Diseño, Simulación y Estudio de una red de sensores

Autor: Vicente Aliaga
Director: Alberto Bonastre
Fecha: Diciembre del 2011
Titulación: Ingeniería Técnica en Informática de Sistemas

Resumen

El objetivo de este proyecto es el diseño y estudio de una red de sensores de bajo consumo.

La red será auto-configurable. La configuración de red y los saltos entre nodos no serán controlados externamente, sino que serán los nodos mismos los que se encargarán de la configuración de red.

La red utiliza nodos inalámbricos de sensores cuyos recursos energéticos se limitan a sus propias baterías. Estos nodos se encargan de recopilar información del medio y pasan la mayor parte del tiempo en un estado de bajo consumo de energía, despertándose puntualmente para medir el entorno, procesarlos y transmitir o recibir datos. Para minimizar la energía que utiliza el sistema, debemos maximizar el tiempo que pasan los nodos en el estado de bajo consumo.

En muchas ocasiones, los nodos pasan a modo activo (se "despiertan") periódicamente y de forma coordinada. Sin embargo, en este proyecto se explorará otra vía, consistente en el denominado Wake-up activo. Este mecanismo consiste en dotar a los nodos de un circuito adicional de consumo despreciable capaz de despertar al nodo como consecuencia de la recepción de una señal de radio específica. Asimismo, se dotará a los nodos sensores con un emisor de esta señal, de forma que cualquier nodo pueda despertar a otros.

En este trabajo emplearemos diferentes frecuencias de radio para despertar selectivamente diferentes nodos. Así, cada nodo responderá a una frecuencia predeterminada distinta de los nodos de su entorno. Por otro lado, el circuito emisor está diseñado para que cada nodo pueda seleccionar la frecuencia emitida, y por tanto a qué otro nodo despertar. El diseño del proyecto se centrará en dos aspectos:

- **La configuración de la topología de red:** los nodos de nuestra red deben ser capaces de organizarse en una buena configuración de red que se adapte a las exigencias de las redes inalámbricas de bajo consumo por ellos mismos.

  Diseñaremos un algoritmo de adopción de nodos que dotará al sistema de una configuración de red en forma de árbol. El algoritmo será flexible en cuanto a parámetros como la distancia entre nodos, el número de nodos del sistema, y la introducción o pérdida de nodos en el sistema.

  Estudiaremos distintas configuraciones de red en forma de árbol para refinar nuestro algoritmo de configuración de red.

- **Políticas de wake-up:** Para permitir que los nodos pasen la mayor parte del tiempo en el estado de bajo consumo, definiremos distintas políticas de wake-up que permitirán no sólo despertar a nodos específicos en el sistema, sino además establecer caminos dentro de la red para ofrecer comunicación indirecta entre nodos.

  Nuestras políticas de wake-up están diseñadas sobre frecuencias de radio específicas, las cuales son únicas para cada todo del sistema – esto nos permitirá controlar qué nodos son despertados, y podremos establecer distintos caminos por el árbol sin generar conflictos o despertar más nodos de lo necesario.

Palabras clave

red de sensores, red de bajo consumo, nodos de bajo consumo, nodos inalámbricos, técnicas de wake-up, técnicas de despertar, redes wake-up, protocolos de redes de sensores, configuración de redes de sensores, simulador ns

Contenidos

1. **Introducción**
   - 1.1. Introducción a las redes de sensores de bajo consumo
   - 1.2. Estructura de la memoria
2. **Objetivos y requisitos del sistema y de la red**
   - 2.1. Nodos
   - 2.2. Configuración de red
   - 2.3. Funcionalidad
   - 2.4. Estudio
3. **Diseño de la red de sensores**
   - 3.1. Pautas de diseño. Repaso de requisitos.
   - 3.2. Diseño de los nodos a utilizar
   - 3.3. Diseño de la configuración de red y del algoritmo de adopción
     - 3.3.1. Uso del algoritmo de adopción
   - 3.4. Diseño de las funcionalidades para la red de sensores
4. **Componentes de la simulación de red**
   - 4.1. Introducción a la simulación
   - 4.2. Interfaz de usuario para la especificación de red
   - 4.3. Simuladores de red
     - 4.3.1. ns-2
     - 4.3.2. ns-3
   - 4.4. Visualizador de red
5. **Estudio de redes de sensores utilizando distintas configuraciones de red**
   - 5.1. Configuración en árbol
   - 5.2. Configuración en árbol degenerado
   - 5.3. Configuración en Estrella
   - 5.4 Comparativa
6. **Resultados**
7. **Conclusiones y futuras ampliaciones**
   - 7.1. Conclusiones
   - 7.2. Futuras ampliaciones
8. **Bibliografía y aplicaciones utilizadas**
   - 8.1. Bibliografía y fuentes usadas

1. Introducción

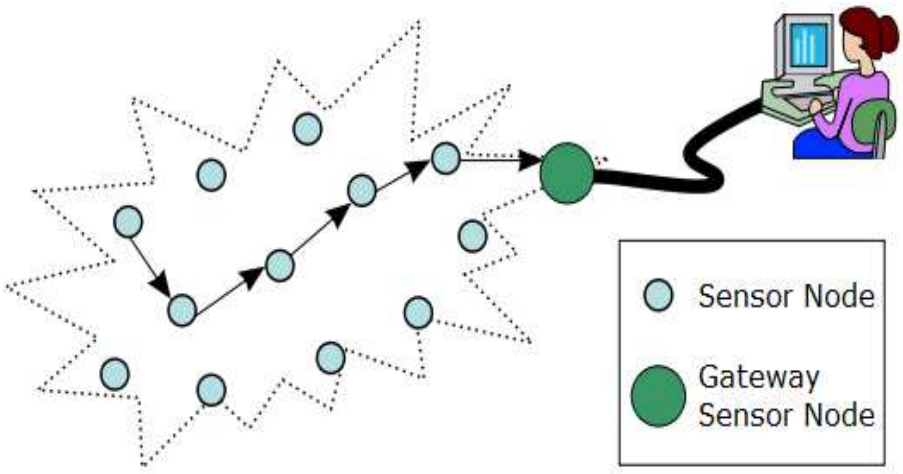
1.1. Introducción a las redes de sensores de bajo consumo

El objetivo de esta introducción es establecer el contexto, las características, y las definiciones de las redes de sensores de bajo consumo. Aunque no relacionado directamente con los objetivos y temas a tratar en el proyecto en sí, es interesante exponer el contexto del trabajo que estamos realizando. La introducción ha sido recopilada y estructurada a partir de información bibliográfica de libre acceso que será listada en el apartado correspondiente (Sección 8).

Los avances en micro-tecnología y en la tecnología inalámbrica han permitido el desarrollo y expansión de las redes inalámbricas de sensores (RIS).

Una RIS consiste en un conjunto de nodos autónomos inalámbricos que pasan la mayor parte del tiempo en un estado de bajo consumo. Este conjunto de nodos se encarga de recoger información de su entorno y enviarla de forma cooperativa a un nodo central o sumidero, el cual se encarga de recolectar los datos de la red.

Típicamente este tipo de redes se utiliza para recopilar información de distintos medios, por ejemplo para monitorizar cambios ambientales o físicos, como la temperatura, sonido, humedad o presión.



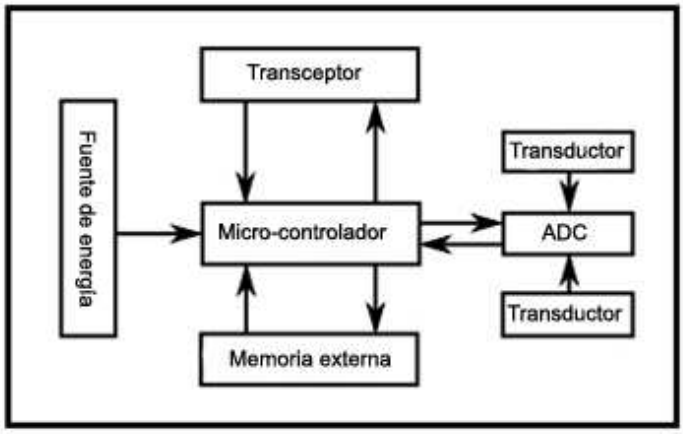
Red de sensores

Una de las primeras aplicaciones de las RIS fueron las aplicaciones militares, en particular para la monitorización del campo de batalla. Hoy en día este tipo de redes se utiliza en gran cantidad de aplicaciones, como en operaciones industriales, de tráfico, salud, etc.

El mayor desafío de las redes inalámbricas de sensores es producir nodos pequeños y de bajo coste, denominamos “sensores”.

Cada nodo de la red tiene varias partes: uno o varios transductores, usados para recopilar información. Estos nodos tienen una antena de radio, y un micro-controlador que se comunica con el sensor. Poseen además una fuente de energía, normalmente algún tipo de batería.

Los nodos pueden verse como pequeños computadores. Típicamente poseen una unidad de procesamiento y una memoria muy limitada.



Arquitectura típica de un nodo

Veamos de cerca los componentes típicos de un nodo:

Controlador *(controller)* – Se encarga de realizar diversas tareas, de procesar datos, y de controlar y manejar a los otros componentes del nodo. Suele implementarse mediante un micro-controlador, debido a su bajo coste (tanto de fabricación como de consumo energético), a su pequeño tamaño, versatilidad, y a la facilidad de integración con otros componentes.

Transceptor *(transceiver)* – Un transceptor combina las funciones de transmisor y receptor. Un transceptor tiene cuatro estados distintos: envío, recepción, inactivo, y suspendido. La mayoría de los transceptores consumen casi tanta energía en modo inactivo que en modo de recepción, por lo que es mejor suspender el nodo cuando no está ni transmitiendo ni recibiendo. De ahí la necesidad de tecnologías de “wake-up” independientes de la etapa de radio del nodo..

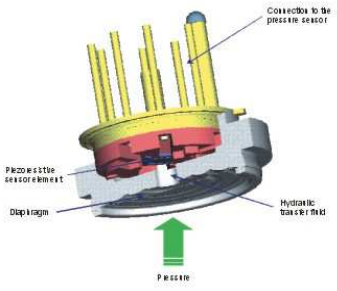
Dependiendo del tipo de red de sensores se podría transmitir por radio, infra-rojos o por laser. La radio suele ser el medio más popular en el campo de redes de sensores. La mayoría de los transceptores usados por los nodos de sensores utilizan la banda ISM.

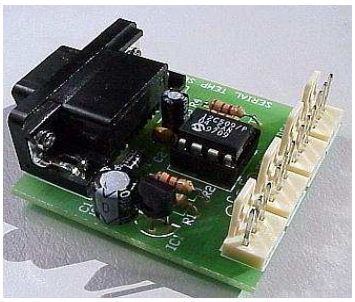
Memoria externa *(external memory)* – Los nodos suelen estar dotados de poca memoria RAM como memoria de usuario, para las aplicaciones o datos personales, y de algún tipo de memoria no volátil , como la memoria flash, para el almacenamiento de programas y datos permanentes, debido a su bajo coste y una aceptable capacidad de almacenamiento. **Fuente Energética** *(power source)* – El nodo consume energía durante el proceso de comunicación, el procesamiento de datos, y al utilizar los sensores. El proceso de comunicación tiene el coste energético más alto de los tres. Normalmente se utilizan baterías o condensadores como fuentes de energía, que en algunos casos pueden recargarse a través de fuentes externas, como la energía solar o por cambios de temperatura, si el nodo dispone de los dispositivos adecuados.

Normalmente se utilizan dos políticas para conservar energía: la primera es el *"Dynamic Power Managament",* a través de la cual se conserva energía dejando partes del nodo no utilizadas en estado de bajo consumo. La segunda política, *"Dynamic Voltage Scaling"* cambia el voltaje y la frecuencia dentro del nodo dependiendo de la actividad del nodo.

Transductor *(transducer)* – Los transductores son aparatos hardware que responden ante un cambio físico del ambiente, como la temperatura, humedad, vibración o presión. La señal analógica producida por los transductores se digitaliza a través de un conversor analógico-digital y permite que la unidad de procesamiento del nodo procese la información.

Los transductores utilizados en redes de sensores son sensores de tipo pasivo. Esto quiere decir que reciben datos sin manipular o modificar el entorno. Cada transductor cubre un área determinada a su alrededor que monitoriza.






Distintos tipos de sensores

El nodo recolector o sumidero se asemeja más a los computadores normales, al tener toda la energía necesaria para realizar sus funciones y no tener que apagarse periódicamente. Esto además permite que se usen con hardware y software más potentes.

Las características de una red de sensores son diversas. Entre ellas, el tema de energía y longevidad de la red es de las más importantes. Esto está directamente relacionado a

la vida y energía de los nodos que componen la red, por lo que aprovechar el tiempo en que un nodo está despierto es algo muy importante.

Podemos ver que el software y capacidad de cálculo de un nodo son limitados. Al aumentar los cálculos realizados por el nodo (los cuales ocurren por la necesidad de establecer comunicación entre nodos), la vida del nodo se acorta. Es por ello necesario definir un sistema de wake-up por radio selectivo, para mantener a los nodos despiertos sólo cuando sea necesario. Por esta razón los algoritmos usados en los nodos se centran en maximizar la vida del mismo, en darle robustez y tolerancia a fallos al nodo, y en auto-configurar al nodo.

Otra de las características de las redes de bajo consumo es la habilidad de tolerar situaciones inestables o dinámicas: es decir, la red puede tolerar cambios en la topología de red, movimiento o re-localización de los nodos, fallos en la comunicación entre nodos, reestructuración de la red con la introducción de nuevos nodos, y reestructuración de la red con el fallo de nodos determinados.

Con estos conceptos y definiciones, introducimos este proyecto de fin de carrera.

Teniendo en cuenta los paradigmas de las redes de sensores de bajo consumo ya mencionados, como la escasez de energía y la tolerancia a fallos y cambios, el proyecto se centrará en varios temas:

Por una parte, el objetivo es diseñar una red de sensores cuyos nodos serán capaces de auto-configurarse y establecer caminos en la red de forma automática y dinámica.

Los nodos de la red sólo permanecerán activos cuando sea necesario que lo estén (como para establecer estos caminos), y se pasarán en estado de bajo consumo el resto del tiempo (wake-up). La red soportará distintas técnicas de wake-up de nodos.

Por otra parte, realizaremos un estudio sobre distintos tipos de árboles de redes de sensores, en el cuál observaremos distintas ventajas e inconvenientes de uso.

Utilizaremos distintas aplicaciones para la realización del proyecto, como el simulador de redes ns-2 junto con el visualizador del simulador (nam) para realizar la simulación y estudio de la red.

1.2. Estructura de la memoria

Objetivos

En este apartado hablaremos de las especificaciones y objetivos del proyecto:

El primero objetivo del proyecto consiste en el diseño y estudio de una red de sensores de nodos de bajo consumo.

Nuestra red será capaz de organizar todos los nodos del sistema en forma de árbol con relaciones de parentesco entre los nodos de sensores del sistema, y entonces será capaz de despertar de forma selectiva a distintos nodos del sistema de forma inalámbrica. Gracias a este sistema para despertar a los nodos, podremos enviar información a través de caminos dinámicamente configurables dentro del árbol, despertando sólo a los nodos necesarios para hacer posible la propagación de este mensaje.

Para poder llevar a cabo el diseño e implementación de esta red, necesitaremos un algoritmo de encaminamiento y formación de la configuración para el árbol que sea capaz de difundir la información necesaria por la red inalámbrica de sensores. El diseño correcto de estos dos elementos nos permitirá aplicar las distintas políticas a la hora de despertar a los nodos.

Otro objetivo del proyecto es el estudio sobre distintos casos de árbol para la red. En dicho estudio observaremos el funcionamiento y calidad los distintos casos para varias técnicas de despertar, y en general como responden ante el uso de redes de sensores inalámbricas de bajo consumo.

Diseño

En el siguiente apartado hablaremos de las decisiones de diseño adoptadas para la formación de la red. Veamos un resumen elemento a elemento:

En cuanto al diseño de los nodos nos centraremos en dos partes: en la parte de hardware: cada nodo tiene su propia frecuencia de wake-up, lo que nos permite despertar nodos de forma selectiva. La segunda parte trata sobre la información necesaria que un nodo debe conocer para poder participar de forma satisfactoria en el sistema, como la frecuencia específica y la dirección de encaminamiento para despertar y establecer comunicación con su padre e hijos.

En cuanto al diseño del algoritmo, necesitaremos un algoritmo capaz de establecer las relaciones de parentesco necesarias entre los nodos del sistema, y crear una configuración de red en la cual cualquier nodo pueda llegar a comunicarse con el sumidero (el cual ocupará la posición de raíz en el árbol) o viceversa, independientemente de la distancia o los caminos entre nodos que deban establecerse. Para ello un nodo intentará adoptar a cualquier otro nodo que esté a su alcance, siempre y cuando el nodo objetivo no tenga padre o la adopción deje al nodo

objetivo más cerca del sumidero del sistema. Tras la adopción, la información de parentesco se propagará a través del árbol.

En lo referido a los sistemas de wake-up a emplear en el árbol, nos apoyaremos en la información transmitida por el árbol gracias al sistema de adopción y al diseño del nodo del sistema Se aplicarán diversas políticas a la hora de despertar nodos, entre ellas el despertar de un camino de nodos hasta la raíz, el despertar de un camino de nodos hasta un nodo específico, el despertar de todos los hijos, y otros.

Estudio

El proyecto continua con un estudio sobre el uso de diversas configuraciones de red para los distintos tipos de despertares en el sistema. Veremos el efecto que cada configuración de red puede tener en la calidad de la red, en costes energéticos de los nodos, en tiempo que permanecen los nodos despiertos, tiempo útil de los nodos, etc.

2. Objetivos y requisitos del sistema

El objetivo de este proyecto es el diseño e implementación de una red de sensores de nodos de bajo consumo.

Buscamos algo claramente importante en el campo de las redes de bajo consumo: reducir el consumo de energía en la red, y minimizar el tiempo que pasan los nodos fuera del modo de bajo consumo. Las decisiones tomadas se orientan hacia estos dos objetivos.

Veamos pues las especificaciones y requisitos del sistema, en particular de los elementos que lo componen, del conjunto que ellos forman, y de la funcionalidad que el sistema debe tener:

2.1. Requisitos y especificaciones de los nodos

En lo referido a los **nodos** a utilizar:

Nuestra red está compuesta por dos tipos de elementos: **nodos inalámbricos de sensores de bajo consumo**, cuyo número es variable, y un sólo **nodo controlador sumidero,** conectado a la corriente.

Para ahorrar energía, los nodos permanecerán en modo de bajo consumo la mayor parte del tiempo hasta que en algún momento despertarán, recibirán y/o enviarán un mensaje, y volverán a dormir. El circuito de wake-up de nuestros nodos es programable y cada uno de ellos estará preparado para funcionar a una frecuencia distinta. Esto nos permitirá despertar a nodos específicos sin sacar al resto del estado de bajo consumo de forma innecesaria.

Para comunicarse con un nodo cualquiera del sistema, será necesario asegurar que el nodo está despierto con un mensaje enviado a la correcta frecuencia de wake-up para ese nodo específico. Tanto los nodos inalámbricos como el controlador son capaces de despertar a otros nodos inalámbricos si envían el mensaje de despertar a la frecuencia adecuada.

Los nodos tienen un alcance de señal determinada y bastante limitada para el tamaño de la red, por lo que un nodo no será capaz de alcanzar y despertar (o de comunicarse) con todos los nodos del sistema. Esto incluye al nodo sumidero, el cual tiene un alcance semejante al de los nodos de sensores. Para establecer comunicación entre dos nodos que no puedan comunicarse directamente, será necesario crear un camino de nodos, despertando a los nodos necesarios para llegar al nodo objetivo.

2.2. Requisitos de la Configuración de red y uso de algoritmo generador

En lo referido a la **configuración de red**:

Buscamos el establecimiento de una red en forma de árbol, con relaciones de parentesco entre nodos. La raíz del árbol será el nodo controlador o sumidero, el cual conoce la topología del árbol entero. El nodo sumidero no tiene un alcance mayor que el resto de los nodos y deberá respaldarse en caminos formados por los nodos hijos. Una vez el árbol ha sido establecido, cada nodo puede considerarse como sumidero de su propio subárbol, y conocerá la configuración de red del mismo.

El resultado de esta organización será que los nodos superiores del árbol, los más cercanos al sumidero principal, serán los que más información contienen y los que pasan mayor tiempo realizando cálculos, mientras que los nodos más cercanos a las hojas pueden centrarse en reenviar la información hacia el sumidero del árbol a través de caminos de nodos.

Además de la organización de los nodos iniciales, el sistema debe ser capaz de incorporar nuevos nodos en el árbol con facilidad, y reaccionar ante la pérdida de un nodo existente en el sistema.

En resumidas cuentas, tras colocar nodos de forma pseudo-aleatoria (algo que puede seguir ocurriendo a lo largo del tiempo), nuestro sistema debe ser capaz de organizarlos en configuración de red de árbol.

Para ello diseñaremos un **algoritmo** de configuración de red, capaz de organizar la red en forma de árbol de y proporcionar a los nodos conectividad con el sumidero (de forma directa o indirecta).

2.3. Funcionalidad requerida del sistema: tipos de Wake-up soportados

En lo referido a la **comunicación** entre los elementos de la red:

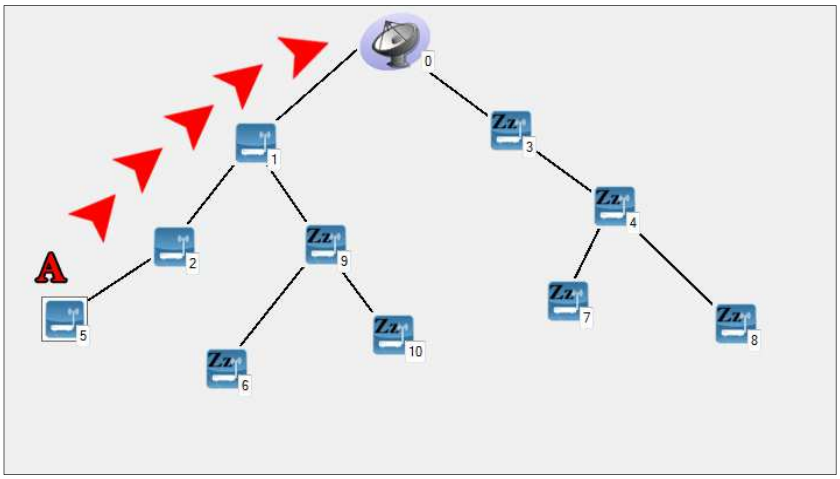
En nuestra red, un nodo cualquiera sólo podrá comunicarse directamente con su padre y con sus hijos, pero indirectamente con cualquier nodo. Aunque un nodo no tiene ni el alcance necesario para llegar a todos los nodos del árbol, ni la información necesaria para despertar o enviar mensajes a nodos distintos no considerados como hijo/padre por el algoritmo del punto 2.2, un nodo es capaz de llevar un mensaje a cualquier nodo del árbol a través de la creación de un camino multi-hop de nodos que actuarán como encaminadores del mensaje a través del árbol. Un camino que será establecido a través de un wake-up selectivo a través del uso de las frecuencias de despertar adecuadas.

Obtenemos así la posibilidad de transmitir datos entre dos nodos del árbol de forma indirecta, independientemente de sus relaciones de parentesco, de su posición, y la distancia entre ellos.

Esto nos lleva a la mención de los distintos tipos de funcionalidades de despertar que se esperan de la red, necesarias ya sea para una comunicación directa (ya que un nodo siempre debe haber despertado antes de participar en comunicación) o indirecta (un nodo también deberá despertar para actuar de camino):

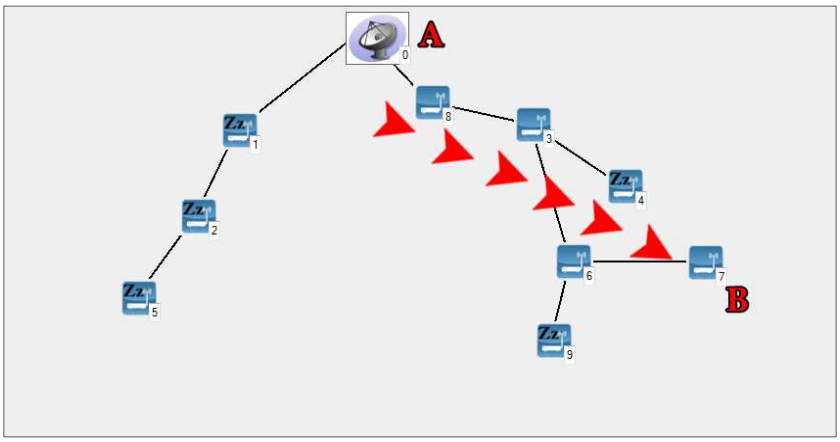
El tipo de políticas de wake-up que esperamos del sistema son los siguientes:

- Debido a la naturaleza del sistema, si un nodo se despierta, debe ser capaz de enviar la información recogida del entorno al sumidero del sistema para el análisis o almacenamiento de la misma. Un nodo A cualquiera debe ser capaz de generar un camino de nodos hasta la raíz del sistema. Por lo tanto, un nodo debe ser capaz de despertar a su propio padre y de hacer que despierte a su vez a otros nodos hasta llegar a la raíz del árbol.



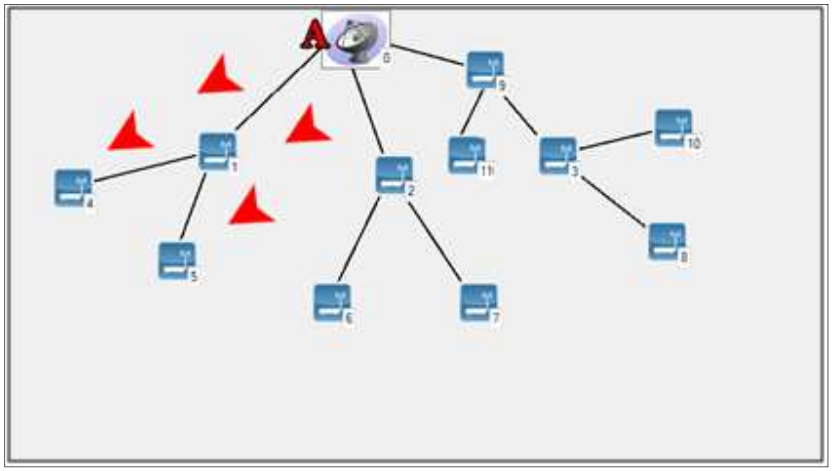
El primer tipo de despertar ilustrado. El nodo A propaga un despertar a través del árbol hasta llegar a la raíz, al sumidero del sistema.

- Un nodo A cualquiera (incluyendo a la raíz del sistema) debe ser capaz de despertar y crear un camino de nodos hasta un nodo B cualquiera del sistema. Un nodo, entonces, debe ser capaz de propagar un mensaje de despertar. Este tipo de funcionalidad es necesaria, por ejemplo, si la raíz del sistema desea consultar la información recopilada por un nodo cualquiera del sistema antes de que el mismo nodo la envía hacia el sumidero.



El segundo tipo de despertar ilustrado. El nodo raíz necesita información del nodo B, por lo que establece un camino hasta llegar a él.

- Un nodo A cualquiera debe ser capaz de despertar a todos sus hijos. Este despertar debe poder extenderse hasta las hojas, o detenerse a un nivel determinado. Este despertar es necesario si queremos tener la opción de despertar a grandes parte del sistema al mismo tiempo. Un nodo raíz de un subárbol (o del sistema completo) puede estar interesado en recopilar información de todos sus nodos hijos.



El tercer tipo de despertar ilustrado. El nodo raíz se prepara para recopilar información de una rama.

Sea cual sea el tipo de despertar, debemos mantener tantos nodos dormidos (y por lo tanto ahorrando energía) como sea posible. Esto será posible gracias a las frecuencias de wake-up distintas para cada nodo.

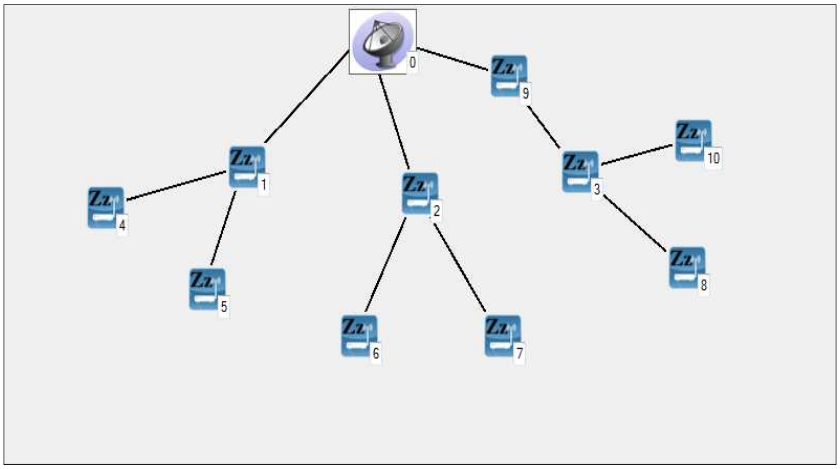
2.4. Estudio sobre los distintos métodos de wake-up sobre distintas configuraciones

En lo que se refiere a la parte del **estudio**, forzaremos la organización de la red en distintas configuraciones, sobre las cuales estudiaremos las distintas políticas de wake-up ya mencionadas en el apartado anterior.

Ciertas políticas de wake-up funcionarán mejor para ciertas configuraciones de red, en lo que se refiere a energía usada, tiempo que los nodos pasan despiertos, y otros datos que recogeremos a lo largo del estudio.

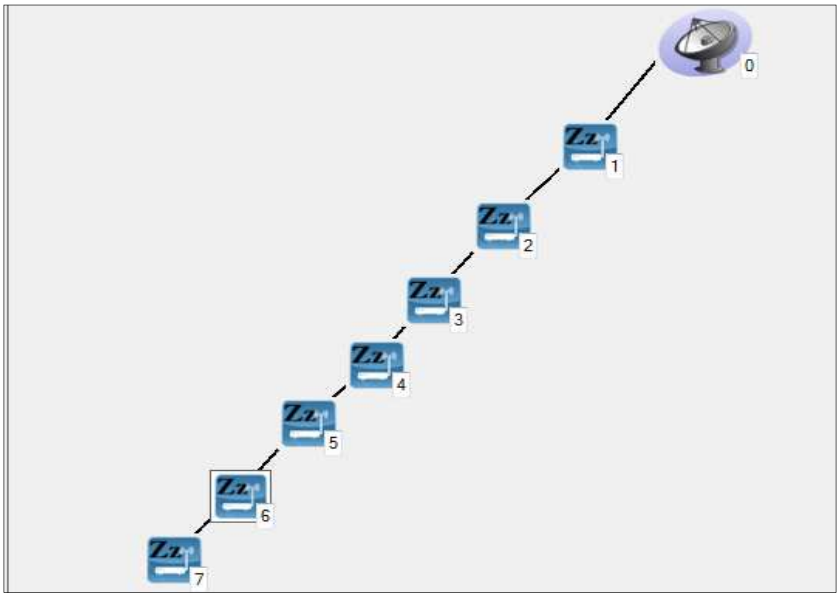
Las configuraciones de red serán las siguientes:

- ***Árbol equilibrado:*** Cada nodo tiene de promedio el número máximo de hijos establecidos. Esta configuración cubre un terreno relativamente amplio en varias direcciones en el cual los nodos pueden recoger la información necesaria a través de sus sensores.



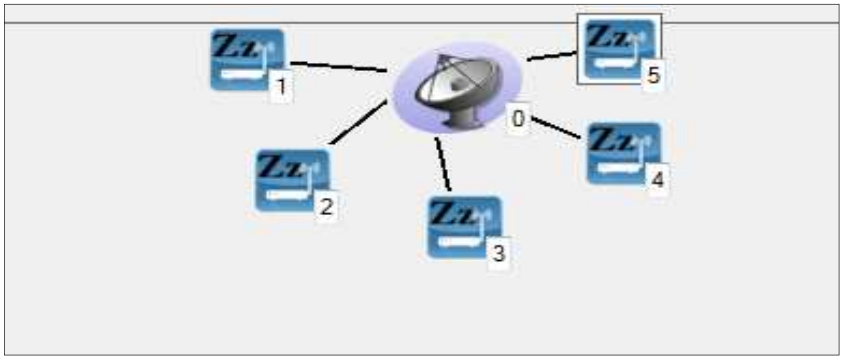
Configuración de árbol equilibrado ilustrada, con un número máximo de tres hijos por nodo.

- ***Árbol degenerado:*** Cada nodo tendrá como mucho un hijo. Esta configuración cubre una distancia muy amplia en una sola dirección. Una desventaja clara del sistema es el gran número de nodos del sistema que se despertarán para enviar información a la raíz.



Configuración de árbol degenerado ilustrada.

- ***Estrella:*** Todos los nodos se encuentran en alcance directo a la raíz. Esta configuración tiene poco alcance, al estar limitada por el radio de alcance de la señal del sumidero, pero es la configuración en la que menos nodos serán "molestados" para llevar información hasta el sumidero.



Configuración de árbol en estrella ilustrada.

Realizaremos un estudio sobre la ventaja, calidad y eficiencia de las distintas políticas de wake-up ya mencionados para cada una de estas distintas configuraciones, junto con un análisis de otras ventajas e inconvenientes de cada una de ellas. Nos centraremos sobre todo en costes energéticos y tiempo que cada nodo permanece

despierto, como ya mencionamos al principio de esta sección. Esto nos permitirá investigar las situaciones en las cuales cada configuración es óptima, y cuál es el mejor uso para cada una de ellas.

El objetivo principal del siguiente apartado es combinar las ventajas de los tres sistemas para alcanzar una configuración de red óptima.

3. Diseño de la red

En esta sección discutiremos las distintas decisiones tomadas para el diseño de los nodos, de la red, del algoritmo de organización, y de los distintos tipos de despertar.

3.1. Pautas de diseño. Repaso de requisitos:

Recordemos ciertas condiciones de las redes de sensores que debemos que tener en cuenta a la hora de diseñar tanto nuestros elementos como la red en sí.

El diseño se basará en dos principios: los nodos deben de gastar la menor energía posible, y deben maximizar el uso del tiempo en el que están despiertos.

Veremos que nos comprometemos a sobrecargar ligeramente los nodos que están más cerca de la raíz o del controlador. Estos nodos serían más sencillos de sustituir o reparar. Por otro lado, los nodos más lejanos del sumidero realizan menos cálculos, se mantienen despiertos durante menos tiempo, y guardan menos información.

Esta estrategia es más deseable que sobrecargar el árbol entero, ya que cuanto más lejos los nodos estén del sumidero, probablemente sean más difíciles de sustituir, por lo que es arriesgado delegar demasiada información o responsabilidad sobre esos nodos.

Nuestro algoritmo de organización es costoso en lo que se refiere a tiempo en que los nodos deben de estar activos, pero siempre y cuando nodos nuevos no se unan al sistema frecuentemente, el coste inicial debería de ser compensado por las ventajas de enrutamiento y de creación de caminos gracias a la información que se difunde por el árbol.

3.2. Diseño del nodo del sistema:

Comenzaremos mencionando la información y funcionalidad común a cualquier tipo de nodo del sistema (tanto inalámbrico como controlador), y mencionando las diferencias en el caso que sea necesario.

En todo momento, un nodo del sistema debe conocer tanto a su padre como a sus hijos directos. Almacenamos en cada nodo pues la información necesaria para el envío de tanto un paquete de wake-up (la frecuencia de despertar de estos nodos) como de cualquier otro tipo de mensaje a estos nodos. Esto nos permitirá establecer comunicación directa entre un nodo y su padre/hijos sin molestar a ningún otro nodo.

Un nodo debe conocer la distancia hasta su propio padre y hasta el sumidero del sistema. Este dato ayudará a la hora de organizar los nodos y de encontrar las relaciones de parentesco óptimas.

Un nodo del sistema almacenará información sobre nodos nuevos (sin contar a los hijos directos o al padre) que descubra. El nodo se guardará un identificador para cada nodo y el hijo correspondiente con el que comenzar el camino para conectar con dicho nodo. Esta información de descubrimiento será enviada por un nodo hacia los niveles superiores tras ganar un hijo nuevo: cuando un nodo A establezca a otro nodo B como su hijo, enviará un mensaje con la información del nodo B al padre de A, para que este se anote al nodo nuevo en su tabla.

El resultado de utilizar este sistema es que el nodo raíz tendrá la información necesaria para saber qué hijo utilizar a la hora de establecer un camino hacia cualquier nodo del sistema, ya que es indirectamente padre de todos los nodos del sistema, por lo que le ha llegado información sobre todos los nodos del sistema. Esta información tendrá un tiempo limitado que se refrescará periódicamente cada vez que padre e hijo se envíen un mensaje. Si el tiempo de validez se agota, la información será borrada de los mapas.

Además, y como efecto secundario, observaremos que con este sistema, son los nodos superiores, más cercanos a la raíz aquellos que guardan la mayor cantidad de información (ya que les están llegando información de un mayor número de hijos).

Gracias a esto, podemos empezar a hablar de enrutamiento a través del árbol: Si un nodo A necesita comunicarse con un nodo B con el que no tiene comunicación directa, puede mirar primero en su tabla, para ver si este nodo B es accesible a través de los hijos de A.

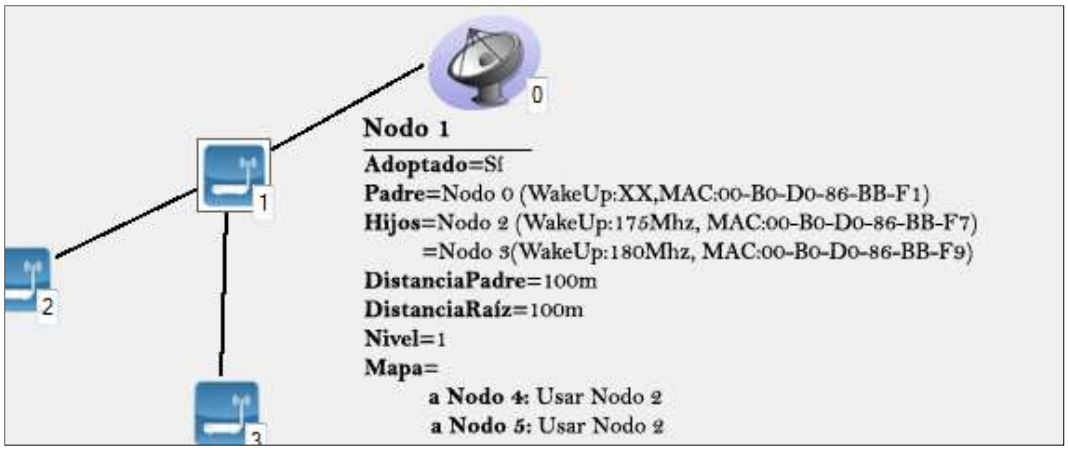
En caso de no poder llegar a un nodo determinado a través de los hijos, el nodo A enviará el intento de wake-up (y subsecuentes mensajes para la comunicación) a su padre, el cual repetirá la operación (buscará al nodo objetivo en su tabla). En algún momento un nodo superior encontrará al nodo B objetivo en su tabla, o llegará a la raíz, el cual necesariamente obtuvo esta información cuando el nodo B entró al sistema.

Se hablará del sistema de enrutamiento con más detalle en la sección correspondiente.

Cada nodo guarda un flag de adopción para hacer más sencillo el sistema de adopción de los nodos. Un nodo no hará ningún tipo de comprobación a la hora de adoptar a un nodo sin padre, y dejará que otros posibles nodos padre juzguen si deben robarle ese hijo o no. Utilizando otra información como la distancia del nodo a adoptar hasta la raíz comparada con la distancia a la que la nueva adopción le dejaría.

Por último, cada nodo se guardará un número que representa el nivel del árbol al que pertenece. Este dato será útil a la hora de adoptar y a la hora de enviar mensajes de difusión por el árbol.

Mostraremos un ejemplo de un nodo con toda su información. Asumimos que el nodo ha adoptado a dos hijos: Nodo 4 y Nodo 5:



Información de un nodo cualquiera del sistema

3.3. Diseño de la configuración y del algoritmo de adopción

En este apartado hablaremos de las decisiones tomadas a la hora de crear el algoritmo que generará el estado por defecto del árbol, detallado en la Sección 2: "Metas y Objetivos del Proyecto", punto 2.2 "Configuración de red y algoritmo generador".

En resumidas cuentas, la configuración por defecto que queremos que genere el algoritmo de organización es la de un árbol relativamente equilibrado, con un número variable de hijos por nodo (a establecer antes de generar el árbol) y una distancia de alcance para los nodos del árbol variable (a establecer antes de generar el árbol).

Estos parámetros afectarán considerablemente la organización de la configuración de red.

El algoritmo que establece la forma del árbol no se ejecuta de forma global, es decir, los nodos no se organizarán desde fuera por otra entidad, sino que serán los nodos mismos los que intenten adoptar a otros nodos para generar el árbol de parentescos, usando su mejor juicio en ese momento, y teniendo en cuenta entre otras cosas el alcance de su señal (distancia), la cantidad de hijos permitidos, y si la adopción lleva al hijo a una mejor situación en el árbol (es decir, más cerca del nodo raíz).

Utilizaremos este sistema para dotar de mayor autoconfiguración a la red, y a obligar a que ciertas situaciones del sistema se solucionen solas sin tener que depender de intervención externa. Por ejemplo, cuando un nodo deje de funcionar por agotar la batería, o cuando un nodo nuevo se introduce en el sistema.

Comencemos a exponer el algoritmo que siguen los nodos para establecer la forma del árbol que buscamos, y para dejar preparado un sistema que permite un fácil enrutamiento de paquetes entre nodos siguiendo las políticas de wake-p ya mencionadas.

Cómo hemos dicho, un nodo puede elegir adoptar (o no) a otro nodo que está al alcance de su antena. Veamos entonces el tipo de decisiones que un nodo debe tomar en cuanto se refiere a la decisión de adoptar:

- Lo primero que un nodo comprueba es su propio estado de adopción. El proceso será empezado por el controlador o raíz, el cual siempre es capaz de adoptar, por lo que se salta este paso. No obstante, los nodos de sensores inalámbricos no podrán adoptar a ningún nodo si no han sido adoptados previamente, para evitar la creación de sub-árboles fuera de la red de sensores principal.

- Lo siguiente que el nodo debe comprobar es que no ha alcanzado el número máximo de hijos permitido. El número de hijos por defecto de nuestro sistema será dos, pero el algoritmo estará diseñado para permitir que este número sea variable. Podemos cambiar este valor en los nodos, por ejemplo, a través de los mensajes correctos, y simplemente tomará efecto la próxima vez que un nodo lleve a cabo el proceso de adopción.

Si un nodo ha detectado otro nodo a su alcance, tiene permitido adoptar más hijos, y ya es parte del árbol que comenzó el sumidero, entonces el nodo se encuentra en condiciones de adoptar. Veamos lo que el nodo comprueba para decir si debe adoptar al nodo hijo futuro:

- Comprueba es que el nodo a adoptar en cuestión no es su nodo padre, ni tampoco puede intentar adoptar al nodo sumidero. Evitamos así conflictos entre dos nodos y mantenemos la estructura jerárquica del árbol.

- Comprueba que el futuro hijo no está en un nivel superior del árbol al del propio nodo. Ya comentamos en el punto 3.1 de este mismo apartado que un nodo se guardará el nivel del árbol al que pertenece, y que ese dato se iba a

guardar para facilitar el proceso de adopción. Este número comenzará por 0 en la raíz (el más alto) y se incrementará a cada nivel.

Esta comprobación impide que aparezcan ciclos en los sub-árboles del sistema. Esto podría ocurrir si por ejemplo un nodo de un nivel inferior se considerara a sí mismo como un mejor padre de un nodo de un nivel superior, y al adoptar cortara el camino de ese nodo hacia la raíz. Nos evitamos esta situación si todo nodo debe comprobar que su nivel es menor numéricamente que el nivel del candidato a hijo.

- Lo siguiente que un nodo comprueba es que el futuro hijo no tenga un padre (a través del flag de adopción mencionado en el punto 3.1 de esta sección). Si este es el caso, entonces lo adopta directamente.

  En caso de que el nodo objetivo SÍ tiene un padre: el nodo debe juzgar si sería un padre mejor que el padre original. Comprobará que puede dejar al nodo más cerca de la raíz que su padre anterior, utilizando los campos de distancia al padre y distancia a la raíz del nodo que se está intentando adoptar. Dejaremos un margen en el cual no se realizará adopción incluso si nodo que intenta adoptar sería un mejor padre, ya que no merece la pena realizar el proceso de adopción si la diferencia entre dos padres es muy pequeña. Veremos más adelante el por qué.

Como ya hemos visto, la responsabilidad de descubrir relaciones de parentesco óptimas cae completamente en los padres. Por lo que una vez un padre ha realizado las operaciones necesarias para decidir si un nodo debe ser adoptado, se envía un mensaje de adopción que el hijo simplemente acepta, y ocurre así la adopción.

Una vez que el nodo es adoptado:

El nodo padre se debe encargar entonces de proporcionar cierta información al nuevo hijo, entre ella el nivel al que pertenecerá, quién es su nuevo padre, etc., además de guardarse la información necesaria para establecer comunicación con el hijo. El nuevo nodo hijo se encargará de poner su flag de adopción a “cierto” (si no tenía ya un padre), y de guardarse la información necesaria de su nuevo padre para poder establecer comunicación con el mismo.

En el caso de que el nodo adoptado tuviese un padre, deberá despertarlo y enviarle un mensaje indicando que ha sido adoptado por otro nodo. El padre anterior se encargará de borrar la información sobre el antiguo hijo.

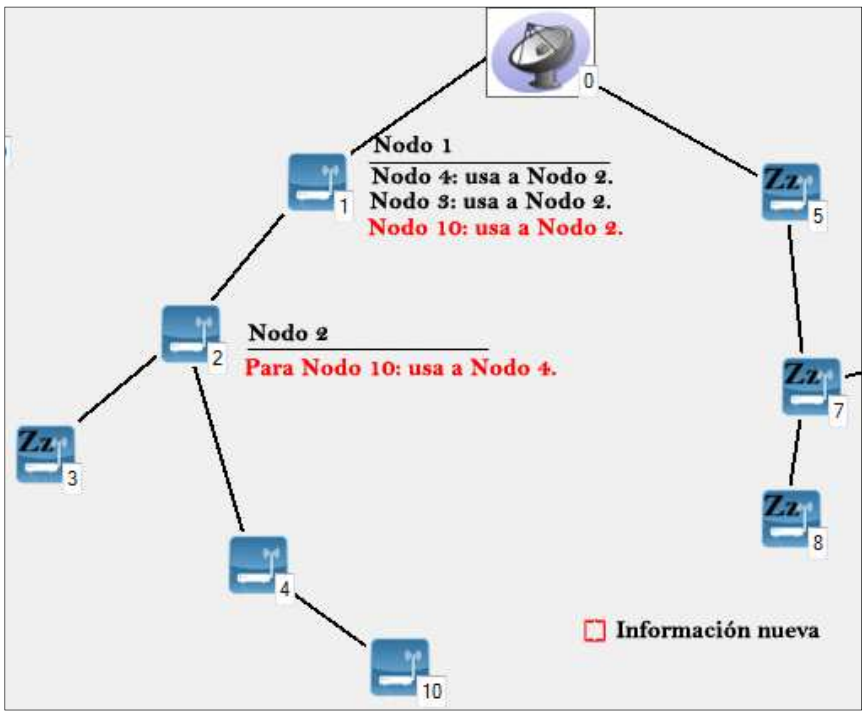
El nodo padre (A) entonces enviará un paquete a su propio padre (B), para indicarle a su padre dos cosas: una: la existencia de un nuevo nodo en el sistema (C) (o que la situación de un nodo existente ha cambiado, en el caso de que el nodo C tuviera un padre antes de esta adopción), y dos: que si B desea despertar o enrutar algo hacia C, debe usar a A como camino.

Esta información se reenvía hacia los niveles superiores del árbol, hasta llegar al nodo raíz del sistema. Lo que cambia en cada envío es la información del nodo que debe usarse como siguiente salto, la cual siempre corresponde al nodo que realizará el siguiente envío.

Si un nodo pierde un hijo, reenvía la información del mapa hacia los niveles superiores del árbol, para que la entrada de su hijo perdido se borre en todos los nodos superiores. Con esto nos evitamos que los nodos superiores intenten llegar a un nodo a través de un tercero que ya no sirve de camino o de acceso directo para ese nodo.

Se realiza así un mapeo progresivo en los nodos sobre qué camino tomar cuando quieren enviar un paquete hacia los niveles inferiores del árbol. Como ya hemos mencionado en varias ocasiones, los niveles superiores del árbol estarán más sobrecargados con información debido a tener un mayor número de hijos en sus sub-árboles.

Veamos un ejemplo gráfico del mapeo en el que se muestran los nodos afectados por el proceso de envío de información. En este ejemplo, el nodo 4 acaba de adoptar al nodo 10, y envía esa información al nodo 2, su padre. El nodo 2 entonces repite el proceso con el nodo 1, el cual acaba enviándoselo a la raíz:



Este algoritmo de organización también es aplicable en el caso en que algún nodo del árbol deje de funcionar. Como ya mencionamos en el apartado del diseño de nodos, podemos añadir un tiempo de validez a los datos.

En particular, los campos de padre, hijos y mapas tienen un tiempo limitado de validez. Si estos campos no se refrescan cada cierto tiempo, se asumirá que el nodo del que ya no se reciben paquetes ha dejado de funcionar, y entonces estos datos pueden ser simplemente borrados.

En el caso de que un padre pierda un hijo por inactividad, se sigue la misma estrategia que cuando se pierde un hijo por adopción: se borran las entradas de ese hijo, y la información del mapa se envía hacia los niveles superiores. En el caso de que un hijo borre a un padre por inactividad, pondrá todos sus campos al valor por defecto (borrará todo) y simplemente esperará a ser adoptado, como si acabara de entrar al sistema.

Concluye así el algoritmo de organización de nuestro sistema.

3.3.1. Uso del algoritmo de adopción

Como puede verse a simple vista, el proceso de adopción es relativamente costoso en lo que se refiere al número de nodos que deben permanecer despiertos. Es por ello que idealmente, los nodos sólo intentarán adoptar en intervalos determinados, tras un mensaje procedente de la raíz, o tras la inclusión de nodos nuevos al sistema.

Un único proceso de adopción mantiene despiertos a dos nodos y llega a despertar a un tercero en el caso de robo de un hijo, sin incluir el despertar de gran parte del árbol ya que es necesario enviar hacia los niveles superiores la información de mapeo de nodos después de cada adopción, por lo que el número real de nodos que participan en cada adopción es bastante mayor en la realidad. Una adopción con robo a niveles inferiores podría fácilmente despertar a todo el árbol ya que el nuevo padre y el antiguo padre deben enviar la información de mapeo a los niveles superiores.

Este proceso claramente no es algo deseable. Si los nodos usan un tiempo significativo peleándose por decisiones de adopción, el tiempo disponible para operaciones útiles en el sistema cae, y es posible que desaparezca la ventaja que algunas adopciones hubieran podido proporcionar. Ya mencionamos en su momento que dejamos un margen para la diferencia entre la calidad de un padre u otro, para asegurarnos de que sólo hay robos si realmente merece la pena.

El proceso de adopción es un proceso iterativo: su coste es mayor si el árbol acaba de formarse, y a medida que los padres óptimos son encontrados, ocurrirán menos adopciones, ya que la condición de encontrar a un padre mejor no será frecuente, por lo que habrá menos difusión de mapas de enrutamiento.

La entrada de un nodo nuevo al sistema comenzará con ese nuevo nodo esperando a que alguien despierte. Una vez algún nodo despierte, podrá adoptar a este nuevo nodo.

Siempre y cuando un nodo despierte, se enviará un paquete de adopción a los nodos de su alrededor. Aquellos nodos que estén despiertos para recibir este mensaje podrán entonces ser re-adoptados, si eso llevaría a una mejor organización del árbol.

En el caso de que haya una adopción, una gran parte del árbol acabará despertándose debido al mapeo ya explicado. Estos nuevos nodos que despierten por culpa de la adopción inicial podrán entonces llevar a cabo sus propios intentos de adopción para continuar con la reestructuración del árbol.

Como hemos mencionado, es muy probable que si la estructura del árbol es relativamente reciente, un proceso de adopción desencadene en todo el árbol despertándose y tratando de adoptar, y en la mayoría de los casos, esas adopciones ocurrirán, lo cual hacer que intervengan aún más nodos.

No obstante, con el paso del tiempo el árbol llegará a estabilizarse y las peticiones de adopción no serán aceptadas (ya que todos los nodos habrán encontrado a su padre óptimo), y el árbol sólo se despertará cuando un nodo nuevo entre al sistema. Cabe añadir que cuanto mayor sea el margen que permite ignorar el robo de un hijo, más fácil será llegar a una situación estable en el árbol.

3.4. Diseños de las funcionalidades del sistema

Como ya se ha mencionado en diversas ocasiones, buscamos poder establecer comunicación entre dos nodos cualquieras del árbol a través de otros nodos. Para poder establecer comunicación con un nodo, es muy probable que haya que despertarlo antes.

Recordemos pues el tipo de políticas de wake-up que buscamos poder realizar en nuestro sistema, mencionados ya en la sección 2, punto 2.3. Los dividiremos en políticas más simples, que se combinarán para darnos la funcionalidad deseada en la red de sensores:

- Un nodo A cualquiera debe ser capaz de generar y camino de nodos hasta la raíz del sistema ***(despertar 2).*** Por lo tanto, debe ser capaz de despertar a su propio padre ***(despertar 1).***

- Un nodo A cualquiera (incluyendo a la raíz del sistema) debe ser capaz de despertar y crear un camino de nodos hasta un nodo B cualquiera del sistema ***(despertar 3).***

- Un nodo A cualquiera debe ser capaz de despertar a todos sus hijos **(despertar 4)**. Este despertar debe poder extenderse hasta las hojas ***(despertar 5).***

Vemos 5 tipos de despertares definidos:

- **Despertar 1:** Si repasamos el punto de Diseño de Nodos en esta misma sección, ya indicamos que todo nodo tiene la información necesaria para despertar y enviar mensajes a su propio padre (la frecuencia necesaria para tal nodo, y la dirección del mismo).

- **Despertar 2:** Extendemos en el despertar 1 para el despertar 2: los medios y la información están preparados, y sólo necesitamos indicar de alguna manera que el mensaje de despertar debe de ser reenviado hacia los niveles superiores del árbol hasta que no queden padres (es decir, hasta que sea recibido por la raíz).

- **Despertar 3:** Este despertar es probablemente el más complicado y el que mayor información necesita. No obstante, la labor se simplifica bastante gracias al mapeo de nodos e hijos mencionado en apartados anteriores.

  Lo primero que hace el nodo A es buscar al nodo B a despertar entre sus hijos. Si el nodo B se encuentra entre sus hijos, entonces el despertar es directo, y como mencionamos en el primer apartado de esta sección, tenemos la información y medios necesarios para despertar a un nodo hijo. El nodo A simplemente envía un paquete de despertar al nodo B, y comienza la comunicación entre ellos.

  En el caso de que el nodo no se encuentre entre los hijos, el siguiente paso que debe tomar el nodo A es buscar al nodo B en el mapa de nodos que tiene guardado. Si el nodo B se encuentra en el sub-árbol inferior a nuestro nodo A, entonces aparecerá en el mapa, ya que algún hijo del sub-árbol ha enviado la

información hacia arriba tras adoptarlo necesariamente (véase el algoritmo de adopción ya mencionado). Tras encontrarlo, despertamos al hijo que empieza el camino hacia el nodo B, e informamos a este hijo que continúe construyendo el camino de nodos hasta llegar a B (este hijo sigue el mismo sistema de despertar 3).

Si el nodo B objetivo no aparece en el mapa, entonces podemos asumir que no está en nuestro sub-árbol. Enviamos entonces la petición de despertar al padre de A, el cual repetirá este proceso para encontrar y llegar a B.

- **Despertar 4:** Tenemos la información y medios para despertar a un hijo, como ya se ha mencionado varias veces. Simplemente se repite este proceso hasta que se haya enviado un despertar a todos los hijos de un nodo.

- **Despertar 5:** Este tipo de despertar simplemente extiende el Despertar 4: necesitaremos algo para indicar que el despertar debe de ser repetido para todos los hijos del nodo al que despertemos, por ese nodo. El proceso se detiene cuando un nodo no tiene hijos.

Finaliza así el diseño de políticas de wake-up para nuestro sistema.

Cabe añadir que los datos guardados y mecanismos usados para hacer posibles estos despertares han sido elegidos en base a una configuración de árbol específica (la de árbol balanceado). El estudio a realizar más adelante se encargará de comentar las ventajas y desventajas de los diseños de políticas de wake-up usadas, tanto para esta configuración de red como para otras, y finalizará con una reflexión sobre los cambios de diseño que podríamos realizar para cada configuración.

4. Componentes de la simulación de red

4.1. Introducción a la simulación

La simulación de la red se divide principalmente en tres partes:

- Por un lado se ha desarrollado una interfaz gráfica programada en C#, en particular con Windows Forms de Visual Studio, con la que el usuario puede colocar los nodos, elegir la distancia de la señal, y el número de hijos permitidos para cada nodo de manera cómoda.

  Esta interfaz gráfica se utilizó para depurar ciertos algoritmos por lo que además simula el funcionamiento del algoritmo que sigue cada nodo a la hora de adoptar, y por además emula distintas políticas de wake-up que se pueden realizar entre los diversos nodos.

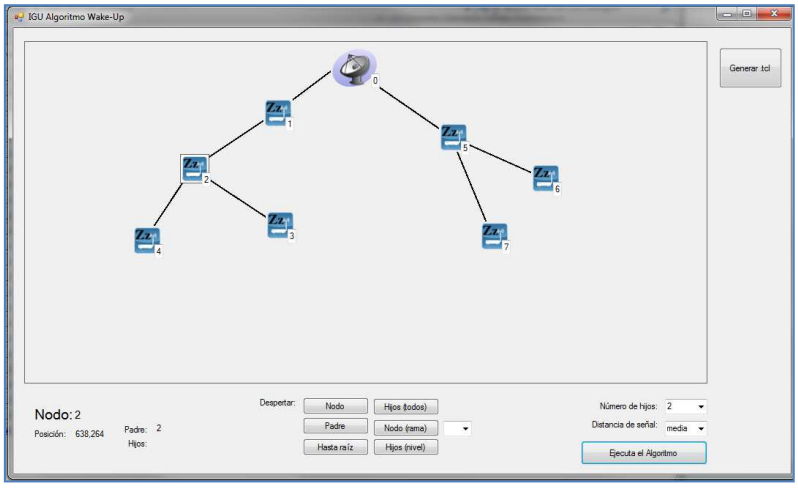


Figura 1: Definición de la red en la interfaz de usuario

- Por otro lado tenemos el simulador de red en sí: el simulador de red ns-2. Este simulador crea una simulación a partir de cierta información generada por la aplicación ya mencionada, y simula los nodos de sensores, el nodo sumidero, instala los protocolos de red adecuados, simula los medios del sistema, y establece una comunicación real entre ellos. Para obtener un mejor resultado y facilidad a la hora de construir la simulación, elegimos el simulador ns-2.

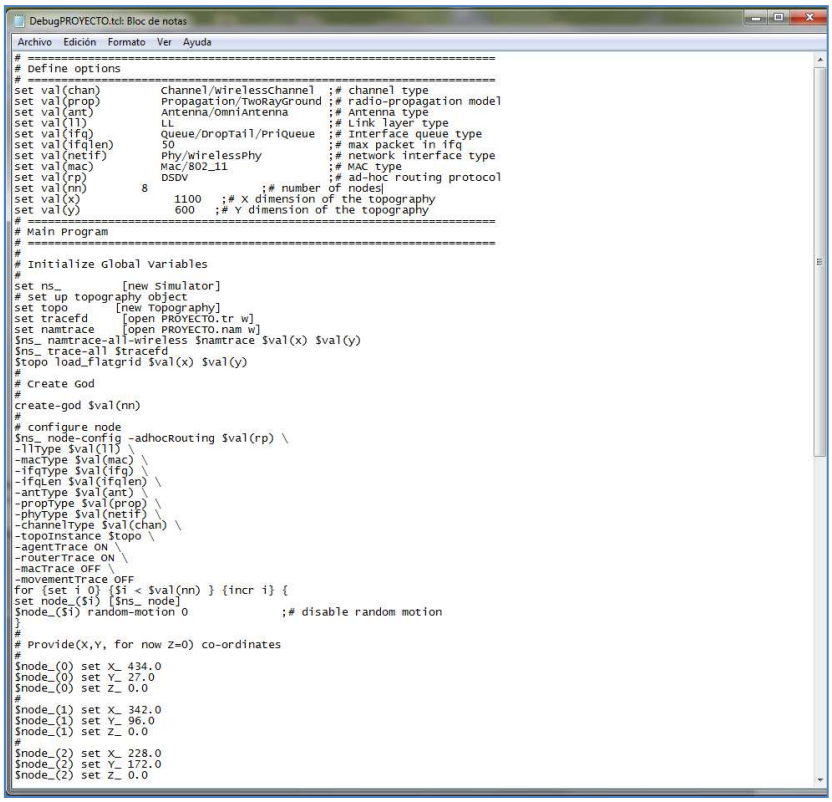


Figura 2: Archivo .tcl generado por la aplicación con la red definida, que se usará en el simulador

- Por último tenemos un visualizador de simulaciones, llamado nam, que principalmente nos muestra la simulación de la red generada por ns-2 de forma más visible y entendible. El visualizador nam además ofrece una visión clara sobre los paquetes enviados/recibidos.

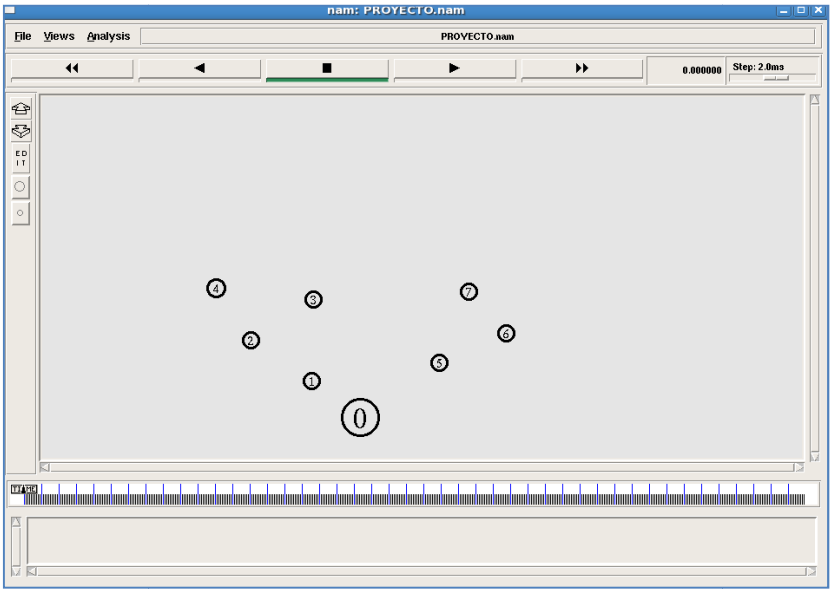


Figura 3: La red en nam

4.2. Interfaz de usuario para la especificación de red

La primera parte de la simulación consiste en la programación de una aplicación de interfaz de usuario en Visual Studio 2008 (para más información, véase la sección 8, Bibliografía). Los objetivos de esta aplicación son dos:

- Ofrecer una interfaz a través de la cual observar el comportamiento del algoritmo y de las técnicas de wake-up de forma gráfica, antes de entrar a una simulación real.

  Una vez lanzada la aplicación, el usuario puede colocar diversos nodos en un espacio, y generar funciones de parentesco entre los mismos. Acto seguido es posible ejecutar las diversas políticas de despertar en el sistema ya explicadas en apartados anteriores.

- Generar un archivo a través del cual cargar una configuración de red real con el simulador

La motivación de programar la aplicación surgió en las etapas de planificación del proyecto, antes de empezar con el uso de simuladores de redes reales.

4.3. Simuladores de red

Para la simulación de las redes diseñadas en este proyecto se consideraron principalmente dos simuladores de red: el ns-2 y el ns-3. Veamos un poco de información sobre cada uno de ellos, y continuemos con las ventajas y problemas encontrados a la hora de utilizar los mismos.

4.3.1. Ns-2

El simulador de red ns-2 es un simulador de eventos para el tema de investigación de redes. Ns-2 tiene un buen soporte para la simulación de TCP, enrutado y protocolos multicast sobre redes cableadas e inalámbricas. El simulador ns-2 está escrito en C++.

Para realizar simulaciones sobre ns-2, es necesario escribir un script TCL (tool command language), en el cual definir la topología, los parámetros de la red y las acciones a realizar en la simulación. La versión de TCL que utiliza ns-2 es un dialecto orientado a objetos llamado OTcl.

Veamos las características de ns-2 y el soporte de simulaciones que ofrece:

- Para simulaciones cableadas
  - ✓ Enrutamiento por vector de distancias
  - ✓ Enrutamiento por estado de enlace
  - ✓ Enrutamiento por PIM-MD *(Prot. Independent Multicast-Dense Mode)*
- Para simulaciones inalámbricas
  - ✓ Enrutamiento ad-hoc
  - ✓ Difusión dirigida
- Protocolos de transporte: TCP, UDP, SCTP
- Tipos de tráfico: web, ftp, telnet, etc.

Y veamos los componentes del ns-2:

- **El simulador ns-2**
- **El visualizador nam**, que se encarga de leer la traza generada por el simulador para presentarla de forma gráfica

Ns-2 suele ser criticado por su complejidad. El modelado de redes es algo relativamente complejo y lleva bastante tiempo (como pudimos comprobar) debido a la necesidad de aprender a utilizar el lenguaje TCL entre otras cosas.

Uno de los problemas a la hora de utilizar ns-2 para el proyecto aparece con la necesidad de especificar el tipo de nodo a usar. Como ya vimos en su apartado, nuestros nodos a utilizar no son simples nodos inalámbricos, sino que poseen ciertas características como la posibilidad de apagarse y encenderse (wake-up). Ns-2 ofrece modelos de nodo inalámbrico, pero ajustarlos a nuestro tipo de nodo no es tan sencillo.

Para simular nuestros nodos, ajustaremos los parámetros de los nodos inalámbricos de ns-2 para adaptarlos a nuestra especificación de nodo bajo consumo cambiando los costes de envío y recepción.

El otro problema aparece a la hora de especificar un protocolo de enrutado, para ser específicos, el definido en el apartado de Diseño. Para definir un protocolo de enrutado en ns-2 es necesario modificar el código del simulador en sí. Debido a la complejidad de nuestro protocolo diseñado, es difícil ajustarlo a los modelos que ofrece ns-2 por defecto.

4.3.2. Ns-3

El simulador de red ns-3 es un simulador de redes para sistemas de internet (con énfasis en las capas 1-4), diseñado principalmente para investigación y educación. Posee además integración de ciertas herramientas de red, como Wireshark o tcpdump.

Ns-3 fue creado para sustituir al simulador ns-2. Debido a la evolución de diversos sistemas, ns-2 tuvo que adaptarse más allá de sus metas iniciales. Ns-3 ofrece un nuevo comienzo, con el objetivo de ser más fácil de utilizar y más extensible.

Ns-3 está escrito en C++, tanto el código del simulador en sí como los ficheros de simulación de los usuarios. Además utiliza python para scripting y para la visualización por parte del usuario.

Veamos algunas de las características del simulador ns-3:

- Simulación muy sofisticada
  - ✓ Muchas opciones de parametrización
  - ✓ Trazas configurables que permiten depositar resultados en logs de texto o en PCAP (para visualizar por tcpdump, por ejemplo)
  - ✓ Modela el stack IP real: los nodos poseen dispositivos de red con IPs propias
  - ✓ Emula el stack TCP/UDP
  - ✓ Soporta IPv4, IPv5
  - ✓ Soporta el enrutamiento Dijkstra y MANET
  - ✓ Soporta IEEE 802-11 y sus variantes, además de las capas físicas de IEE 802
  - ✓ Tiene varios modelos de movilidad: Dirección aleatoria, TW, TWP, 3D GMM
  - ✓ API de sockets basada en eventos
  - ✓ Los paquetes poseen bytes serializados, etiquetas, y metadatos, lo que permite una fácil extensión y traza de los mismos
- Es muy orientado objetos
  - ✓ Control de memoria automático
  - ✓ Agregar estados y comportamientos nuevos a objetos es sencillo

Veamos los elementos arquitectónicos del ns-3:

- Los nodos poseen varias características, por ejemplo movilidad
- Como ya se ha mencionado, los nodos poseen dispositivos de red:
  - ✓ Estos dispositivos envían paquetes por canales
  - ✓ Incorporan la Capa 1 y Capa 2

- Los dispositivos tienen una interfaz con la Capa 3
- La capa 3 soporta la Capa 4
- La capa 4 es usada por objetos de Capa 5

4.4. Visualizador de red

El visualizador de red NAM funciona con ns-2 y con ns-3, y nos da una visualización de la simulación de red especificada. Algunas características:

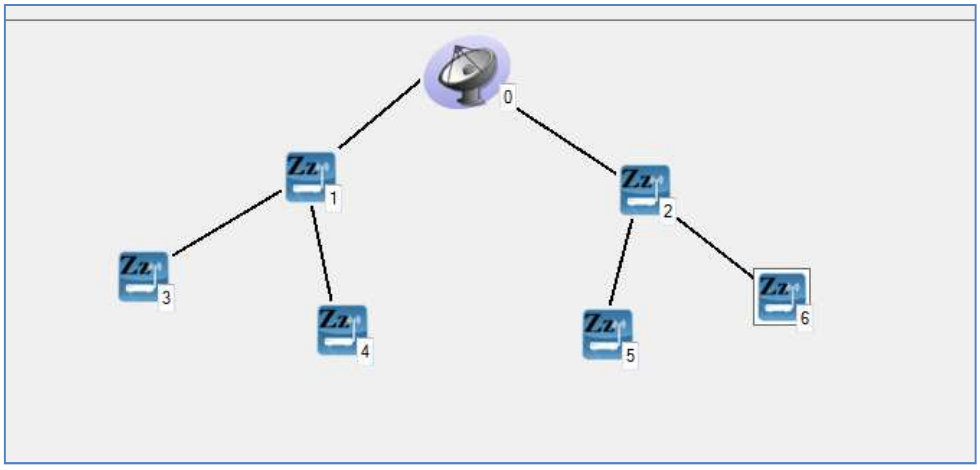
- Interpreta visualmente la red creada
- Ejecuta un script .tcl o un script .nam generado por la ejecución de un script .tcl
- Tiene varios controles, entre ellos Play, Stop, Fast-Forward, Pause, y nos permite controlar la velocidad de la simulación y ver los paquetes enviados en la misma
- Muestra información como el número de paquetes de cada elemento

5. Estudio de redes de sensores utilizando distintas configuraciones de red

5.1. Configuración en árbol equilibrado

Una configuración en árbol equilibrado es aquella configuración en que los nodos se encuentran a una relativamente larga distancia del sumidero, lo cual fuerza al sistema a establecer conexiones de parentesco entre los nodos.

En una configuración en árbol, el sumidero es padre de un número relativamente pequeño de nodos en comparación con el número total de nodos del sistema. Los nodos del sistema son a su vez padres de un determinado número de nodos (en nuestro caso consideramos a un árbol equilibrado a aquel árbol cuyos nodos tienen siempre 2 como máximo).



Cuando un nodo cualquiera quiere enviar un mensaje en este tipo de configuración, debe crear un camino de nodos hasta llegar al nodo destino del mensaje. Un mensaje enviado en un árbol equilibrado fuerza a despertar a los dos nodos relacionados en el mensaje y a todos los nodos utilizados como camino.

Bajo el punto de vista de la energía que gasta el sistema: el coste de un mensaje cualquiera esta directa y fuertemente atado al nivel en el que se encuentren ambos el nodo emisor y el nodo receptor, y a la posición de los nodos en el árbol en sí.

Vemos que el peor caso de mensaje es aquel en el que un nodo hoja en un extremo del árbol intenta establecer un camino hasta llegar a otro nodo hoja en el otro extremo del árbol y el mensaje debe pasar por la raíz.

Estudiemos distintas estimaciones de costes de energía para este tipo de árbol, basándonos en algunos de las políticas de wake-up del apartado de Diseño. Por la simplicidad de algunos de ellos, nos vamos a centrar en dos tipos de despertares:

despertar 2, en el que un nodo cualquiera despierta a un camino de nodos hasta llegar a la raíz del sistema – y despertar 3, en el que un nodo cualquiera (incluyendo a la raíz) despierta un camino de nodos hasta llegar a un nodo cualquiera del sistema.

Podemos calcular el coste de cada despertar a partir de dos costes:

1. *El coste energético de del sistema para el despertar – el cual obtenemos con el número de nodos involucrados en el despertar, junto con el coste energético de despertar estos nodos.*
2. *El coste energético de nodos del sistema – el cual obtenemos a partir del número de nodos del sistema involucrados, el coste energético de los mismos por mantenerse despiertos, y el tiempo que se mantienen despiertos.*

Estos costes son entonces comparados con el coste de utilizar a todos los nodos del sistema – cuanto más cercanos los dos costes sean, menos eficiente es el despertar.

Nos centraremos principalmente en calcular el número de nodos involucrados para cada despertar, ya que el resto de parámetros (coste de despertar a un nodo, coste de mantener a un nodo despierto, etc.) son constantes y no están relacionados con la configuración del árbol. El tiempo que los nodos pasan despiertos está relacionado con los mensajes enviados tras el despertar, pero el tema de coste de mensaje se sale de los objetivos del proyecto, ya que no estamos centrando en los costes de despertar. Asumiremos un tiempo constante.

Veamos pues los despertares hablados para un árbol en configuración de árbol equilibrado. Recordemos que un árbol equilibrado es aquel en el que los nodos tienen como mucho dos hijos. Asumiremos que los nodos tienen todos dos hijos.

- **Despertar 2 – Nodo despierta nodos hacia la raíz del sistema:**

  El coste depende muy ligeramente del nivel del nodo. Un nodo cualquiera despertará a su padre hasta llegar a la raíz del sistema, y obviamente nodos en niveles superiores del árbol (más cercanos a las hojas) tendrán más ancestros que despertar. Empecemos calculando el número de nodos involucrados en un despertar cualquiera:

  - ***Número de nodos utilizados =*** *Nivel del emisor en el árbol*

  No olvidemos que los nodos están repartidos en ramas, y pueden verse no molestados en el despertar de una rama específica. También debemos tener en cuenta los nodos que no serán molestados en la propia rama.

  - ***Número de nodos totales*** = 2 ^ (Número total de niveles del árbol +1) -1

Podemos ver claramente que el *Nivel del emisor en el árbol* siempre va a ser un número mucho menor que el *Número total de nodos del sistema*.

Aquí está la eficiencia de este tipo de configuración: para el despertar 2, el número de nodos utilizados en un despertar es mucho menor que el número de nodos totales del árbol equilibrado.

Vemos que el coste total sería:

Coste total *= Nivel del emisor en el árbol * ((Coste de mantener a un nodo despierto * tiempo que pasan los nodos despiertos) + Coste de despertar al nodo)*

- **Despertar 3 – Nodo despierta nodos hacia un nodo cualquiera del sistema:**

  El coste depende ligeramente del nivel del emisor y del receptor. Para despertar a un nodo cualquiera, el nodo emisor deberá tomar una decisión: despertar en su propio sub-árbol, o despertar al padre.

  El resultado de la primera decisión es relativamente sencillo – simplemente es realizar un camino en el propio sub-árbol hasta encontrar al receptor. La operación está completamente aislada dentro de este subárbol, y el Número de nodos involucrados en la búsqueda será pequeño comparado con el Número de nodos totales.

  - ***Número de nodos utilizados en el camino de un subárbol =*** *|Nivel del emisor– Nivel del receptor| + 1*

  - ***Número de nodos totales*** = 2 ^ (Número total de niveles del árbol +1) -1

  Veamos el resultado de realizar un camino hacia el padre: debido a la información que guardan nuestros nodos, es posible lanzar un mensaje hacia el padre hasta llegar a un ancestro capaz de llegar al nodo receptor – al ocurrir esto, el nodo ancestro se encarga de construir un camino hacia este receptor.

  Dividimos el cálculo de este coste en dos pasos: el cálculo de encontrar a un padre capaz de continuar el camino, y el cálculo de crear este camino.

  Podemos utilizar el coste de llegar hasta la raíz, sustituyendo a la raíz por un nodo cualquiera:

  - ***Número de nodos utilizados para encontrar un padre cualquiera =*** *|Nivel del emisor - Nivel del padre| +1*

Y a esto le sumaremos el número de nodos necesarios para encontrar un hijo:

- ***Número de nodos utilizados en el camino de un subárbol =*** *|Nivel del emisor – Nivel del receptor| +1*

Teniendo en cuenta que el padre es usado dos veces, nos quedamos con lo siguiente:

- ***Número de nodos utilizados para encontrar un nodo cualquiera =*** *|(Nivel del emisor - Nivel del padre)| + |(Nivel del padre – nivel del receptor)| +1*

Teniendo en cuenta que cuesta 0 despertar y mantener despierta a la raíz, el coste total sería:

- ***Coste total*** *= |(Nivel del emisor - Nivel del padre)| + |(Nivel del padre – nivel del receptor)| +1 * ((Coste de mantener a un nodo despierto * tiempo que pasan los nodos despiertos) + Coste de despertar al nodo)*

El árbol equilibrado ofrece bastante eficiencia en cuanto a despertares se refiere – los nodos involucrados siempre son menores que los nodos totales, y incluso en los peores casos, no se despertarán a todos los nodos del árbol. Es además posible de implementar en sistemas reales, ya que nos permite cubrir cierta distancia.

El problema viene con la necesidad de utilizar más nodos que un árbol degenerado para cubrir distancia en una sola dirección, o corremos el riesgo de degenerar ramas de nuestro árbol.

No obstante, tampoco podemos dejar por mencionar que mantener un árbol equilibrado puede ser algo costoso si se introducen nuevos nodos al sistema, ya que el sistema debe ser capaz de re-estructurarse para mantener la propiedad de equilibrio, y esto no es siempre posible en un sistema real.

El coste de establecer un camino desde una hoja a la rama es logarítmico con el número de nodos – debido a la organización en ramas equilibradas de todos los nodos.

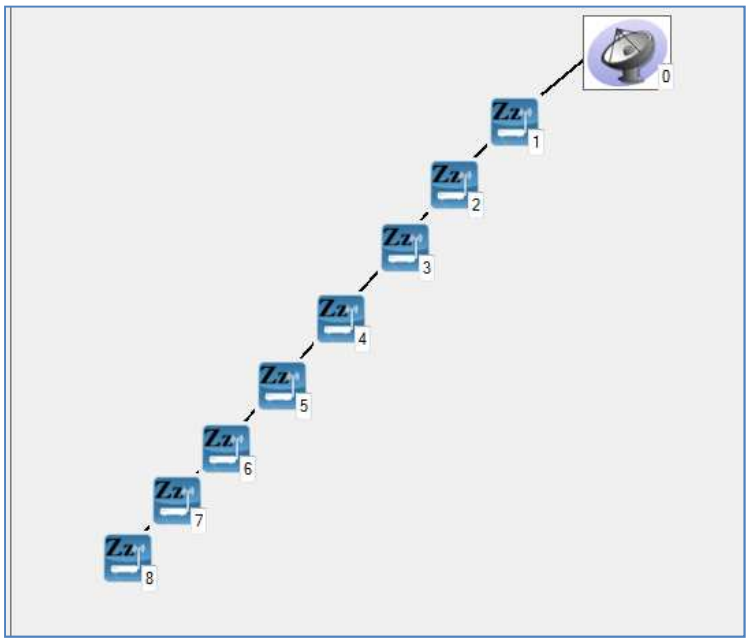
A pesar de las desventajas y de la imposibilidad de mantener un árbol equilibrado real en ciertas situaciones, y como ya hemos mencionado, cualquier aproximación a un árbol equilibrado ofrece grandes ventajas a la hora de gastar energía en despertares y de cubrir distancia al mismo tiempo.

Véase el fichero "EQUIL.tcl" para ver la simulación de un despertar en ns-2/nam para una configuración de árbol degenerado.

5.2. Configuración en árbol degenerado

Una configuración en árbol degenerado es aquella configuración en la que la mayoría de los nodos se encuentran relativamente alejados unos de otros, y todos se encuentran situados en la misma dirección en el espacio. Esto fuerza al árbol a generar conexiones de parentesco en las que cada nodo tiene como mucho un hijo, y existe generalmente sólo una rama en la que se encuentran todos los nodos.

En esta configuración, cada nodo tiene un único subárbol y la organización del árbol puede compararse a la de una lista.



Cuando un nodo cualquiera quiere enviar un mensaje en este tipo de configuración, debe crear un camino de nodos hasta llegar al nodo destino del mensaje. Un mensaje enviado en un árbol degenerado fuerza a despertar a los dos nodos relacionados en el mensaje y a todos los nodos que se encuentren entre los mismos.

La gran diferencia entre el camino de un árbol degenerado y el camino de un árbol equilibrado es que el camino de un árbol degenerado aumenta considerablemente con el nivel del nodo involucrado en el mensaje, al sólo existir una rama de nodos en el árbol. Básicamente podríamos decir que sólo existe un camino a tomar en el árbol degenerado: la cuestión se encuentra en cuanto camino necesitaremos utilizar.

Bajo el punto de vista de la energía utilizada por el sistema: el coste de un mensaje está fuertemente atado al nivel ambos el nodo emisor y receptor. En el caso del árbol

degenerado, llegar a la raíz desde una hoja (o viceversa) es el peor caso, ya que para llegar a la raíz, un nodo hoja deberá pasar por básicamente todos los nodos del sistema, despertándolos en el proceso.

Estudiemos distintas estimaciones de costes de energía para este tipo de árbol, basándonos en algunas de las políticas de wake-up del apartado de Diseño. Por la simplicidad de algunos de ellos, nos vamos a centrar en dos tipos de despertares: despertar 2, en el que un nodo cualquiera despierta a un camino de nodos hasta llegar a la raíz del sistema – y despertar 3, en el que un nodo cualquiera (incluyendo a la raíz) despierta un camino de nodos hasta llegar a un nodo cualquiera del sistema.

Podemos calcular el coste de cada despertar a partir de dos costes:

1. *El coste energético de del sistema para el despertar – el cual obtenemos con el número de nodos involucrados en el despertar, junto con el coste energético de despertar estos nodos.*
2. *El coste energético de nodos del sistema – el cual obtenemos a partir del número de nodos del sistema involucrados, el coste energético de los mismos por mantenerse despiertos, y el tiempo que se mantienen despiertos.*

Estos costes son entonces comparados con el coste de utilizar a todos los nodos del sistema – cuanto más cercanos los dos costes sean, menos eficiente es el despertar.

Nos centraremos principalmente en calcular el número de nodos involucrados para cada despertar, ya que el resto de parámetros (coste de despertar a un nodo, coste de mantener a un nodo despierto, etc.) son constantes y no están relacionados con la configuración del árbol. El tiempo que los nodos pasan despiertos está relacionado con los mensajes enviados tras el despertar, pero el tema de coste de mensaje se sale de los objetivos del proyecto, ya que no estamos centrando en los costes de despertar. Asumiremos un tiempo constante.

Veamos pues los despertares hablados para un árbol en configuración de árbol degenerado:

- **Despertar 2 – Nodo despierta nodos hacia la raíz del sistema:**

  El coste depende directamente del nivel del nodo. Los niveles más lejanos a la raíz aumentan el número de nodos involucrados en el despertar considerablemente. Un despertar iniciado desde la hoja utilizará a todos los nodos del sistema como camino, requiriendo despertar a todos los nodos.

  - ***Número de nodos utilizados =*** *Nivel del emisor en el árbol*

Como podemos ver, este despertar puede ser muy poco eficiente dependiendo del nivel del nodo que desee llegar hasta la raíz**, ya que el número total de nodos del árbol es igual al nivel más alto del árbol**.

- ***Número de nodos totales*** = Nivel más alto del árbol

Podemos calcular el coste de este tipo de despertar de la siguiente manera:

- ***Coste total del despertar:*** *Nivel del emisor * ((Coste de mantener a un nodo despierto * tiempo que pasan los nodos despiertos) + Coste de despertar al nodo)*

- **Despertar 3 – Nodo despierta nodos hacia un nodo cualquiera del sistema:**

  El coste depende del nivel en que se encuentra los dos nodos involucrados en el despertar: importan tanto el nivel del nodo emisor como el del receptor, y lo más cercanos que sus niveles sean, mejor será en cuanto al número de nodos involucrados se refiere. Despertares entre nodos que se encuentran en niveles medios del árbol serán menos costosos que los despertares entre extremos.

  - ***Número de nodos utilizados:*** *|Nivel del emisor – Nivel del receptor| +1*

  - ***Número de nodos totales*** = Nivel más alto del árbol

  Podemos calcular el coste de este despertar de la siguiente forma, tratando al nodo raíz como un nodo cualquiera cuyos costes son 0:

  - ***Coste total del despertar:*** *|Nivel del emisor – Nivel del receptor| +1 * ((Coste de mantener a un nodo despierto * tiempo que pasan los nodos despiertos) + Coste de despertar al nodo)*

Como conclusión, podemos ver que debido a que el *Número de nodos involucrados en los despertares* se aproxima demasiado al *Número de nodos totales del árbol* si la comunicación no es entre nodos muy cercanos, estos despertares son muy poco eficientes en este árbol.

El coste de hacer despertares siempre aumenta con nada nodo nuevo del sistema. Cada nodo que introduzcamos requerirá un nuevo salto para llegar a la raíz, es decir, un nuevo nodo a despertar para llegar desde la raíz a una hoja y viceversa.

Es recomendable evitar este tipo de configuración excepto en los casos en que la comunicación ocurra sólo en los niveles medios del árbol… aunque debido a la

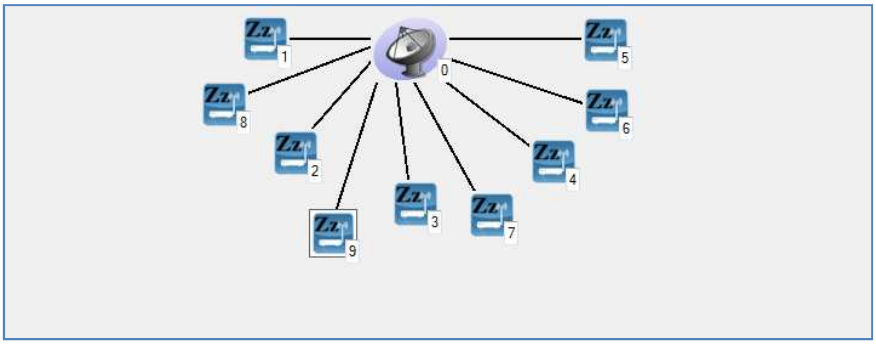
naturaleza y al uso de este tipo de redes, esta no será una situación que se dé con frecuencia.

No obstante, existen situaciones en la que es imposible evitar un árbol degenerado, o al menos una rama degenerada. Es posible que no interese colocar más nodos en una misma dirección, y lo necesario para eliminar la degeneración no es rentable ya que los nodos nuevos a introducir no aportarán mucho al sistema.

Véase el fichero "DEGENERADO.tcl" para ver la simulación de un despertar en ns-2/nam para una configuración de árbol degenerado.

5.3. Configuración en estrella

Una configuración en estrella es aquella configuración en la que todos los nodos se encuentran relativamente cercanos al sumidero raíz, por lo que se opta por minimizar el número de saltos. Esto fuerza al sistema a establecer a todos los nodos como hijos del sumidero y por tanto el camino entre dos nodos sólo suele necesitar el paso por la raíz.



Como ya se ha mencionado, el camino en una configuración en estrella está formado únicamente por la raíz (la cual siempre esta despierta) por lo que el número de nodos afectados por mensaje es siempre dos (emisor y receptor) o uno, si la comunicación es nodo-raíz.

Bajo del punto de vista de la energía utilizada por el sistema: es constante, ya que el número de nodos necesarios para establecer una comunicación es constante, y suele ser un coste bajo al necesitar pocos nodos.

Estudiemos distintas estimaciones de costes de energía para esta configuración, basándonos en algunos de las políticas de wake-up del apartado de Diseño. Por la simplicidad de algunos de ellos, nos vamos a centrar en dos tipos de despertares: despertar 2, en el que un nodo cualquiera despierta a un camino de nodos hasta llegar

a la raíz del sistema – y despertar 3, en el que un nodo cualquiera (incluyendo a la raíz) despierta un camino de nodos hasta llegar a un nodo cualquiera del sistema.

Podemos calcular el coste de cada despertar a partir de dos costes:

1. *El coste energético de del sistema para el despertar – el cual obtenemos con el número de nodos involucrados en el despertar, junto con el coste energético de despertar estos nodos.*
2. *El coste energético de nodos del sistema – el cual obtenemos a partir del número de nodos del sistema involucrados, el coste energético de los mismos por mantenerse despiertos, y el tiempo que se mantienen despiertos.*

Estos costes son entonces comparados con el coste de utilizar a todos los nodos del sistema – cuanto más cercanos los dos costes sean, menos eficiente es el despertar.

Nos centraremos principalmente en calcular el número de nodos involucrados para cada despertar, ya que el resto de parámetros (coste de despertar a un nodo, coste de mantener a un nodo despierto, etc.) son constantes y no están relacionados con la configuración del árbol. El tiempo que los nodos pasan despiertos está relacionado con los mensajes enviados tras el despertar, pero el tema de coste de mensaje se sale de los objetivos del proyecto, ya que no estamos centrando en los costes de despertar. Asumiremos un tiempo constante.

Veamos pues los despertares hablados para un árbol en configuración de estrella:

- **Despertar 2 – Nodo despierta nodos hacia la raíz del sistema:**

  La raíz del sistema se encuentra a un salto de cada nodo, por lo que los costes son bastante constantes: el número de nodos involucrados no cambia, ya que sólo se despierta al nodo emisor, y sólo se mantiene despierto al nodo emisor.

  Llegar a la raíz del sistema sólo requiere un salto, y no requiere establecer ningún tipo de camino. Este despertar es trivial (si no inútil) para este sistema, ya que los nodos pueden simplemente comunicarse con la raíz del sistema sin tener que llevar a cabo ningún wake-up o establecimiento de caminos.

  El coste podría ser el siguiente:

  - ***Coste total del despertar:*** *((Coste de mantener al emisor despierto * tiempo que el emisor pasa despierto) + Coste de despertar al nodo emisor)*

- **Despertar 3 – Nodo despierta nodos hacia un nodo cualquiera del sistema:**

Cualquier nodo del sistema se encuentra a un salto de la raíz, y a otro salto de cualquier otro nodo. El coste de mensaje de este despertar es el coste de despertar como mucho a dos nodos del sistema.

Este despertar requiere como mucho que dos nodos se mantengan despiertos.

No se requiere establecer ningún tipo de camino (ya que la raíz se utilizará como camino y siempre esta despierta). Este despertar es trivial (si no inútil) para este sistema, ya que los nodos pueden simplemente comunicarse con la raíz del sistema sin tener que llevar a cabo ningún wake-up o establecimiento de caminos.

Coste del despertar, considerando a la raíz como un nodo más cuyos costes son 0:

- ***Coste total del despertar:*** *2*((Coste de mantener un nodo despierto * tiempo que pasan los nodos despiertos) + Coste de despertar un nodo)*

Como conclusión, podemos ver que debido a que el *Número de nodos involucrados en los despertares* es casi despreciable comparado con el *Número de nodos totales de la estrella* (sin mencionar que es además un valor casi constante), estos despertares son realmente eficientes en esta configuración. Tan eficientes que podrían considerarse inútiles debido a los pocos despertares que ocurren.

El coste de hacer despertares en esta configuración es constante con el número de nodos. No importa el número de nodos tenga el sistema, el coste de establecer un camino nunca cambiará.

En teoría, esta sería la mejor configuración para una red de sensores de bajo consumo, debido a los pocos despertares y poco consumo de energía del sistema para cada despertar.

No obstante, observemos la realidad: una red con este tipo de configuración requiere que el sumidero alcance a todos los nodos del sistema – esto es algo que puede ocurrir para sistemas muy específicos, pero utilizar una estructura en estrella total no es realmente posible si queremos cubrir una gran dimensión.

Véase el fichero "ESTRELLA.tcl" para ver la simulación de un despertar en ns-2/nam para una configuración de árbol degenerado.

5.4. Comparativa

Asumamos 15 nodos con 175mW de coste de despertar, 175mW de coste de envío de mensaje, y 175mW de coste de recepción. Para establecer un camino, todos los nodos despiertos deben enviar un mensaje a su padre, y el padre por supuesto tendrá que recibirlo. Como ya sabemos, los costes de las operaciones por parte de la raíz son eliminados.

Caso 1: Árbol degenerado

- Coste de despertar para generar un camino hasta la raíz:
    - 14 * 175mW + 14*175mW + 13*175mW = 7175mW

Caso 2: Estrella

- Coste de despertar para generar un camino hasta la raíz:
    - 1 * 175mW + 1*175mW + 0*175mW = 350mW

Caso 3: Árbol binario equilibrado

- Coste de despertar para generar un camino hasta la raíz:
    - 3 * 175mW + 3*175mW + 2*175mW = 1400mW

6. Resultados

Las configuraciones de red estudiadas ofrecen distintas ventajas y desventajas para distintas situaciones. Vamos a recordarlas de forma muy resumida:

- Árbol degenerado
  - ✓ Puede cubrir largas distancias
  - ✓ Es muy sencillo de establecer y mantener
  - × El coste de establecer un camino entre rama y raíz es costoso

- Estrella
  - ✓ Muy eficiente a la hora de establecer caminos. El coste de los mismos es siempre constante
  - × No permite cubrir grandes distancias
  - × Poco realista, no existen muchos casos en los que podamos dejar los nodos tan cerca de la raíz

- Árbol equilibrado
  - ✓ Cubre una buena distancia
  - ✓ Costes de despertares buenos
  - × Costoso de mantener equilibrado
  - × Introducir nuevos nodos al sistema puede requerir reestructurar el árbol

Es posible tratar de utilizar las ventajas de todas ellas para formar una configuración híbrida, la configuración de red buscada por nuestro algoritmo, y por este proyecto en general.

- Configuración Híbrida
  - ✓ Cubre una larga distancia. En una dirección o en varias
  - ✓ Costes de despertares mejores que en árbol equilibrado
  - × Bastante costoso de mantener equilibrado
  - × Introducir nuevos nodos al sistema gasta energía extra. Es posible que no merezca la pena utilizar nuestra configuración si van a introducirse la mayoría de nodos de forma periódica

El objetivo de los algoritmos mencionados en el apartado de Diseño fue siempre la creación de este sistema híbrido – con un algoritmo de adopción basado en distancias y números de hijos, buscamos generar un árbol equilibrado con bastantes cambios y flexibilidad para incluir las ventajas de las distintas configuraciones de red.

Una configuración de red híbrida será aquella que utilice las tres configuraciones de red mencionadas. Podemos conseguir esta configuración de la siguiente manera:

- Permitimos a la raíz que tome más de dos hijos – los nodos alrededor de la raíz tienen forma de estrella, y cada nodo hijo puede considerarse como la raíz de un árbol equilibrado, teniendo en cuenta que estos sub-padres pueden y se dormirán cuando exista inactividad.
    - Es por ello que nuestro algoritmo permite a la raíz adoptar tantos hijos como pueda, así mantenemos la propiedad del árbol en estrella siempre y cuando la distancia lo permita.

- Permitimos que algunas ramas queden degeneradas si es necesario cubrir más distancia. Siempre y cuando el nodo hoja de la rama degenerada no sea demasiado activo, llegar a la raíz no llevará un gran coste durante un largo periodo de tiempo. La degeneración es difícil de solucionar si no colocamos más nodos para dejar la rama equilibrada… pero esto no es siempre algo que queramos. Es posible que colocar más nodos en una dirección sea un mal gasto de los recursos del sistema.
    - Es por ello que el algoritmo adopta cualquier nodo nuevo introducido al sistema. No obstante, tratará de reestructurar el árbol para equilibrarlo, siempre y cuando lo permita el alcance de los nodos.

Conseguimos así un sistema híbrido, cuyo funcionamiento ya fue explicado a lo largo del proyecto, y cuya existencia queda justificada en el análisis de distintas configuraciones de redes de sensores de bajo consumo.

En resumen, nuestro sistema comienza en la configuración de estrella: los nodos más próximos a la raíz serán hijos inmediatos de la misma, y siempre que sea posible, la raíz mantendrá una estrella a su alrededor. Con el paso del tiempo y movido por la introducción de nuevos nodos al sistema, este será capaz de reestructurar partes del mismo para equilibrar las ramas lo mejor que se pueda, si por ejemplo se encuentran padres mejores para nodos del sistema debido a la introducción de un nuevo nodo. El sistema también acepta ramas degeneradas siempre y cuando no quede otro remedio.

Tenemos así un sistema formado por las tres configuraciones de red mencionadas.

7. Conclusiones y futuras ampliaciones

7.1. Conclusiones

Como ya se mencionó en el apartado de objetivos, el objetivo principal del proyecto era diseñar y evaluar una red auto-configurable capaz de alcanzar una estructura y configuración de nodos óptima. Después de ver las distintas configuraciones de red, podemos ver que nuestra red no intenta acoplarse a uno de ellos en particular, sino que es capaz de utilizar las ventajas de las distintas configuraciones de red en árbol.

En este proyecto:

- Se han estudiado distintos tipos de encaminamiento de mensaje en una red inalámbrica de sensores
- Se han estudiado distintas tipos de wake-up
- Se han diseñado y establecido distintas políticas de wake-up
- Se han realizado distintos modelos de ns-2 que fueron luego simulados
- Se ha obtenido la configuración de red híbrida a través del estudio de distintas configuraciones de red

7.2. Futuras ampliaciones

Este proyecto podría ampliarse con la introducción de lo siguiente:

- Implementación real de los nodos diseñados, implementando el algoritmo propuesto .
- Medición de tiempos y costes reales en el sistema en funcionamiento. Estudio de costes y análisis del sistema para distintos parámetros y eventos: número óptimo de hijos, frecuencia de nodos nuevos introducidos al sistema, distancias, etc.

8. Bibliografía y aplicaciones utilizadas

8.1. Bibliografía, fuentes y aplicaciones utilizadas

Para la construcción de la Introducción al proyecto, el objetivo del cual no es otro que introducir el concepto, definición y uso de las redes de sensores, se han utilizado un par de fuentes bibliográficas que serán listados a continuación:

http://www.ece.gatech.edu/research/labs/bwn/surveys/sensornets.pdf

http://en.wikipedia.org/wiki/Wireless_sensor_network

Para la redacción, comparativa y descripción de los simuladores ns-2 y ns-3, se han utilizado ciertos documentos como guía y fuente listados a continuación:

http://isi.edu/nsnam/ns/

http://home.iitk.ac.in/~chebrolu/scourse/slides/ns-tutorial.pdf

http://www.osdp.cs.tsukuba.ac.jp/khoriba/demo.pdf

http://www.ittc.ku.edu/~jpgs/courses/mwnets/lecture-lab-intro2ns3-display.pdf

http://www.cis.upenn.edu/~boonloo/cis505-sp08/ns3.pdf

http://www.nsnam.org/workshops/wns3-2009/ns-3-tutorial-part-1.pdf

http://www.cs.nuim.ie/research/reports/2004/nuim-cs-tr-2004-05.pdf

Para la redacción del resto del proyecto, en particular para el desarrollo de los apartados de diseño, se han utilizado multitud de artículos y escrito. Estos artículos no han sido utilizados como fuentes de información que fue utilizada en el proyecto en sí, sino más bien como fuentes para entender y estudiar el funcionamiento de las redes de sensores, lo cual ha hecho posible la redacción de los apartados mencionados. Se listan algunos de ellos a continuación:

http://www.sensorsmag.com/networking-communications/wireless-sensor-network-topologies-778

http://newslab.cs.wayne.edu/DSAP.pdf

http://k5systems.com/TP0001_v1.pdf

http://nms.lcs.mit.edu/projects/leach/

Para el tema relacionado con ***la interfaz del usuario y la especificación para la construcción de la red***, se ha utilizado ***Microsoft Visual Studio 2008 (C#)*** por su facilidad para programar interfaces gráficas.

Información sobre Visual Studio: http://www.microsoft.com/spain/visualstudio/

Información sobre C#: http://msdn.microsoft.com/en-us/vcsharp/aa336809.aspx

Para la solución de ciertos problemas y para el uso de diversas estructuras de datos y funciones se ha utilizado la biblioteca msdn de Microsoft de Visual Studio C#: http://msdn.microsoft.com/en-us/library

Para el tema relacionado con la ***simulación de la red***, se ha utilizado el simulador de redes ***ns-2***. El simulador ns está escrito en C++ y Python.

Cabe añadir que en un momento dado se consideró el uso de la siguiente versión del simulador ns, ns-3, pero se acabó optando por ns-2.

Información sobre el simulador ns-2: http://www.isi.edu/nsnam/ns/

Información sobre el simulador ns-3: http://www.nsnam.org/

Información sobre C++: http://en.wikipedia.org/wiki/C%2B%2B

Información sobre Python: http://www.python.org/

Puede encontrarse más información y características de cualquiera de los dos simuladores en los apartados relacionados con la simulación de la red de este proyecto.

Para la solución de diversas dudas y problemas se ha utilizado tanto el tutorial de ns, el cual puede accederse a través de la página principal de ns-2 ya mencionada (http://www.isi.edu/nsnam/ns/).

Para el tema relacionado con ***la visualización de la simulación de la red***, se ha utilizado ***nam*****,** el visualizador de redes del simulador ns-2, por su alta integración con scripts y simulaciones de ns-2, y su facilidad de uso.

Información sobre nam: http://www.isi.edu/nsnam/nam/