, se muestra una representación del modelo del funcionamiento del protocolo *PEGASIS*:

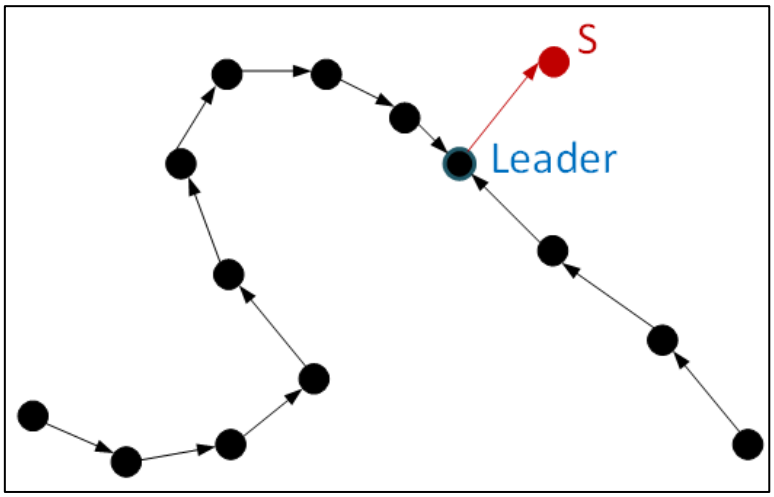

Figura 8. Esquema representativo del funcionamiento del protocolo *PEGASIS*.

Con la implementación y ejecución en *MatLab* del protocolo, se consigue con cada simulación un caso distinto al anterior debido a que el posicionamiento de los nodos en el escenario es aleatorio. A continuación, se muestra una figura de *MatLab* correspondiente a una de las simulaciones:

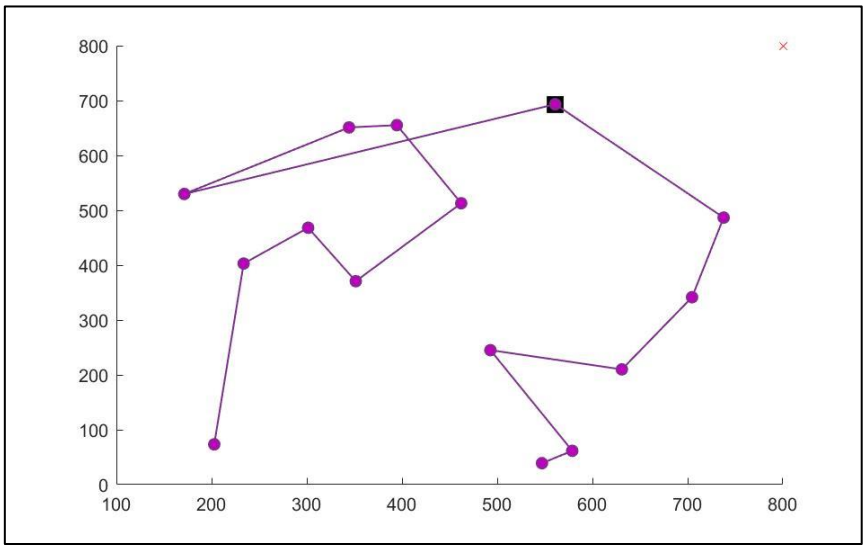

Figura 9. Simulación del protocolo *PEGASIS*.

Con respecto a la figura 9, la cruz roja corresponde a la estación base y el nodo con un cuadrado de color más oscuro corresponde al nodo maestro.

Para la simulación mostrada se ha utilizado un escenario, cuya área tiene unas dimensiones de 800 x 800 m^2.

El número de estaciones móviles es igual a 15. Se ha elegido, para este caso, un número bastante pequeño de sensores con el fin de poder representar con mayor claridad el modelo y el funcionamiento del protocolo. En el código *MatLab* utilizado para calcular el consumo y eficiencia energética de cada protocolo (ANEXO C – 3.5), el número de estaciones móviles es igual a 150.

Al igual que sucedía con el protocolo *LEACH*, el algoritmo *PEGASIS* presenta ventajas e inconvenientes.

Ventajas:

- La eficiencia energética representa una mejoría aproximadamente del doble en comparación con el protocolo *LEACH*. Por lo tanto, mejora el doble el tiempo de vida de la red de sensores inalámbricos móviles.

- Minimiza la distancia de transmisión entre nodos.

- En general, el protocolo *PEGASIS* es una mejora del algoritmo *LEACH*.

Desventajas:

- Al igual que en el protocolo *LEACH*, la implementación de *PEGASIS* está pensada para que todos los nodos puedan transmitir los datos directamente hasta la estación base. Por lo tanto, no es viable para escenarios cuya área sea muy elevada.

- Al utilizar la técnica de agregación de datos y debido a su funcionamiento, el tiempo de demora es elevado.

- *PEGASIS* requiere de una continua actualización de las tablas de enrutamiento para que los nodos puedan conocer el estado energético del resto de nodos, con el fin de poder conseguir las rutas de encaminamiento y de conocer, en todo momento, la localización del resto de nodos. Esto se traduce en aumento en el consumo de energía que puede provocar una sobrecarga significativa si existe un tráfico de datos elevado en la red.

Se puede concluir que el objetivo principal del protocolo *PEGASIS* es alargar el tiempo de vida de toda la red de sensores inalámbricos móviles y que existen mejoras del protocolo para corregir las desventajas y para solventar problemas que puedan surgir durante su uso.

2.6.4 Protocolo DREAM

DREAM (Distance Routing Effect Algorithm for Mobility) es un protocolo proactivo, ad-hoc y basado en la localización [10]. Su objetivo principal es dirigir la transmisión de datos hacia una única dirección hacia el nodo destino o estación base.

En este algoritmo, cada nodo tiene una tabla que contiene información sobre los demás nodos con el fin de tener una posible ruta en todo momento.

Además, todos los paquetes transmitidos por los nodos contienen información sobre el nodo fuente y sobre el nodo destino o estación base.

DREAM utiliza dos principios:

- Uno denominado, tal y como su nombre indica, Efecto de Enrutamiento a Distancia *(Distance Routing Effect).* Este principio consiste en que dependiendo de la distancia a la que se encuentre un nodo respecto del otro, la frecuencia de actualización de la tabla que contiene información sobre el otro nodo varía.

  Cuanto más cerca esté el nodo, mayor será la frecuencia de actualización de la tabla. Esto se debe a que existe movilidad en los nodos de la red, de forma que un movimiento producido en un nodo más cercano se traduce a un mayor cambio del que se produciría con el movimiento de un nodo más lejano.

- Principio basado en la Tasa de Movilidad, el cual consiste en que si un nodo se mueve a mayor velocidad debe avisar a los demás nodos sobre su nueva localización.

El funcionamiento de este protocolo se explica, de manera ordenada y aludiendo a la figura que se muestra, siguiendo los siguientes pasos:

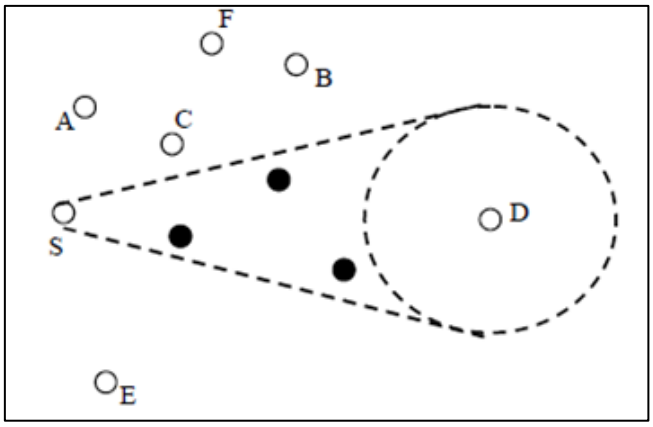

Figura 10. Esquema de funcionamiento del protocolo *DREAM*.

1) El nodo origen o fuente comienza la transmisión de datos. Este envía la información, al siguiente nodo, teniendo en cuenta dos condiciones:

   - Que se encuentre en el área encerrada por dos rectas divergentes, siendo tangentes al círculo cuyo centro corresponde al nodo destino. El ángulo de divergencia puede variar en función de lo que se desee limitar el área y la directividad para enviar los datos hacia la estación base.

   - Que sea el nodo más cercano.

   - Que la distancia nodo actual – nodo destino sea siempre menor que la había en el paso anterior.

2) Repetir la retransmisión de datos, siguiendo lo explicado en el apartado anterior, hasta que la información alcance la estación base.

Al igual que en el protocolo *PEGASIS*, el algoritmo *DREAM* utiliza también la técnica de agregación de datos.

En la figura 11 se puede observar un esquema de cómo quedaría la ruta del protocolo *DREAM* siguiendo los pasos de funcionamiento del algoritmo:

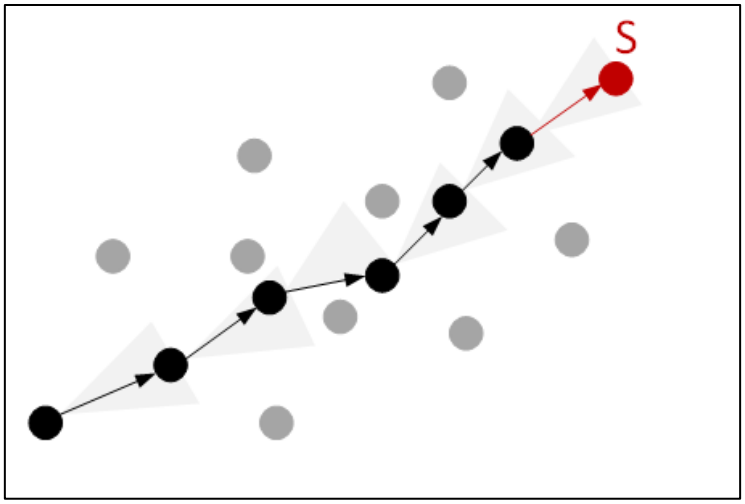

Figura 11. Esquema de enrutamiento del protocolo *DREAM*.

Con la implementación y ejecución del protocolo, se consigue con cada simulación un caso distinto al anterior debido a que el posicionamiento de los nodos en el escenario es aleatorio. A continuación, se muestra en la figura 12 la correspondiente a una de las simulaciones, donde la cruz roja corresponde a la estación base.

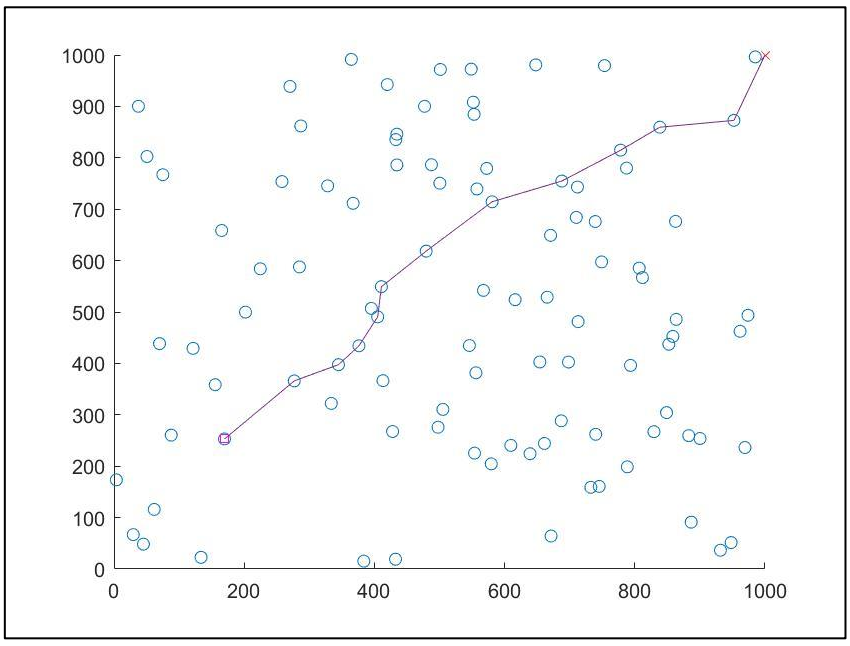

Figura 12. Simulación en *MatLab* del protocolo *PEGASIS*.

El primer nodo de donde empieza la transmisión de datos corresponde al nodo origen o fuente.

Para la simulación mostrada se ha utilizado un escenario, cuya área tiene unas dimensiones de 1000 x 1000 m^2.

El número de estaciones móviles es igual a 100. Se ha elegido, para este caso, un menor número de sensores con el fin de poder representar con mayor claridad el modelo y el funcionamiento del protocolo. En el código *MatLab* utilizado para calcular el consumo y eficiencia energética de cada protocolo, el número de estaciones móviles es igual a 150.

Es muy importante tener en cuenta que, para todos los demás protocolos, su implementación ha sido realizado para conseguir que todos los nodos de la red de sensores inalámbricos móviles transmitan sus datos a la estación base.

Pero en este protocolo la implementación para que todos los nodos de la red transmitan su información hasta la estación base era muy compleja.

Por lo tanto, para solventar este problema se pensó en mantener un sólo nodo como fuente y crear una única ruta. Además, para poder realizar una comparación del consumo y eficiencia energética adecuada con el resto de los protocolos, ha sido necesario cambiar sus códigos.

Esto se debe a que, si se comparase los demás algoritmos, que tienen en cuenta todos los nodos para que envíen sus datos, con el protocolo *DREAM*, que sólo tiene en cuenta los nodos que definen una ruta, el consumo energético del resto de algoritmos sería mucho mayor que el producido por *DREAM*. En cambio, con la corrección de códigos realizada se consigue una comparación de la eficiencia energética correcta y viable.

Al igual que todos los protocolos, *DREAM* presenta ventajas e inconvenientes.

Ventajas:

- Reduce notablemente el consumo energético gracias a la implementación de las dos técnicas principales del protocolo: Efecto de Enrutamiento a Distancia y el principio basado en la Tasa de Movilidad.

- Es un protocolo proactivo, y, por tanto, las tablas de localización deben de ser actualizadas constantemente, lo cual consume energía. Pero se obtiene una reducción en el consumo de energía debido a que estas tablas deben contener información únicamente sobre la localización de los demás nodos, y no es necesario una información completa de la tabla como ocurre en otros protocolos.

- La eficiencia energética, como se verá más adelante, es la mejor con respecto a los protocolos comentados en este proyecto.

Desventajas:

- Como se ha comentado al principio, *DREAM* es un protocolo proactivo y esto significa que la tabla de localización de todos los nodos tiene que ser actualizada de manera constante para que cada nodo de la red contenga esa información. Esto se traduce en un aumento significativo en el consumo de energía general en la red de sensores inalámbricos móviles.

- Y al igual que en *LEACH*, este protocolo también utiliza la técnica de agregación de datos, y, en consecuencia, el tiempo de demora es también elevado.

Ya se ha explicado que el protocolo DREAM retransmite los datos de los nodos en un rango direccional restringido sin necesidad de enviar un paquete de enrutamiento.

Es un protocolo muy útil para escenarios donde el área es grande ya que no tiene en cuenta que cada nodo tiene que poder enviar información de manera directa hasta la estación base. Pero, por otra parte, el consumo de energía aumenta notablemente para grandes áreas [10].

2.7 Funciones *MatLab*

Para la comparación de todos los protocolos desde el punto de vista de la eficiencia energética se han implementado diferentes funciones en *MatLab*, que ejecutándose de manera conjunta consiguen ese propósito.

2.7.1 Generación del posicionamiento de las estaciones móviles

Esta función (ANEXO C – 3.1) es la encargada de generar, de manera aleatoria, las posiciones de las estaciones móviles.

La función tiene como *inputs* el área del escenario, el número de estaciones móviles y unos límites que determinan la mínima y la máxima distancia entre dos estaciones móviles.

Por lo tanto, este código genera un escenario con un área dada, donde se sitúan aleatoriamente un número de estaciones móviles teniendo en cuenta los límites introducidos.

A continuación, se muestra en la figura 13 un ejemplo dada una simulación específica:

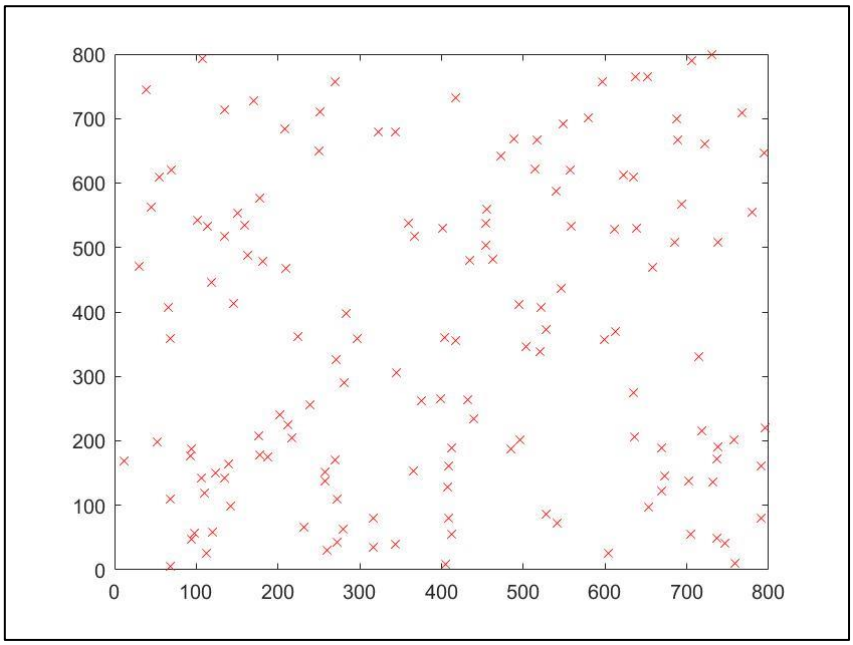

Figura 13. Generación del posicionamiento de los nodos simulado en *MatLab*.

Para la simulación en *MatLab* de la figura 13 se han utilizado 150 estaciones móviles, un escenario cuya área es de 800 x 800 m^2, una distancia mínima entre nodos de 10 m y una distancia máxima entre nodos de 600 metros.

2.7.2 Cálculo de las distancias entre nodos

Esta función (ANEXO C – 3.6) genera una matriz donde se presenta el cálculo de la distancia existente entre todos los nodos.

Para ello, simplemente le es necesario introducir como entrada una matriz que contenga la posición de todas las estaciones móviles y te devuelve la matriz comentada previamente.

2.7.3 Cálculo de la atenuación en el espacio libre

Esta función (ANEXO C – 3.2) crea una matriz que calcula la atenuación entre cada uno de los nodos de la red de sensores inalámbricos móviles en el espacio libre.

El cálculo se realiza simplemente mediante la introducción de una entrada cuyo parámetro es la distancia y utilizando la siguiente ecuación, que obtiene la pérdida básica de transmisión en el espacio libre entre enlaces punto a punto:

$$\boldsymbol{Lbf = 32.4 + 20 \times \log f + 20 \times \log d} \quad [\boldsymbol{dB}] \qquad \textbf{(3)}$$

Donde:

- L_{bf}: Pérdida básica de transmisión en el espacio libre (*dB*).
- *d*: Distancia (*km*).
- *f*: Frecuencia (*MHz*).

2.7.4 Cálculo de la capacidad

Esta función (ANEXO C – 3.7) genera una matriz dónde se presenta el cálculo de las capacidades entre cada uno de los dispositivos de la red. Para ello necesita como *inputs* el ancho de banda de cada canal de comunicación (*B*) y la relación señal a ruido (*SINR*).

Para la obtención de la capacidad se hace uso del *teorema de Shannon – Hartley*, el cual especifica la velocidad máxima a la que se puede transmitir la información a través de un canal de comunicaciones de un ancho de banda especificado y en presencia de ruido:

$$\boldsymbol{C = B \times \log_2(1 + SINR_w)} \quad [\boldsymbol{Mbps}] \qquad \textbf{(4)}$$

Donde:

- *C*: Capacidad (*Mbps*).
- *B*: Ancho de banda (*Hercios*).
- $SINR_w$: Relación señal – ruido (*Watts*).

Para calcular $SINR_w$ se siguen los siguientes pasos:

1) Calcular la atenuación de transmisión en el espacio libre *(PL – Pathloss)* entre todos los nodos utilizando la función comentada en el apartado (2.7.3).

2) Calcular la potencia recibida utilizando la siguiente ecuación:

$$\boldsymbol{Pr = Pt - PL} \quad [\boldsymbol{dBm}] \qquad \textbf{(5)}$$

3) Calcular la interferencia producida por la potencia recibida de otros transmisores en *dBm*.

4) Calcular el ruido térmico utilizando la siguiente ecuación:

$$N = Ancho\ de\ banda \times 4 \times 10^{-9} \quad [Watt] \qquad (6)$$

5) Transformar la interferencia calculada en el apartado 3 a *Watts* y sumarlo al ruido térmico obtenido previamente.

6) Transformar la suma de ruido e interferencia a *dBm (NI – Noise + Interference)*.

7) Aplicar la siguiente ecuación:

$$SINR = Pr - NI \quad [dBm] \qquad (7)$$

8) Transformar *SINR*, obtenido previamente, a *Watts* y usarlo en la ecuación (4) correspondiente al *teorema de Shannon – Hartley*.

2.7.5 Modelo del consumo energético

Para el cálculo de la potencia correspondiente al consumo de energía de los sensores de la red de sensores inalámbricos móviles se sigue el modelo *Lauridsen* [11] (ANEXO C – 3.5), el cual utiliza la siguiente ecuación:

$$P_{con} = P_{on} + m_{Rx} \times [P_{Rx} + P_{RxBB}(R_{Rx}) + P_{RxRF}(S_{Rx})] + m_{Tx} \times [P_{Tx} + P_{TxBB}(R_{Tx}) + P_{TxRF}(S_{Tx})] \quad [Watt] \qquad (8)$$

Donde:

- P_{on}: Potencia que consume el sensor inalámbrico móvil al estar en funcionamiento en *Watts*.

- m_{Rx}: Variable binaria que indica el estado de recepción del sensor inalámbrico móvil:

  - 0 – si no está recibiendo información.
  - 1 – si está recibiendo información.

- P_{Rx}: Potencia recibida por el sensor inalámbrico móvil en *Watts*.

- P_{RxBB}: Potencia banda base recibida en *Watts*. Este parámetro se calcula siguiendo una ecuación mostrada más adelante en la tabla (1).

- R_{Rx}: Tasa de datos recibido o capacidad de los sensores de la red en *Mbps*.

- P_{RxRF}: Potencia de radiofrecuencia recibida en *Watts*. Este parámetro se obtiene siguiendo una ecuación y condición mostradas más adelante en la tabla (1).

- S_{Rx}: Potencia recibida debido a la bajada de datos *(DL – Downlink)* en *dBms*.

- m_{Tx}: Variable binaria que indica el estado de transmisión del sensor inalámbrico móvil:

  - 0 – si no está transmitiendo información.
  - 1 – si está transmitiendo información.

- P_{Tx}: Potencia transmitida por el sensor inalámbrico móvil en *Watts*.

- P_{TxBB}: Potencia banda base transmitida en *Watts*. Este parámetro se calcula siguiendo una ecuación mostrada más adelante en la tabla (1).

- R_{Tx}: Tasa de datos transmitido o capacidad de los sensores de la red en *Mbps*.

- P_{TxRF}: Potencia de radiofrecuencia transmitida en *Watts*. Este parámetro se obtiene siguiendo una ecuación y condición mostradas más adelante en la tabla (1).

- S_{Tx}: Potencia transmitida debido a la subida de datos *(UL – Uplink)* en *dBms*.

A continuación, se muestra una tabla que muestra las ecuaciones y condiciones que se siguen para la obtención de los parámetros comentados anteriormente:

Parámetros	Polinomios	Condiciones
P_{TxRF}	$0.78 \times S_{Tx} + 23.6$	$S_{Tx} \leq 0.2$ dBm
P_{TxRF}	$17 \times S_{Tx} + 45.4$	0.2 dBm $\leq S_{Tx} \leq 11.4$ dBm
P_{TxRF}	$5.9 \times S_{Tx}^2 - 118 \times S_{Tx} + 1195$	11.4 dBm $< S_{Tx}$
P_{TxBB}	0.62	
P_{RxRF}	$-0.04 \times S_{Rx} + 24.8$	$S_{Rx} \leq$ -52.5 dBm
P_{RxRF}	$-0.11 \times S_{Rx} + 7.86$	$S_{Rx} >$ -52.5 dBm
P_{RxBB}	$0.97 \times R_{Rx} + 8.16$	
P_{ON}	853, 29.9, 25.1	Subsistema celular, T_x, R_x activos

Tabla 1. Modelo Lauridsen [6] para el cálculo del consumo energético.

El esquema que se ha seguido para el cálculo del consumo de energía es el que se muestra a continuación:

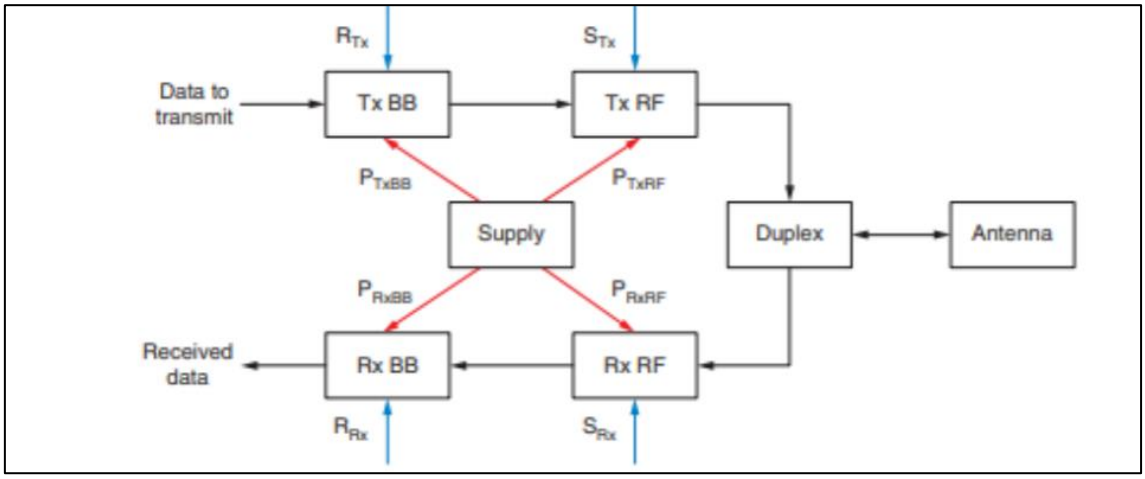

Figura 14. Esquema del modelo de consumo de energía utilizado [6].

Una vez se ha obtenido la potencia correspondiente al consumo de energía (P_{con}), se aplica la siguiente ecuación para, finalmente, obtener la energía consumida (E):

$$\boldsymbol{E = T \times P_{con}} \quad [J] \qquad \textbf{(9)}$$

Donde T es el tiempo de transmisión, cuya unidad son los segundos.

2.7.6 Generación de movilidad en los sensores inalámbricos móviles

Para la generación de movilidad en todos los nodos de la red de sensores inalámbricos móviles se ha utilizado un modelo ya creado e implementado en *MatLab*, el cual se denomina *Random Waypoint Mobility Model* [12] (ANEXO C – 3.12).

Este es un modelo que afecta, de manera aleatoria, al cambio producido en la localización, velocidad y aceleración, según transcurre el tiempo, para el movimiento de las estaciones móviles que forman la red. Para ser más específicos, el destino, la velocidad y la dirección se eligen de forma aleatoria e independientemente de otros nodos.

En la Figura 15 se muestra un ejemplo dada una simulación específica:

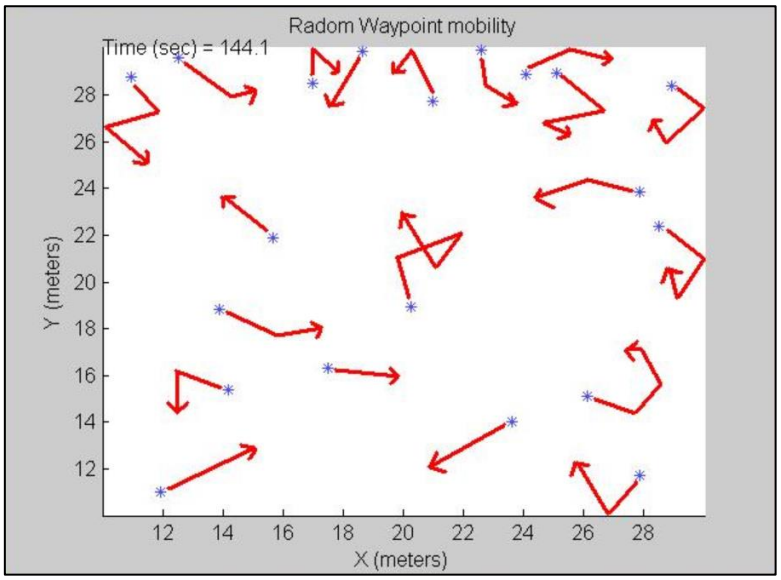

Figura 15. Generación de movilidad en los nodos simulado en *MatLab* [7].

El movimiento de los nodos sigue el funcionamiento explicado a continuación:

1) Cada nodo comienza pausado durante un intervalo de tiempo determinado.

2) Posteriormente, el nodo selecciona una dirección y sentido aleatorios (destino) en el área de simulación y una velocidad aleatoria entre una velocidad mínima, en este caso 0.2 m/s, y alguna velocidad máxima, en este caso 10 m/s.

   Se ha aplicado este rango de velocidades para poder simular tanto movimiento de personas como todo tipo de transportes, teniendo en cuenta sus diferentes velocidades.

3) El nodo se mueve hacia su destino marcado y nuevamente hace una pausa por un período fijado a la espera de que le asignen un nuevo destino, con el fin de repetir el segundo paso explicado previamente.

4) Este proceso se repite para el período de tiempo determinado en la simulación.

2.7.7 Cálculo de la potencia de señal recibida

Esta función (ANEXO C – 3.8) calcula una matriz con los valores de la potencia de señal recibida *(RSS – Received Signal Strength)* por cada dispositivo correspondiente a la red de sensores inalámbricos móviles.

Para ello requiere de las entradas de los siguientes parámetros:

- Número de estaciones móviles.
- Matriz con la potencia de transmisión de cada estación móvil.
- Matriz con la atenuación en el espacio libre producida en cada canal de comunicación de las estaciones móviles.

2.7.8 Control de potencia transmitida

Esta función (ANEXO C – 3.9) es la encargada de definir un límite de potencia transmitida a los dispositivos móviles inalámbricos.

Para poder asignar un control de potencia transmitida necesita los siguientes *inputs*:

- Potencia de ruido producido en cada canal de comunicación de las estaciones móviles.
- Atenuación en el espacio libre producida en los canales de comunicación de las estaciones móviles.
- Umbral definido de la relación señal – ruido de los sensores.

Es una mejora añadida al proyecto que tiene el objetivo de reducir el consumo de energía y, en consecuencia, alargar el tiempo de vida de la red.

2.7.9 Elección de los clústeres

Esta función (ANEXO C – 3.10) es la encargada de seleccionar los clústeres de la red de sensores inalámbricos móviles en el protocolo *LEACH*.

Para una correcta asignación de los clústeres, se requiere de los siguientes parámetros como entradas:

- Posición de las estaciones móviles en la red.
- Número de estaciones móviles.
- Número de clústeres definido.

La función devuelve la posición de los nodos que tendrán la función de clústeres en la red siguiendo la ecuación (2) que se utiliza para la elección de clústeres en el protocolo *LEACH*.

2.7.10 Asociación de los nodos que transmiten a los clústeres

Esta función (ANEXO C – 3.11), utilizada en el protocolo *LEACH*, es la encargada de asignar a cada estación móvil su clúster correspondiente al que tiene que transmitirle la información.

Para ello simplemente requiere de los siguientes inputs:

- Posición de las estaciones móviles en la red.
- Posición de los nodos seleccionados como clústeres.

La función devuelve como *outputs*:

- Para cada nodo su correspondiente clúster más cercano al que debe de transmitir los datos.
- La distancia existente entre la estación móvil y su clúster, para que este pueda ser posteriormente procesado para el cálculo del consumo energético.

2.8 Resultados

Ejecutando todas las funciones y código *MatLab* de manera cooperativa y conjunta se obtienen los siguientes resultados, los cuales se presentan desde dos puntos de vista diferentes:

- Dispositivos de la red de sensores inalámbricos en estado estático (sin movilidad).
- Con movilidad en la red de sensores inalámbricos.

Para cada resultado obtenido, se ha realizado un total de 500 ejecuciones que arrojan como resultados finales un promedio.

2.8.1 Resultados obtenidos en el caso estático

A continuación, se muestra en la figura 16 el consumo de energía en *Joules* como eje *Y* y en el eje *X* la dimensión del área del escenario que varía entre 700 y 1000 m^2, para el caso de que todos los sensores transmiten la información recogida:

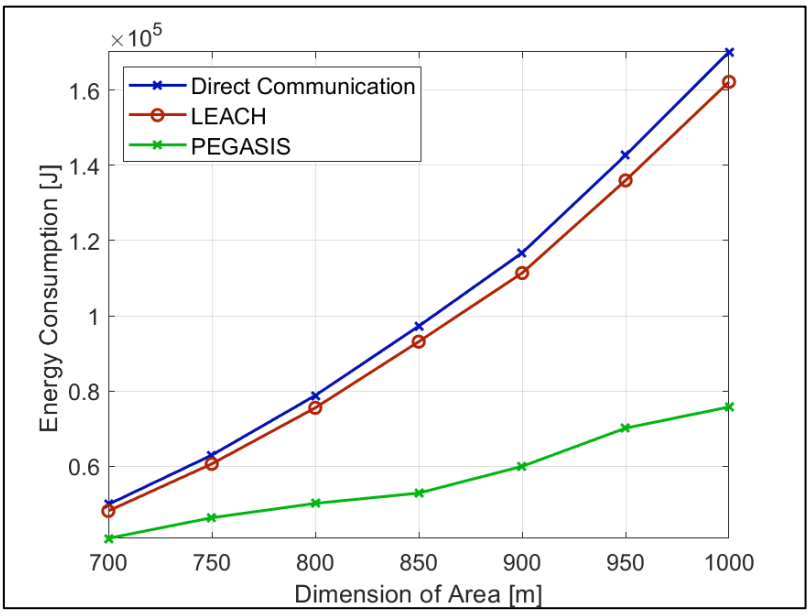

Figura 16. Consumo de energía donde todos los nodos transmiten datos y para una red estática.

Como se puede observar en la simulación anterior, no se representa el consumo de energía del protocolo *DREAM* debido a lo ya explicado: en este protocolo, su implementación para que todos los nodos de la red transmitan su información hasta la estación base era muy compleja.

Con la solución a este problema, que consistía en mantener un sólo nodo como fuente y crear una única ruta, además de cambiar los códigos *MatLab* del resto de protocolos para adaptarse a esta situación, se obtiene el siguiente resultado:

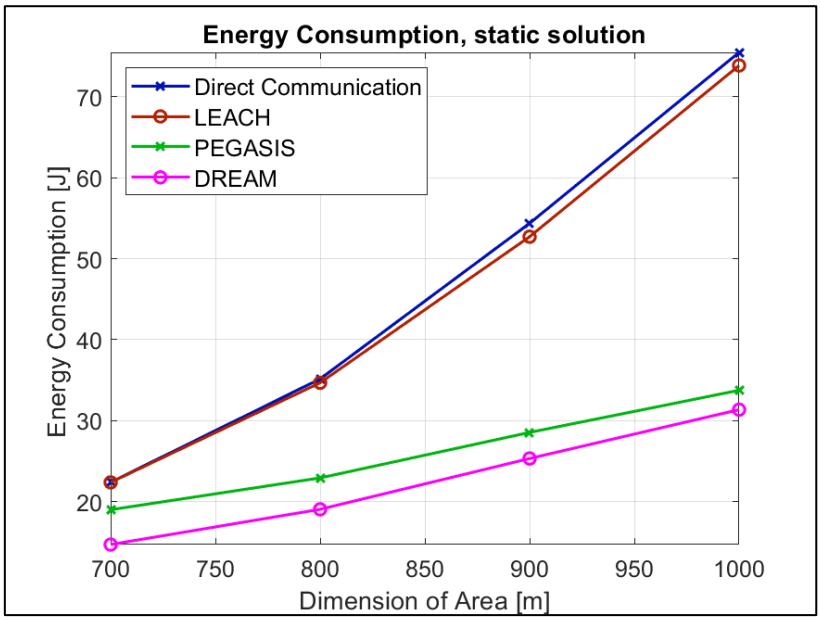

Figura 17. Consumo de energía con un único nodo fuente transmitiendo y para una red sin movilidad.

Si comparamos los resultados de la figura 16 con la 17, se observa que el consumo de energía es mucho más elevado, por el hecho de que en el primer caso todos los nodos de la red transmiten su información hasta la estación base, mientras que, en el segundo caso, únicamente un nodo de todo el conjunto de dispositivos de la red actúa como nodo fuente, y, por lo tanto, es el único que transmite información.

En base a los resultados obtenidos se determina que:

- El protocolo de *Comunicación Directa* presenta el peor de los casos.
- *LEACH* mejora significativamente el consumo de energía respecto al protocolo de *Comunicación Directa*.
- *PEGASIS* muestra una mejora que reduce el consumo de energía en más de la mitad con respecto al protocolo *LEACH*.
- *DREAM* aún presenta un consumo de energía menor que el protocolo *PEGASIS*. Por lo tanto, es el mejor algoritmo desde el punto de vista de la eficiencia energética.

2.8.2 Resultados obtenidos implementada la movilidad en la red

En este apartado se muestran los resultados obtenidos con la implementación de la generación de movilidad en los nodos de la red de sensores inalámbricos móviles.

A continuación, se observa la figura 18 que muestra el consumo de energía en *Joules* como eje *Y* y en el eje *X* la dimensión del área del escenario que varía entre 700 y 1000 m^2, para el caso de que exista un único nodo fuente transmitiendo información:

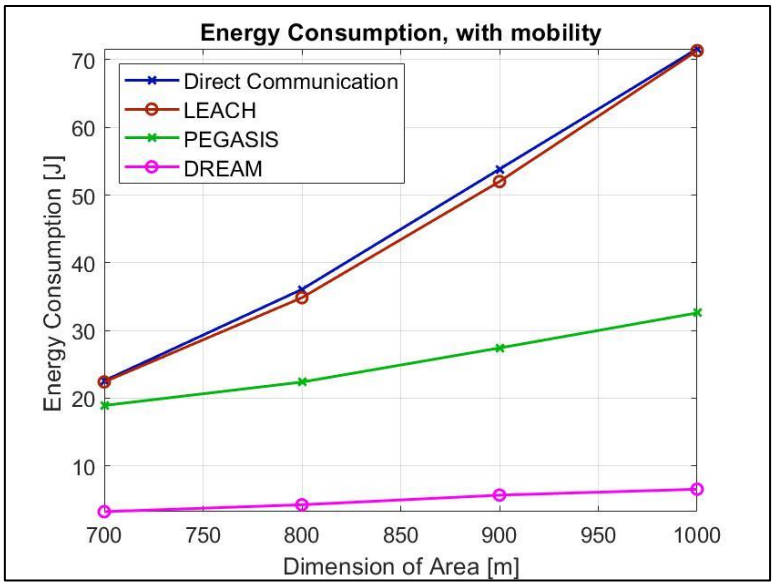

Figura 18. Consumo de energía con un único nodo fuente transmitiendo y para una red con movilidad.

Y si en lugar de variar las dimensiones del área del escenario simulado se analiza el consumo de energía desde el punto de vista del transcurso del tiempo, se obtiene la siguiente gráfica:

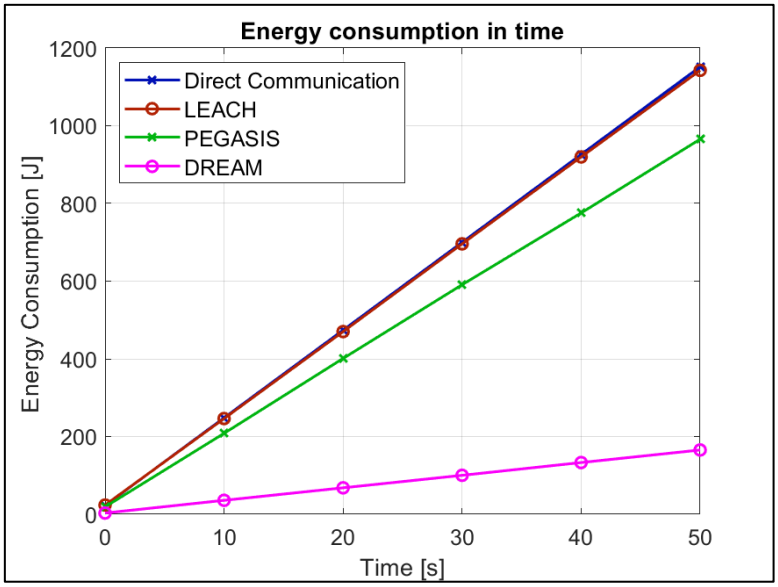

Figura 19. Consumo de energía, según transcurre el tiempo, con un único nodo fuente y para una red con movilidad.

En ambos resultados obtenidos, se observa como el protocolo *DREAM* mejora notablemente en el aspecto de la eficiencia energética con la implementación de movilidad en la red de sensores inalámbricos móviles. Eso se debe a que el funcionamiento de este algoritmo se adapta mucho mejor a los movimientos y cambios de posición de los nodos.

2.9 Conclusiones y trabajo futuro

En las redes de sensores inalámbricos móviles, los protocolos de enrutamiento tienen como principal premisa la transmisión segura sin pérdidas de los datos y reducir el consumo de energía con el fin de alargar la vida de toda la red.

Como se ha explicado en este documento, los protocolos se pueden dividir básicamente dependiendo de su modo de operación y de su topología de la red.

En función de su modo de operación están los algoritmos proactivos, reactivos e híbridos. Y dependiendo de la topología de la red, los protocolos se pueden dividir en planos, jerárquicos y basado en la localización.

Actualmente existen una gran cantidad de protocolos distintos aplicables a las diferentes redes de sensores inalámbricos móviles.

En este proyecto se han seleccionado y estudiado cuatro protocolos: *Comunicación Directa*, *LEACH*, *PEGASIS* y *DREAM*. Obteniendo las siguientes conclusiones:

- El protocolo *LEACH* reduce el consumo de energía en comparación con el de *Comunicación Directa* y esto se debe principalmente los clústeres se dividen aleatoriamente, lo que da como resultado una distribución desigual de los clústeres.

- Los protocolos *PEGASIS* y *DREAM* muestran una gran ganancia desde el punto de vista de energía consumida. Los múltiples saltos requeridos en *PEGASIS* producen una gran demora que puede presentar un gran inconveniente. *DREAM* también sufre con el problema de la demora, pero en este caso, con la ruta aplicada se consigue una reducción de la demora bastante notable, lo que nos lleva a que, para conseguir una mayor eficiencia energética, la selección de este algoritmo será una buena propuesta, sobre todo, para escenarios cuya área no sea de muy grandes dimensiones.

En este proyecto las simulaciones realizadas se han llevado a cabo en un escenario y condiciones específicos, dándonos unos resultados aplicables a esos casos. Pero se debe tener en cuenta que para diferentes posibilidades donde pueden variar el área del escenario, número de estaciones móviles y de estaciones base, rango de potencia transmitida en los sensores y la movilidad en los nodos, entre otras, se obtendrían unos resultados distintos.

Por lo tanto, no se puede generalizar el uso de un protocolo de enrutamiento, sino que para cada caso se ha de analizar y estudiar las distintas posibilidades para obtener cual sería el protocolo que mejor se adaptase con el principal objetivo de obtener una mayor eficiencia energética y alargar la vida de la red.

Es posible que, en un futuro, con el rápido avance tecnológico se consigan unos procesadores y *hardware* mucho más eficaces y rápidos que permitiesen a los sensores recopilar una mayor cantidad de datos, no sólo que hayan de ser transmitidos, sino que puedan ser usados para realizar un enrutamiento de la información más eficaz e inteligente.

BIBLIOGRAFÍA

[1] MathWorks, "Audio System Toolbox", https://es.mathworks.com/products/audio-system.html [Online].

[2] Rabiner, Lawrence R.; Gold, Bernard (1975). Theory and application of digital signal processing. Englewood Cliffs, N.J.: Prentice-Hall. pp. 63–67. ISBN 0-13-914101-4.

[3] Oppenheim, Alan V., Willsky, Alan S., and Young, Ian T.,1983: Signals and Systems, p. 256 (Englewood Cliffs, New Jersey: Prentice-Hall, Inc.) ISBN 0-13-809731-3.

[4] Akyildiz IF, Su W, Sankarasubramaniam Y, Cayirci E (2002). A survey on sensor networks. IEEE Commun. Mag. 40(8):102-114.

[5] Ramasamy, V., 2017. Mobile Wireless Sensor Networks: An Overview. n Wireless Sensor Networks-Insights and Innovations. InTech.

[6] García Villalba, L.J., Sandoval Orozco, A.L., Triviño Cabrera, A. and Barenco Abbas, C.J., 2009. Routing protocols in wireless sensor networks. Sensors, 9(11), pp. 8399-8421.

[7] AL-KARAKI, J. N. and KAMAL, A. E. Routing techniques in wireless sensor networks: a survey. Wireless communications, IEEE, Vol. 11, No. 6, 2004, pp. 628.

[8] J. Habibi, A. Ghrayeb, A.G. Aghdam. Energy-efficient cooperative routing in wireless sensor networks: a mixed-integer optimization framework and explicit solution. IEEE Trans. Commun., 61 (8) (2013), pp. 3424-3437.

[9] Manal Abdullah, Aisha Ehsan, 2014 Routing protocols for wireless sensor networks: classifications and challenges. Quest journals journal of electronics and communication engineering research volume 2 ~ issue 2 pp: 05-15 issn(online) : 2321-5941.

[10] "A distance routing effect algorithm for mobility (DREAM)" in Basagni, Stefano; Imrich Chlamtac; Violet R. Syrotiuk; Barry A. Woodward (1998). International Conference on Mobile Computing and Networking Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking. New York: ACM Press. pp. 76–84. doi:10.1145/288235.288254. ISBN 1-58113-035-X.

[11] Lauridsen, Mads, et al. "An empirical LTE smartphone power model with a view to energy efficiency evolution." Intel Technology Journal 18.1 (2014): 172-193.

[12] MathWorks; Mathieu Boutin. "Random Waypoint mobility model" https://es.mathworks.com/?s_tid=gn_logo [Online].

ANEXOS

1. ANEXO A: Código base del capítulo 2 para una red estática

Los protocolos de *Comunicación Directa* y *LEACH* se hallan implícitos en el siguiente código base. Para los algoritmos de enrutamiento restantes se han utilizado funciones a parte, las cuales son adjuntadas más adelante en los anexos.

```
function[Total_energy_LEACH_random_a,Total_energy_Direct_a,Total_energ
y_pegasis_a,Total_energy_dream_a,clustersss]=main_over_drops_no_mob(Ar
ea)

%% Declaration of Main simulations parameters
Number_MS = 150;  % Number of CUE in the system
Number_CH = 10;   % Number of cluster heads in system
const(1) = 4;     % minimum constrain in meters
const(2) = 1400;  % maximum constrain in meters
f = 2;            % carrier frequency in GHz
Pt_full = 20;     %transmission power in dBm
BW = 20*10.^6;    % channel bandwidth
Data = 20e6;      % packet size

% interference = 1; % 1 - on, 0 - off
sim_time =Number_MS/Number_CH ; % rounds
SINRth = -10;     %treshold for computing the transmitting power, in dB


%% calculation of noise
N_W = (BW*4*10.^-12)/10.^9;   % noise [W]
N_dBm = 10*log10(N_W/0.001);  % noise [dBm]

%% matrix for faster computation
Total_energy_LEACH_random = zeros(1,length(Area));
Total_energy_Direct = zeros(1,length(Area));
Total_energy_Direct_PC = zeros(1,length(Area));
Total_energy_pegasis = zeros(1,length(Area));
Total_energy_dream = zeros(1,length(Area));

clustersss=zeros(sim_time,Number_CH);

%% simulation
for a = 1:length(Area)

    Area_x = Area(a); % Definition of Area in meters (x-axes)
    Area_y = Area(a); % Definition of Area in meters (y-axes)

    %%%% generation of position
    Pos_MS = MS_position(Area_x,Area_y,Number_MS,const); % position of
     MS
    Pos_sink = [Area_x; Area_y]; % position of sink
    used_clusters=[];
    for b = 1:sim_time

        %% dream
        [chain,pos_x_source] = dream2 (Area_x, Number_MS, Pos_MS);
        PL_first = PL_dream (chain,Pos_MS,Area_x,f);
        Pr_1 = Pt_full - PL_first;
        SNR_first_chain = Pt_full - PL_first - N_dBm; % in dB
        SNR_1 = 10.^(SNR_first_chain/10);
        C_1 = capacity(BW,SNR_1);
        Data1 = Data:Data:length(chain)*Data;
        Pcon_dream_first = Pcon_function(Pt_full,Pr_1,C_1,1,0,1,1);
```

```
T_dream_first = Data1./(C_1*1e6); % transmission time in
seconds
Energy_DREAM1 = T_dream_first.*Pcon_dream_first;
Total_energy_dream(a) = Total_energy_dream(a) +
sum(Energy_DREAM1);

%% direct transmition with power control
% distance and pathloss are the same as in direct with full
power
d_to_sink = distance_2_points(Pos_sink,Pos_MS);
Path_loss = pathloss(Number_MS,d_to_sink,f);
Pt = power_control (N_dBm, Path_loss, SINRth);
Pt_source_to_sink = Pt(pos_x_source);
Pr = Pt - Path_loss;
Pr_source_to_sink = Pr(pos_x_source);
SNR_to_sink = Pt - Path_loss - N_dBm;        % in dB
SNR = 10.^(SNR_to_sink/10);
C_to_sink = capacity(BW,SNR);                % capacity to
sink
C_source_to_sink = C_to_sink(pos_x_source);  % capacity from
source to sink

%%%% calculation of power consumption
Pcon_Direct_PC =
Pcon_function(Pt_source_to_sink,Pr_source_to_sink,C_source_to_
sink,1,0);
%%%% calculation of consumed energy
T_Direct_PC = Data./(C_source_to_sink*1e6); % transmission
time in seconds
Energy_Direct_PC = T_Direct_PC.*Pcon_Direct_PC;
Total_energy_Direct_PC(a) = Total_energy_Direct_PC(a) +
sum(Energy_Direct_PC);

%% direct transmition with full power
Pr = Pt_full - Path_loss;
Pr_source_to_sink = Pr(pos_x_source);
SNR_to_sink = Pt_full - Path_loss - N_dBm;   % in dB
SNR = 10.^(SNR_to_sink/10);
C_to_sink = capacity(BW,SNR);                % capacity to
sink
c_source_to_sink = C_to_sink(pos_x_source);  % capacity from
source to sink

%%%% calculation of power consumption
Pcon_Direct =
Pcon_function(Pt_full,Pr_source_to_sink,C_source_to_sink,1,0);
%%%% calculation of consumed energy
T_Direct = Data./(C_source_to_sink*1e6); % transmission time
in seconds
Energy_Direct = T_Direct.*Pcon_Direct;
Total_energy_Direct(a) = Total_energy_Direct(a) +
sum(Energy_Direct);

%% LEACH random choose of cluster - full power
probability=zeros(1,Number_MS);
for i_proc=1:Number_MS

    probability(i_proc)=1/(Number_MS-((b-1)*Number_CH));
    if ismember(i_proc,used_clusters )==1
        probability(i_proc)=0;
```

```
    end

end
if sum(probability)==0
    used_clusters=[];
end

cluster =
choose_point_with_probability2(Pos_MS,Number_CH,probability);
used_clusters=[used_clusters,cluster];
if a==1
    clustersss(b,:)=cluster;
end

[d_to_CH ,CH_assoc] = MS_association (Pos_MS, cluster);
Path_loss = pathloss(1,d_to_CH(pos_x_source),f);
Pr1 = Pt_full - Path_loss;
SNR = Pr1 - N_dBm;
SNR_to_CH = 10.^(SNR/10);
C_to_CH = capacity(BW,SNR_to_CH);
pom = CH_assoc(pos_x_source);
pom = cluster(pom);
d_to_sink = distance_2_points(Pos_sink,Pos_MS(:,pom));
Path_loss = pathloss(1,d_to_sink,f);
Pr = Pt_full-Path_loss;
SNR = Pr - N_dBm;
SNR_to_sink = 10.^(SNR/10);
C_to_sink=capacity(BW,SNR_to_sink);
C_source_to_CH_to_sink = [C_to_CH C_to_sink];
Pr_to_sink = [Pr1 Pr];
if d_to_CH(pos_x_source) == 1e5
    C_source_to_CH_to_sink = C_to_sink;
    Pr_to_sink = Pr;
end
Pcon_LEACH_random =
Pcon_function(Pt_full,Pr_to_sink,C_source_to_CH_to_sink,1,0,1,
1);
T_LEACH_random = Data./(C_source_to_CH_to_sink*1e6); %
transmission time in seconds
Energy_LEACH_random = T_LEACH_random.*Pcon_LEACH_random;
Total_energy_LEACH_random(a) = Total_energy_LEACH_random(a) +
sum(Energy_LEACH_random);


%% pegasis
[first_chain,second_chain] = pegasis2 (Area_x, Number_MS,
Pos_MS);
PL_first = PL_pegasis (first_chain,Pos_MS,Area_x,f);
if ismember(pos_x_source,first_chain)==1
Pr_1 = Pt_full - PL_first;
SNR_first_chain = Pt_full - PL_first - N_dBm; % in dB
SNR_1 = 10.^(SNR_first_chain/10);
C_1 = capacity(BW,SNR_1);
C_source = 0;
for k = 1:length(C_1)
C_source = C_source + C_1(k);
end
p_r_used=Pr_1(find(first_chain==pos_x_source):end);
C_used=C_1(find(first_chain==pos_x_source):end);
```

```
            Pcon_PEGASIS_first =
            Pcon_function(Pt_full,p_r_used,C_used,1,0,1,1);
            T_PEGASIS_first = Data./(C_used*1e6); % transmission time in
            seconds
            Energy_PEGASIS1 = T_PEGASIS_first.*Pcon_PEGASIS_first;
            Total_energy_pegasis(a) = Total_energy_pegasis(a) +
            sum(Energy_PEGASIS1);

            else

                PL_second = PL_pegasis (second_chain,Pos_MS,Area_x,f);
                Pr_2 = Pt_full - PL_second;
                SNR_second_chain = Pt_full - PL_second - N_dBm; % in dB
                SNR_2 = 10.^(SNR_second_chain/10);
                C_2 = capacity(BW,SNR_2);
                C_source = 0;
                for k = 1:length(C_2)
                    C_source = C_source + C_2(k);
                end

            p_r_used=Pr_2(find(second_chain==pos_x_source):end);
            C_used=C_2(find(second_chain==pos_x_source):end);
            Pcon_PEGASIS_first =
            Pcon_function(Pt_full,p_r_used,C_used,1,0,1,1);
            T_PEGASIS_first = Data./(C_used*1e6); % transmission time in
            seconds
            Energy_PEGASIS2 = T_PEGASIS_first.*Pcon_PEGASIS_first;
            Total_energy_pegasis(a) = Total_energy_pegasis(a) +
            sum(Energy_PEGASIS2);
            end
        end
    end

%% computation of the average energy
Total_energy_LEACH_random_a = Total_energy_LEACH_random/sim_time;
Total_energy_Direct_a = Total_energy_Direct/sim_time;
Total_energy_pegasis_a = Total_energy_pegasis/sim_time;
Total_energy_dream_a = Total_energy_dream/sim_time
end
```

1.1 Llamada y muestreo de resultados correspondiente al código del Anexo A

Nombre del fichero: *all_main_no_mob.m*

```
clc
close all
clear all
drops_numb=500;
Area = 700:100:1000; % area size
Total_energy_LEACH_random_a=zeros(drops_numb,length(Area));
Total_energy_Direct_a=zeros(drops_numb,length(Area));
Total_energy_pegasis_a=zeros(drops_numb,length(Area));
Total_energy_dream_a=zeros(drops_numb,length(Area));
for i=1:drops_numb
      disp(i);
```

```
      [Total_energy_LEACH_random_a(i,:),Total_energy_Direct_a(i,:),Tot
      al_energy_pegasis_a(i,:),Total_energy_dream_a(i,:),clustersss]=m
      ain_over_drops_no_mob(Area);
end

 y1=mean(Total_energy_LEACH_random_a);
 y2=mean(Total_energy_Direct_a);
 y3=mean(Total_energy_pegasis_a);
 y4=mean(Total_energy_dream_a);

figure
colour_darkblue = [1 17 181] ./ 255;
p1=plot (Area, y2,'Color', colour_darkblue);
set(p1,'Marker','x');
set(p1,'LineStyle','-');
hold on

darkred=[178 34 4] ./ 255;
p2=plot (Area, y1,'Color', darkred);
set(p2,'Marker','o');
set(p2,'LineStyle','-');

colour_green = [1 181 17] ./ 255;
p3=plot (Area, y3,'Color', colour_green);
set(p3,'Marker','x');
set(p3,'LineStyle','-');

fuchsia=[250 0 250] ./ 255;
p4=plot (Area, y4,'Color', fuchsia);
set(p4,'Marker','o');
set(p4,'LineStyle','-');
limyd=min(min(min(y1(1),y2(:,1)),y3(1)),y4(1));
limyu=max([max(y1) max(y2) max(y3) max(y4)]);
xlim([Area(1) Area(end)]);
ylim([limyd limyu]);

grid on

set(p1,'LineWidth',1.5);
set(p2,'LineWidth',1.5);
set(p3,'LineWidth',1.5);
set(p4,'LineWidth',1.5);

h_legend=legend ('Direct Communication','LEACH','PEGASIS','DREAM');
set(h_legend,'FontSize',12,'Location','northwest');
title('Energy Consumption, static solution','FontSize',25);
ylabel('Energy Consumption [J] ','FontSize',17);
xlabel('Dimension of Area [m]', 'FontSize', 17);grid on;

 set(gca,'FontWeight','normal',...
    'FontSize',12);

SaveFigs = 1;
if SaveFigs == 1
    hgsave(sprintf('Leach'));
    saveas(gca,sprintf('Leach.png'));
    print -dtiff -r300 Leach;            %HighResolution Fig
end
ss=2;
```

2. ANEXO B: Código base del capítulo 2 para una red con movilidad

Los protocolos de *Comunicación Directa* y *LEACH* se hallan implícitos en el siguiente código base. Para los algoritmos de enrutamiento restantes se han utilizado funciones a parte, las cuales son adjuntadas más adelante en los anexos.

```
function [Total_energy_LEACH_random_b, Total_energy_Direct_b,
Total_energy_pegasis_b, Total_energy_dream_b,Efficiency_DIRECT_a,
Efficiency_PEGASIS_a, Efficiency_LEACH_a, Efficiency_DREAM_a,
Total_energy_LEACH_random_a, Total_energy_Direct_a,
Total_energy_pegasis_a, Total_energy_dream_a, clustersss] =
main_over_drops_final(Area,sim_time)
%% Declaration of Main simulations parameters
Number_MS = 150;  % Number of CUE in the system
Number_CH = 10;   % Number of cluster heads in system
const(1) = 4;     % minimum constrain in meters
const(2) = 1400;  % maximum constrain in meters
f = 2;            % carrier frequency in GHz
Pt_full = 20;     %transmission power in dBm
BW = 20*10.^6;    % channel bandwidth
Data = 20e6;      % packet size

%% calculation of noise
N_W = (BW*4*10.^-12)/10.^9;   % noise [W]
N_dBm = 10*log10(N_W/0.001);  % noise [dBm]

%% matrix for faster computation
Total_energy_LEACH_random = zeros(1,length(Area));
Total_energy_Direct = zeros(1,length(Area));
Total_energy_pegasis = zeros(1,length(Area));
Total_energy_dream = zeros(1,length(Area));

Total_energy_LEACH_random_b = zeros(1,sim_time+1);
Total_energy_Direct_b = zeros(1,sim_time+1);
Total_energy_pegasis_b = zeros(1,sim_time+1);
Total_energy_dream_b = zeros(1,sim_time+1);

Efficiency_DIRECT = zeros(1,length(Area));
Efficiency_PEGASIS = zeros(1,length(Area));
Efficiency_LEACH = zeros(1,length(Area));
Efficiency_DREAM = zeros(1,length(Area));

clustersss=zeros(sim_time,Number_CH);
%% simulation
for a = 1:length(Area)

    Area_x = Area(a); % Definition of Area in meters (x-axes)
    Area_y = Area(a); % Definition of Area in meters (y-axes)

    %%%% generation of position
    Pos_MS = MS_position(Area_x,Area_y,Number_MS,const); % position of
     MS
    Pos_sink = [Area_x; Area_y]; % position of sink
    coordinates_movers = mobility(Pos_MS, Area_x, sim_time);

    for b = 1:sim_time+1
        used_clusters=[];
        for i = 1:length(Pos_MS)
            Pos_MS(1,i)=coordinates_movers(1,b,i);
```

```
    Pos_MS(2,i)=coordinates_movers(2,b,i);
    if Pos_MS(1,i) == Area_x
        Pos_MS(1,i) = Area_x - 0.001;
    elseif Pos_MS(2,i) == Area_x
        Pos_MS(1,i) = Area_x - 0.001;
    end
end
%% dream
if b == 1
    [chain,pos_x_source] = dream2 (Area_x,Number_MS,Pos_MS);
else
    [chain] = dream3 (Area_x,Number_MS,Pos_MS,pos_x_source);
end
PL_first = PL_dream (chain,Pos_MS,Area_x,f);
Pr_1 = Pt_full - PL_first;
SNR_first_chain = Pt_full - PL_first - N_dBm; % in dB
SNR_1 = 10.^(SNR_first_chain/10);
C_1 = capacity(BW,SNR_1);
[size,~] = one_bit_transmit (C_1);
Pcon_dream = Pcon_function(Pt_full,Pr_1,C_1,1,0,1,1);
T_dream = Data./(C_1*1e6); % transmission time in seconds
T_bit = size./(C_1*1e6);
Energy_DREAM = T_dream.*Pcon_dream;
Energy_DREAM_1bit = T_bit.*Pcon_dream;
Total_energy_dream(a)=Total_energy_dream(a)+sum(Energy_DREAM);
Efficiency_DREAM(a) =
Efficiency_DREAM(a)+(size/sum(Energy_DREAM_1bit));
if a == 1
    Total_energy_dream_b(b) = sum(Energy_DREAM);
end

%% direct transmition with full power
d_to_sink=distance_2_points(Pos_sink,Pos_MS(:,pos_x_source));
Path_loss = pathloss(1,d_to_sink,f);
Pr = Pt_full - Path_loss;
SNR_to_sink = Pt_full - Path_loss - N_dBm; % in dB
SNR = 10.^(SNR_to_sink/10);
C = capacity(BW,SNR); % capacity to sink
%%%% calculation of power consumption
Pcon_Direct = Pcon_function(Pt_full,Pr,C,1,0);
%%%% calculation of consumed energy
[size,~] = one_bit_transmit (C);
T_Direct = Data/(C*1e6); % transmission time in seconds
T_bit = size./(C*1e6);
Energy_Direct = T_Direct*Pcon_Direct;
Energy_DIRECT_1bit = T_bit*Pcon_Direct;
Total_energy_Direct(a) = Total_energy_Direct(a) +
sum(Energy_Direct);
Efficiency_DIRECT(a) = Efficiency_DIRECT(a) +
size/sum(Energy_DIRECT_1bit);
if a == 1
    Total_energy_Direct_b(b) = sum(Energy_Direct);
end

%% LEACH random choose of cluster - full power
probability=zeros(1,Number_MS);
for i_proc=1:Number_MS
    probability(i_proc)=1/(Number_MS-((1-1)*Number_CH));
    if ismember(i_proc,used_clusters )==1
        probability(i_proc)=0;
```

```
    end
end
cluster =
choose_point_with_probability2(Pos_MS,Number_CH,probability);
if a==1
    clustersss(1,:)=cluster;
end
[d_to_CH ,CH_assoc] = MS_association (Pos_MS, cluster);
Path_loss = pathloss(1,d_to_CH(pos_x_source),f);
Pr1 = Pt_full - Path_loss;
SNR = Pr1 - N_dBm;
SNR_to_CH = 10.^(SNR/10);
C_to_CH = capacity(BW,SNR_to_CH);
pom = CH_assoc(pos_x_source);
pom = cluster(pom);
d_to_sink = distance_2_points(Pos_sink,Pos_MS(:,pom));
Path_loss = pathloss(1,d_to_sink,f);
Pr = Pt_full-Path_loss;
SNR = Pr - N_dBm;
SNR_to_sink = 10.^(SNR/10);
C_to_sink=capacity(BW,SNR_to_sink);
C_source_to_CH_to_sink = [C_to_CH C_to_sink];
Pr_to_sink = [Pr1 Pr];
if d_to_CH(pos_x_source) == 1e5
    C_source_to_CH_to_sink = C_to_sink;
    Pr_to_sink = Pr;
end
[size,~] = one_bit_transmit (C_source_to_CH_to_sink);
Pcon_LEACH_random =
Pcon_function(Pt_full,Pr_to_sink,C_source_to_CH_to_sink,1,0,1,
1);
T_LEACH_random = Data./(C_source_to_CH_to_sink*1e6); %
transmission time in seconds
T_bit = size./(C_source_to_CH_to_sink*1e6);
Energy_LEACH_random = T_LEACH_random.*Pcon_LEACH_random;
Energy_LEACH_1bit = T_bit.*Pcon_LEACH_random;
Total_energy_LEACH_random(a) = Total_energy_LEACH_random(a) +
sum(Energy_LEACH_random);
Efficiency_LEACH(a) = Efficiency_LEACH(a) +
size/sum(Energy_LEACH_1bit);
if a == 1
    Total_energy_LEACH_random_b(b) = sum(Energy_LEACH_random);
end

%% pegasis
[first_chain,second_chain] = pegasis2 (Area_x, Number_MS,
Pos_MS);
PL_first = PL_pegasis (first_chain,Pos_MS,Area_x,f);
if ismember(pos_x_source,first_chain)==1
    Pr_1 = Pt_full - PL_first;
    SNR_first_chain = Pt_full - PL_first - N_dBm; % in dB
    SNR_1 = 10.^(SNR_first_chain/10);
    C_1 = capacity(BW,SNR_1);
    C_source = 0;
    for k = 1:length(C_1)
        C_source = C_source + C_1(k);
    end
    p_r_used=Pr_1(find(first_chain==pos_x_source):end);
    C_used=C_1(find(first_chain==pos_x_source):end);
    [size,~] = one_bit_transmit (C_used);
```

```
                Pcon_PEGASIS_first =
                Pcon_function(Pt_full,p_r_used,C_used,1,0,1,1);
                T_PEGASIS_first = Data./(C_used*1e6); % transmission time
                in seconds
                T_bit = size./(C_used*1e6);
                Energy_PEGASIS1 = T_PEGASIS_first.*Pcon_PEGASIS_first;
                Energy_PEGASIS_1bit = T_bit.*Pcon_PEGASIS_first;
                Total_energy_pegasis(a) = Total_energy_pegasis(a) +
                sum(Energy_PEGASIS1);
                Efficiency_PEGASIS(a) = Efficiency_PEGASIS(a) +
                size/sum(Energy_PEGASIS_1bit);
                if a == 1
                    Total_energy_pegasis_b(b) = sum(Energy_PEGASIS1);
                end
            else
                PL_second = PL_pegasis (second_chain,Pos_MS,Area_x,f);
                Pr_2 = Pt_full - PL_second;
                SNR_second_chain = Pt_full - PL_second - N_dBm; % in dB
                SNR_2 = 10.^(SNR_second_chain/10);
                C_2 = capacity(BW,SNR_2);
                C_source = 0;
                for k = 1:length(C_2)
                    C_source = C_source + C_2(k);
                end
                p_r_used=Pr_2(find(second_chain==pos_x_source):end);
                C_used=C_2(find(second_chain==pos_x_source):end);
                [size,~] = one_bit_transmit (C_used);
                Pcon_PEGASIS_first =
                Pcon_function(Pt_full,p_r_used,C_used,1,0,1,1);
                T_PEGASIS_first = Data./(C_used*1e6); % transmission time
                in seconds
                T_bit = size./(C_used*1e6);
                Energy_PEGASIS2 = T_PEGASIS_first.*Pcon_PEGASIS_first;
                Energy_PEGASIS_1bit = T_bit.*Pcon_PEGASIS_first;
                Total_energy_pegasis(a) = Total_energy_pegasis(a) +
                sum(Energy_PEGASIS2);
                Efficiency_PEGASIS(a) = Efficiency_PEGASIS(a) +
                size/sum(Energy_PEGASIS_1bit);
                if a == 1
                    Total_energy_pegasis_b(b) = sum(Energy_PEGASIS2);
                end
            end
        end
    end
    Total_energy_LEACH_random_a = Total_energy_LEACH_random/(sim_time+1);
    Total_energy_Direct_a = Total_energy_Direct/(sim_time+1);
    Total_energy_pegasis_a = Total_energy_pegasis/(sim_time+1);
    Total_energy_dream_a = Total_energy_dream/(sim_time+1);

    Efficiency_DIRECT_a = Efficiency_DIRECT/(sim_time+1);
    Efficiency_PEGASIS_a = Efficiency_PEGASIS/(sim_time+1);
    Efficiency_LEACH_a = Efficiency_LEACH/(sim_time+1);
    Efficiency_DREAM_a = Efficiency_DREAM/(sim_time+1);
    end
```

2.1 Llamada y muestreo de resultados del código correspondiente al Anexo B

Nombre del fichero: *all_main.m*

```
clc
close all
clear all
tic;
drops_numb=500;
Area = 700:100:1000; % area size
sim_time = 50;
Total_energy_LEACH_random_a=zeros(drops_numb,length(Area));
Total_energy_Direct_a=zeros(drops_numb,length(Area));
Total_energy_pegasis_a=zeros(drops_numb,length(Area));
Total_energy_dream_a=zeros(drops_numb,length(Area));

Total_energy_LEACH_random_b = zeros(drops_numb,sim_time+1);
Total_energy_Direct_b = zeros(drops_numb,sim_time+1);
Total_energy_pegasis_b = zeros(drops_numb,sim_time+1);
Total_energy_dream_b = zeros(drops_numb,sim_time+1);

Efficiency_DIRECT_a = zeros(drops_numb,length(Area));
Efficiency_PEGASIS_a = zeros(drops_numb,length(Area));
Efficiency_LEACH_a = zeros(drops_numb,length(Area));
Efficiency_DREAM_a = zeros(drops_numb,length(Area));

for i=1:drops_numb
      disp(i);
      [Total_energy_LEACH_random_b(i,:), Total_energy_Direct_b(i,:),
      Total_energy_pegasis_b(i,:), Total_energy_dream_b(i,:),
      Efficiency_DIRECT_a(i,:), Efficiency_PEGASIS_a(i,:),
      Efficiency_LEACH_a(i,:),
      Efficiency_DREAM_a(i,:),Total_energy_LEACH_random_a(i,:),Total_e
      nergy_Direct_a(i,:),Total_energy_pegasis_a(i,:),Total_energy_dre
      am_a(i,:),clustersss]=main_over_drops_final(Area,sim_time);
end

 y1=mean(Total_energy_LEACH_random_a);
 y2=mean(Total_energy_Direct_a);
 y3=mean(Total_energy_pegasis_a);
 y4=mean(Total_energy_dream_a);

 x1=mean(Efficiency_LEACH_a)/1e6;
 x2=mean(Efficiency_DIRECT_a)/1e6;
 x3=mean(Efficiency_PEGASIS_a)/1e6;
 x4=mean(Efficiency_DREAM_a)/1e6;

 for i = 1:drops_numb
     Total_energy_LEACH_random_b(i,:) =
      cumsum(Total_energy_LEACH_random_b(i,:));
     Total_energy_Direct_b(i,:) = cumsum(Total_energy_Direct_b(i,:));
     Total_energy_pegasis_b(i,:) =
      cumsum(Total_energy_pegasis_b(i,:));
     Total_energy_dream_b(i,:) = cumsum(Total_energy_dream_b(i,:));
 end

 z1=mean(Total_energy_LEACH_random_b);
 z2=mean(Total_energy_Direct_b);
 z3=mean(Total_energy_pegasis_b);
 z4=mean(Total_energy_dream_b);
```

```
figure
colour_darkblue = [1 17 181] ./ 255;
p1=plot (Area, y2,'Color', colour_darkblue);
set(p1,'Marker','x');
set(p1,'LineStyle','-');
hold on

darkred=[178 34 4] ./ 255;
p2=plot (Area, y1,'Color', darkred);
set(p2,'Marker','o');
set(p2,'LineStyle','-');

colour_green = [1 181 17] ./ 255;
p3=plot (Area, y3,'Color', colour_green);
set(p3,'Marker','x');
set(p3,'LineStyle','-');

fuchsia=[250 0 250] ./ 255;
p4=plot (Area, y4,'Color', fuchsia);
set(p4,'Marker','o');
set(p4,'LineStyle','-');

limyd=min(min(min(y1(1),y2(1)),y3(1)),y4(1));
limyu=max([max(y1) max(y2) max(y3) max(y4)]);
xlim([Area(1) Area(end)]);
ylim([limyd limyu]);

grid on
set(p1,'LineWidth',1.5); set(p2,'LineWidth',1.5);
set(p3,'LineWidth',1.5); set(p4,'LineWidth',1.5);
toc

h_legend=legend ('Direct Communication','LEACH','PEGASIS','DREAM');
set(h_legend,'FontSize',12,'Location','northwest');
title('Energy Consumption, with mobility','FontSize',25);
ylabel('Energy Consumption [J] ','FontSize',17);
xlabel('Dimension of Area [m]', 'FontSize', 17);grid on;

set(gca,'FontWeight','normal','FontSize',12);

figure(2)
colour_darkblue = [1 17 181] ./ 255;
p1=plot (Area, x2,'Color', colour_darkblue);
set(p1,'Marker','x');
set(p1,'LineStyle','-');
hold on

darkred=[178 34 4] ./ 255;
p2=plot (Area, x1,'Color', darkred);
set(p2,'Marker','o');
set(p2,'LineStyle','-');


colour_green = [1 181 17] ./ 255;
p3=plot (Area, x3,'Color', colour_green);
set(p3,'Marker','x');
set(p3,'LineStyle','-');

fuchsia=[250 0 250] ./ 255;
```

```
p4=plot (Area, x4,'Color', fuchsia);
set(p4,'Marker','o');
set(p4,'LineStyle','-');
set(p1,'LineWidth',1.5); set(p2,'LineWidth',1.5);
set(p3,'LineWidth',1.5); set(p4,'LineWidth',1.5);

h_legend=legend ('Direct Communication','LEACH','PEGASIS','DREAM');
set(h_legend,'FontSize',12,'Location','northwest');
title('Energy efficiency, one source','FontSize',25);
ylabel('Efficiency [Mb/J] ','FontSize',17);
xlabel('Dimension of Area [m]', 'FontSize', 17);grid on;

set(gca,'FontWeight','normal','FontSize',12);

t = 0:1:sim_time;

figure(3)
colour_darkblue = [1 17 181] ./ 255;
p1=plot (t, z2,'Color', colour_darkblue);
set(p1,'LineStyle','-');
plot(t(1:10:end),z2(1:10:end), 'x')
hold on

darkred=[178 34 4] ./ 255;
p2=plot (t, z1,'Color', darkred);
set(p2,'LineStyle','-');
plot(t(1:10:end),z1(1:10:end), 'o')

colour_green = [1 181 17] ./ 255;
p3=plot (t, z3,'Color', colour_green);
set(p3,'LineStyle','-');
plot(t(1:10:end),z3(1:10:end), 'x')

fuchsia=[250 0 250] ./ 255;
p4=plot (t, z4,'Color', fuchsia);
set(p4,'LineStyle','-');
plot(t(1:10:end),z4(1:10:end), 'o')

grid on

set(p1,'LineWidth',1.5); set(p2,'LineWidth',1.5);
set(p3,'LineWidth',1.5); set(p4,'LineWidth',1.5);
toc

h_legend=legend ('Direct Communication','LEACH','PEGASIS','DREAM');
set(h_legend,'FontSize',12,'Location','northwest');
title('Energy consumption in time','FontSize',25);
ylabel('Energy Consumption [J] ','FontSize',17);
xlabel('Time [s]', 'FontSize', 17);grid on;

set(gca,'FontWeight','normal','FontSize',12);

SaveFigs = 1;
if SaveFigs == 1
    hgsave(sprintf('Leach'));
    saveas(gca,sprintf('Leach.png'));
    print -dtiff -r300 Leach;            %HighResolution Fig
end
ss=2;
```

3. ANEXO C: Códigos de las funciones utilizadas para el capítulo 2

3.1 Función generadora del posicionamiento de las estaciones móviles

```
function [PosMSinit] = MS_position(Area_x,Area_y,Number_MS,constrain)

% Generate position of mobile stations
% input:
%   Area_x ... size of simulation area in x coordinate
%   Area_y ... size of simulation area in y coordinate
%   Number_MS ... number of mobile stations
%   constrain ... min and max distance constrains between 2 MS
% output:
%   PosMSinit ... MS1 x | MS1 y | MS2 x | MS2 y | ....


PosMSinit=nan(2,Number_MS);

% generate first MS position
PosMSinit(1,1) = rand*Area_x;
PosMSinit(2,1) = rand*Area_y;

% generate others MS position
i = 2;
while i <= Number_MS
    PosMSinit(1,i) = rand*(PosMSinit(1,i-1)+constrain(2));
    PosMSinit(2,i) = rand*(PosMSinit(2,i-1)+constrain(2));
    d = distance (i,PosMSinit);
    if d < constrain (2) && d > constrain (1) && PosMSinit(1,i) <
     Area_x && PosMSinit(2,i) < Area_y
        i = i+1;
    end
end

end
```

3.2 Función calculadora de la atenuación en el espacio libre

```
%function to calculate path loss for outdoor to outdoor LOS D2D
communication
%constraints :   1-minimum distance between transmitter and receiver
is > 3 m
%                2-maximum distance between transmitter and receiver
is 5000 m
function [Path_loss]=calculate_O2O_LOS_D2D_path_loss(distance)
Path_loss = zeros(size(distance));
for i = 1:length(distance)
    for j = 1:length(distance)
        d=distance(i,j);
        freq=2;
        hUE = 1.5;
        hBS = 10;
        c = 3*10^8;

        PLfreeSpace = 20*log10(d) + 46.4 + 20*log10(freq/5);
```

```
        %  dxBP - Break Point, Separates Propagation distance
        hxBS=hBS-1; % hxBS= hBS - 1m;   effective eNB antenna height

        hxUE=hUE-1; % hxUE= hUE - 1m;   effective UE antenna height

        dxBP = 4*hxBS*hxUE*(freq*1000000000/c); % freq needed in [Hz]
        if (d == 0)
            PLd=1;
        elseif ((d > 3) && (d <= dxBP)) % 10m < d < dxBP
            PLd = 28 + 22*log10(d) + 20*log10(freq);        % PATH
             LOSS
        elseif ((d > dxBP) && (d < 5000))       % dBP < d < 5000m

            PLd = 7.8 + 40*log10(d) - 18*log10(hxBS) - 18*log10(hxUE)
             + 2*log10(freq);        % PATH LOSS

        end

        PL=PLd;

        Path_loss(i,j) = max(PLfreeSpace,PL);
    end
end
end
```

3.3 Implementación de la función correspondiente al protocolo PEGASIS

```
function [first_chain,second_chain] = pegasis3 (area, s_number,
Pos_MS,pos_x_source)

% Calculate the route following the PEGASIS protocol
% Input:
%   area ... [m^2]
%   s_number ... Number of mobile stations
%   Pos_MS ... Position of mobile stations
%   pos_x_source ... Position of the mobile station working as source
% Output:
%   first_chain ... First calculated chain that follows the PEGASIS
protocol
%   second_chain ... Second calculated chain that follows the PEGASIS
protocol

x = Pos_MS(1,:);
y = Pos_MS(2,:);

dis_from_sink = zeros(1,s_number);
chosen_s = zeros(1,s_number-1);

for i=1:s_number
    dis_from_sink(i)=sqrt(((x(i)-area)^2)+((y(i)-area)^2));
end
index_start_chain=pos_x_source;
[~,index_leader_chain]=min(dis_from_sink);

used_group=[];
for ii=1:(s_number-1)
    if ii==1
        current_s=index_start_chain;
```

```
            chosen_s(ii)=current_s;
        end
        used_group=[used_group,current_s];
        for iii=1:s_number
            if  ismember(iii,used_group)==0
                dis_from_others(iii)=sqrt(((x(current_s)-
                 x(iii))^2)+((y(current_s)-y(iii))^2));
            else
                dis_from_others(iii)=2*(area^2);
            end
        end
        [~,chosen_s(ii+1)]=min(dis_from_others);
        current_s=chosen_s(ii+1);
    end

    chain_of_s=chosen_s;
    index_leader_in_chain=find(chain_of_s==index_leader_chain);
    first_chain=chain_of_s(1:index_leader_in_chain);
    second_chain=fliplr(chain_of_s(index_leader_in_chain:end));
    end
```

3.4 Implementación de la función correspondiente al protocolo DREAM

```
function [chain] = dream3 (area, s_number, Pos_MS,pos_x_source)

% Calculate the route following the DREAM protocol
% Input:
%   area ... [m^2]
%   s_number ... Number of mobile stations
%   Pos_MS ... Position of mobile stations
%   pos_x_source ... Position of the mobile station working as source
% Output:
%   chain ... Calculated chain that follows the PEGASIS protocol

Area_x =area;
Area_y=area;
Number_MS=s_number;
const(1) = 4;     % minimum constrain in meters
const(2) = 1400;  % maximum constrain in meters
hop_distance = 100;

%--------------------------------------------------
x=Pos_MS(1,:);
y=Pos_MS(2,:);
x_source=x(pos_x_source);
y_source=y(pos_x_source);
pos_source=[x_source y_source];
sink = [Area_x Area_y];
chain(1)=pos_x_source;
Pos_MS = [Pos_MS sink'];

distance_to_sink = distance_2_points(sink', pos_source');

[~,index_start_chain]=max(distance_to_sink);

j = 2;

while distance_to_sink > 0
```

```
    d = ones(1,length(x))*1e4;
    for i=1:Number_MS+1
        if Pos_MS(1,i) > Pos_MS(1,chain(j-1)) && Pos_MS(2,i)>
           Pos_MS(2,chain(j-1))
            dist = distance_2_points (Pos_MS(:,chain(j-
             1)),Pos_MS(:,i));
            if dist < hop_distance
                d(i) = point_to_line_distance(Pos_MS(:,i)',
                 Pos_MS(:,chain(j-1)), sink);
            end
        end
    end
    [a,b] = min(d);
    chain(j)= b;
    if a==1e4
        hop_distance = hop_distance+50;
        j = j-1;
    else
        hop_distance = 100;
        distance_to_sink = distance_2_points(sink',
        Pos_MS(:,chain(j)));
    end
    j = j+1;
end

x_chosen = zeros(1,length(chain));
y_chosen = zeros(1,length(chain));
for i = 1:length(chain)
    x_chosen(i) = Pos_MS(1,chain(i));
    y_chosen(i) = Pos_MS(2,chain(i));
end

[~,index_leader_chain]=min(distance_to_sink);

chain(end) = [];
chain = chain;
end
```

3.5 Función del modelo energético

```
function Pcon = Pcon_function(Stx,Srx,C,mtx,mrx,cluster,pegasis)

% Calculate the LTE smartphone power consumption
% Input:
%   Stx ... UL transmit power [dBm]
%   Srx ... DL receive power [dBm]
%   C ... Capacity in Mbps using Shannon
%   mtx ... binary variable indicating active transmission
%   mrx ... binary variable indicating active reception
% Output:
%   Pcon ... Power consumption in RRC_connected mode [W]

Ptx = 29.9e-3;    % [W]
Prx = 25.1e-3;    % [W]
Pon = 853e-3;     % [W]
PtxRF = 0;        % [W]
PrxRF = 0;        % [W]
Stx_Watts = [];   % [W]
Srx_Watts = [];   % [W]
```

```
Pcon = [];

for i=1:length(C)
    Rtx(i) = C(i); % [Mbps]
    Rrx(i) = C(i); % [Mbps]

    if length(Stx) == 1
        if Stx <= 0.2 % [dBm]
            Stx_Watts = 10^(Stx/10); % [mW]
            PtxRF = (0.78*Stx + 23.6)*1e-3; % [W]
        elseif Stx > 0.2 && Stx <= 11.4 % [dBm]
            Stx_Watts = 10^(Stx/10); % [mW]
            PtxRF = (17*Stx + 45.4)*1e-3; % [W]
        elseif Stx > 11.4 % [dBm]
            Stx_Watts = 10^(Stx/10); % [mW]
            PtxRF = (5.9*(Stx)^2 - 118*Stx + 1195)*1e-3; % [W]
        end
    end

    if length(Stx) ~= 1
        if Stx(i) <= 0.2 % [dBm]
            Stx_Watts(i) = 10^(Stx(i)/10); % [mW]
            PtxRF(i) = (0.78*Stx(i) + 23.6)*1e-3; % [W]
        elseif Stx(i) > 0.2 && Stx(i) <= 11.4 % [dBm]
            Stx_Watts(i) = 10^(Stx(i)/10); % [mW]
            PtxRF(i) = (17*Stx(i) + 45.4)*1e-3; % [W]
        elseif Stx(i) > 11.4 % [dBm]
            Stx_Watts(i) = 10^(Stx(i)/10); % [mW]
            PtxRF(i) = (5.9*(Stx(i))^2 - 118*Stx(i) + 1195)*1e-3; %[W]
        end
    end

    PtxBB = 0.62e-3; % [W]

    if Srx(i) <= -52.5 % [dBm]
        Srx_Watts(i) = 10^(Srx(i)/10); % [mW]
        PrxRF(i) = (-0.04*Srx(i) + 24.8)*1e-3; % [W]
    elseif Srx(i) > -52.5% [dBm]
        Srx_Watts(i) = 10^(Srx(i)/10); % [mW]
        PrxRF(i) = (-0.11*Srx(i) + 7.86)*1e-3; % [W]
    end
    PrxBB(i) = (0.97*Rrx(i) + 8.16)*1e-3; % [W]

    if length(Stx) == 1
        Pcon(i) = Pon + mrx*(Prx + PrxBB(i) + PrxRF(i)) + mtx*(Ptx +
        PtxBB + PtxRF);

        if nargin == 6
            if i == cluster
                mtx = 1;
                mrx = 1;
                Pcon(i) = Pon + mrx*(Prx + PrxBB(i) + PrxRF(i)) +
                mtx*(Ptx + PtxBB + PtxRF(i));
            end
        end
    end

    if length(Stx) ~= 1
        Pcon(i) = Pon + mrx*(Prx + PrxBB(i) + PrxRF(i)) + mtx*(Ptx +
        PtxBB + PtxRF(i));
```

```
            if nargin == 6
                if i == cluster
                    mtx = 1;
                    mrx = 1;
                    Pcon(i) = Pon + mrx*(Prx + PrxBB(i) + PrxRF(i)) +
                     mtx*(Ptx + PtxBB + PtxRF(i));
                end
            end
        end


        if length(Stx) == 1
            Pcon(i) = Pon + mrx*(Prx + PrxBB(i) + PrxRF(i)) + mtx*(Ptx +
            PtxBB + PtxRF);
            if nargin == 7
                if i > 1
                    mtx = 1;
                    mrx = 1;
                    Pcon(i) = Pon + mrx*(Prx + PrxBB(i) + PrxRF(i)) +
                     mtx*(Ptx + PtxBB + PtxRF);
                end
            end
        end
        if length(Stx) ~= 1
            Pcon(i) = Pon + mrx*(Prx + PrxBB(i) + PrxRF(i)) + mtx*(Ptx +
            PtxBB + PtxRF(i));

            if nargin == 7
                if i > 1
                    mtx = 1;
                    mrx = 1;
                    Pcon(i) = Pon + mrx*(Prx + PrxBB(i) + PrxRF(i)) +
                     mtx*(Ptx + PtxBB + PtxRF(i));
                end
            end
        end
    end
```

3.6 Función calculadora de las distancias entre nodos

```
function dist = distance(Number_MS,PosMS)

% calculate minimum distance between last MS and rest of them
% input:
%   Number_MS ... Number of MS in the simulation
%   PosMS ... Position of MS (MS1 x | MS1 y | MS2 x | MS2 y | ....)

d=zeros(1,Number_MS-1);


for i=1:Number_MS-1
    d(i)=sqrt((PosMS(1,Number_MS)-PosMS(1,i))^2 + (PosMS(2,Number_MS)-
        PosMS(2,i))^2);
end

dist = min(d);
end
```

3.7 Función calculadora de la capacidad

```
function [C] = capacity(BW,SINR)

%   computation of capacity using Shannon
%   C = BW*log2(1+SINR)
%   capacity is in Mbps

C = BW*log2(1+SINR)/1000000;
% for i = 1:length(SINR)
%     C (i,i) = NaN;
% end

end
```

3.8 Función calculadora de la potencia de la señal recibida

```
function RSS = RSS_function(Number_MS,Pt,PL)

% Input:
%   Number_MS ... Number of MS in the simulation
%   Pt ... Transmission power of MSs
%   PL ... pathlos between individual stations in dB

RSS=zeros(Number_MS);
for i=1:Number_MS
    for j=1:Number_MS
       RSS(i,j) = Pt-PL(i,j);
       if i == j
           RSS(i,j) = 0;
       end
    end
end
```

3.9 Función controladora de la potencia transmitida

```
function Pt = power_control (N_dBm, PL, SINRth)

% compute Pt for given SINR threshold
% Input:
% N_dBm ... noise power in dBm
% PL ... path loss
% SINRth ... SINR threshold
% Output:
% Pt ... transmitting power

for i = 1:length(PL)
    Pt = SINRth + PL + N_dBm;
end
end
```

3.10 Función seleccionadora de los clústeres

```
function cluster =
choose_point_with_probability(Pos_MS,Number_MS,Number_CH)
```

```
% election of cluster heads
% Input:
%   Number_MS ... Number of MS
%   PosMS ... Position of MS
%   Number_CH ... Number of cluster heads
% Output:
%   cluster ... number of MS which is going to work as a cluster in
order

cluster(1) = round(rand*Number_MS);
if cluster(1)==0
cluster(1)=1;
end
for i=2:Number_CH
    P = Probability(Pos_MS,cluster);
    X = 1:length(Pos_MS(1,:));
    Cumulative_Probabilities=cumsum(P);
    rand_number=rand;
    cluster(i) =
     X(find(rand_number<Cumulative_Probabilities,1,'first'));
end
```

3.11 Función que asocia los nodos que transmiten a los clústeres

```
function [dist MS_Assoc] = MS_association (Pos_MS, cluster)

% define the cluster head for each MS
% Input:
%   PosMS ... Position of MS
%   cluster ... number of MS which is going to work as a cluster in
order
% Output:
%   dist ... distance for each MS to cluster head
%   MS_Assoc ... cluster head for MS with shortest distance

Pos_C = [Pos_MS(1,cluster) Pos_MS(2,cluster)];

MS_dist = zeros(length(Pos_MS),length(cluster));
MS_Assoc = zeros(1,length(Pos_MS));

for i = 1:length(Pos_MS)
    for j = 1:length(cluster)
        MS_dist(i,j) = sqrt((Pos_MS(1,i)-Pos_C(1,j))^2 + (Pos_MS(2,i)-
          Pos_C(1,j+length(cluster)))^2);
        k=1;
        while k <= length(cluster) % to eliminate CH from shortest
          distance
            if i == cluster(k)
                MS_dist(i,j) = 1e5;
            end
            k = k + 1;
        end
    end
end
MS_dist = MS_dist';
[dist MS_Assoc] = min(MS_dist);

end
```

3.12 Función generadora de movilidad en la red de sensores inalámbricos móviles

Es una función cuyo fin es generar movilidad en la red de sensores inalámbricos móviles creada por Mathieu Boutin que recibe el nombre de "Random Waypoint mobility model" [12].

```
function coordinates_movers = mobility(Pos_MS, Area, sim_time)

% Generate mobility of mobile stations
% input:
%   Pos_MS ... position of mobile stations
%   Area ... size of simulation area
%   sim_time ... time of simulation
% output:
%   coordinates_movers ... coordinates of the mobile stations in order
%   to generate mobility

x_start=Pos_MS(1,:);
y_start=Pos_MS(2,:);
nodes_numb = length(Pos_MS);

Area_start=0;
Area_end=Area;

s_input = struct('V_POSITION_X_INTERVAL',[Area_start Area_end],...%(m)
    'V_POSITION_Y_INTERVAL',[Area_start Area_end],...%(m)
    'V_SPEED_INTERVAL',[0.2 10],...%(m/s)
    'V_PAUSE_INTERVAL',[1 1],...%pause time (s)
    'V_WALK_INTERVAL',[1.00 1.00],...%walk time (s)
    'V_DIRECTION_INTERVAL',[-180 180],...%(degrees)
    'SIMULATION_TIME',sim_time,...%(s)
    'NB_NODES',nodes_numb);
s_mobility = Generate_Mobility(s_input,x_start,y_start);

close
coor_help=s_mobility.VS_NODE;

length_values=zeros(1,nodes_numb);
for index=1:nodes_numb
    length_values(index)=length(coor_help(index).V_TIME);
end
chosen_length=min(length_values);

for index2=1:nodes_numb
    if length(coor_help(index2).V_TIME)>chosen_length
        for index_deleting=1: length(coor_help(index2).V_TIME)-
chosen_length
            coor_help(index2).V_TIME(end-1)=[];
            coor_help(index2).V_POSITION_X(end-1)=[];
            coor_help(index2).V_POSITION_Y(end-1)=[];
        end
    end
end

%-----------------results
coordinates_movers=zeros(2,chosen_length,nodes_numb);
for iiii=1:nodes_numb
    coordinates_movers(1,:,iiii)=coor_help(iiii).V_POSITION_X'  ;
    coordinates_movers(2,:,iiii)=coor_help(iiii).V_POSITION_Y' ;
end
end
```